

# **MCF51JM128 ColdFire® Integrated Microcontroller Reference Manual**

## **Devices Supported:**

**MCF51JM128  
MCF51JM64  
MCF51JM32**

Document Number: MCF51JM128RM  
Rev. 2  
6/2009

## **How to Reach Us:**

### **Home Page:**

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

### **Web Support:**

<http://www.freescale.com/support>

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

Freescale Semiconductor, Inc.  
Technical Information Center, EL516  
2100 East Elliot Road  
Tempe, Arizona 85284  
1-800-521-6274 or +1-480-768-2130  
[www.freescale.com/support](http://www.freescale.com/support)

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

### **Japan:**

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

Freescale Semiconductor Literature Distribution Center  
P.O. Box 5405  
Denver, Colorado 80217  
1-800-441-2447 or +1-303-675-2140  
Fax: +1-303-675-2150  
[LDCForFreescaleSemiconductor@hibbertgroup.com](mailto:LDCForFreescaleSemiconductor@hibbertgroup.com)

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

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

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

© Freescale Semiconductor, Inc. 2008-2009. All rights reserved.

MCF51JM128RM  
Rev. 2  
6/2009

## Chapter 1 Device Overview

1.1	The MCF51JM128 Series Microcontrollers . . . . .	1-1
1.1.1	Definition . . . . .	1-1
1.1.2	MCF51JM128 Series Devices . . . . .	1-1
1.1.3	MCF51JM128 Series Device Comparison . . . . .	1-2
1.2	MCF51JM128 Series Block Diagram . . . . .	1-3
1.2.1	Block Diagram . . . . .	1-3
1.2.2	Functional Units . . . . .	1-4
1.3	V1 ColdFire Core . . . . .	1-5
1.3.1	User Programming Model . . . . .	1-5
1.3.2	Supervisor Programming Model . . . . .	1-5
1.4	System Clock Generation and Distribution . . . . .	1-6
1.4.1	Clock Distribution Diagram . . . . .	1-6
1.4.2	System Clocks . . . . .	1-7
1.4.3	Clock Gating . . . . .	1-7
1.4.4	MCG Modes of Operation . . . . .	1-7
1.4.5	MCG Mode State Diagram . . . . .	1-8

## Chapter 2 Pins and Connections

2.1	Device Pin Assignment . . . . .	2-1
2.1.1	Pinout: 80-Pin LQFP . . . . .	2-1
2.1.2	Pinout: 64-Pin LQFP and QFP . . . . .	2-2
2.1.3	Pinout: 44-Pin LQFP . . . . .	2-3
2.1.4	Pin Assignments . . . . .	2-4
2.2	Recommended System Connections . . . . .	2-6
2.2.1	Power . . . . .	2-8
2.2.2	Oscillator . . . . .	2-8
2.2.3	RESET . . . . .	2-9
2.2.4	IRQ/TPMCLK . . . . .	2-9
2.2.5	Background / Mode Select (BKGD/MS) . . . . .	2-9
2.2.6	ADC Reference Pins ( $V_{REFH}$ , $V_{REFL}$ ) . . . . .	2-10
2.2.7	USB Data Pins (USBDP, USBDN) . . . . .	2-10
2.2.8	General-Purpose I/O and Peripheral Ports . . . . .	2-10

## Chapter 3 Modes of Operation

3.1	Introduction . . . . .	3-1
3.2	Features . . . . .	3-1
3.3	Overview . . . . .	3-2
3.4	Debug Mode . . . . .	3-3
3.5	Secure Mode . . . . .	3-4
3.6	Run Mode . . . . .	3-4
3.7	Wait Mode . . . . .	3-4
3.8	Stop Modes . . . . .	3-4
3.8.1	Stop2 Mode . . . . .	3-5

3.8.2	Stop3 Mode . . . . .	3-6
3.8.3	Stop4 Mode . . . . .	3-6
	3.8.3.1 LVD Enabled in Stop Mode . . . . .	3-6
3.9	On-Chip Peripheral Modules in Stop and Wait Modes . . . . .	3-6

## Chapter 4 Memory

4.1	MCF51JM128 Series Memory Map . . . . .	4-1
4.2	Register Addresses and Bit Assignments . . . . .	4-2
	4.2.1 Flash Module Reserved Memory Locations . . . . .	4-15
	4.2.2 ColdFire Rapid GPIO Memory Map . . . . .	4-17
	4.2.3 ColdFire Interrupt Controller Memory Map . . . . .	4-17
4.3	RAM . . . . .	4-19
4.4	Flash Memory . . . . .	4-19
	4.4.1 Features . . . . .	4-20
	4.4.2 Register Descriptions . . . . .	4-20
	4.4.2.1 Flash Clock Divider Register (FCDIV) . . . . .	4-20
	4.4.2.2 Flash Options Register (FOPT and NVOPT) . . . . .	4-21
	4.4.2.3 Flash Configuration Register (FCNFG) . . . . .	4-22
	4.4.2.4 Flash Protection Register (FPROT and NVPROT) . . . . .	4-22
	4.4.2.5 Flash Status Register (FSTAT) . . . . .	4-24
	4.4.2.6 Flash Command Register (FCMD) . . . . .	4-25
4.5	Function Description . . . . .	4-26
	4.5.1 Flash Command Operations . . . . .	4-26
	4.5.1.1 Writing the FCDIV Register . . . . .	4-26
	4.5.1.2 Command Write Sequence . . . . .	4-27
	4.5.2 Flash Commands . . . . .	4-28
	4.5.2.1 Erase Verify Command . . . . .	4-28
	4.5.2.2 Program Command . . . . .	4-30
	4.5.2.3 Burst Program Command . . . . .	4-31
	4.5.2.4 Sector Erase Command . . . . .	4-32
	4.5.2.5 Mass Erase Command . . . . .	4-34
	4.5.3 Illegal Flash Operations . . . . .	4-35
	4.5.3.1 Flash Access Violations . . . . .	4-35
	4.5.3.2 Flash Protection Violations . . . . .	4-35
	4.5.4 Operating Modes . . . . .	4-36
	4.5.4.1 Wait Mode . . . . .	4-36
	4.5.4.2 Stop Modes . . . . .	4-36
	4.5.4.3 Background Debug Mode . . . . .	4-36
	4.5.5 Security . . . . .	4-36
	4.5.5.1 Unsecuring the MCU using Backdoor Key Access . . . . .	4-37
	4.5.6 Resets . . . . .	4-38
	4.5.6.1 Flash Reset Sequence . . . . .	4-38
	4.5.6.2 Reset While Flash Command Active . . . . .	4-38
	4.5.6.3 Program and Erase Times . . . . .	4-38
4.6	Security . . . . .	4-38

## 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 (ILOP)	5-3
5.3.3	Illegal Address Detect (ILAD)	5-3
5.4	Interrupts and Exceptions	5-3
5.4.1	External Interrupt Request (IRQ) Pin	5-4
5.4.1.1	Pin Configuration Options	5-4
5.4.1.2	Edge and Level Sensitivity	5-5
5.4.2	Interrupt Vectors, Sources, and Local Masks	5-5
5.4.3	Interrupt Request Level and Priority Assignments	5-8
5.5	Low-Voltage Detect (LVD) System	5-9
5.5.1	Power-On Reset Operation	5-9
5.5.2	LVD Reset Operation	5-10
5.5.3	Low-Voltage Warning (LVW) Interrupt Operation	5-10
5.6	Peripheral Clock Gating	5-10
5.7	Reset, Interrupt, and System Control Registers and Control Bits	5-10
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 (SOPT1) Register	5-13
5.7.4	System Options 2 (SOPT2) Register	5-14
5.7.5	System Device Identification Register (SDIDH, SDIDL)	5-15
5.7.6	System Power Management Status and Control 1 Register (SPMSC1)	5-16
5.7.7	System Power Management Status and Control 2 Register (SPMSC2)	5-17
5.7.8	System Clock Gating Control 1 Register (SCGC1)	5-18
5.7.9	System Clock Gating Control 2 Register (SCGC2)	5-19
5.7.10	System Clock Gating Control 3 Register (SCGC3)	5-20
5.7.11	System Options 3 Register (SOPT3)	5-21
5.7.12	System Options 4 Register (SOPT4)	5-21

## Chapter 6

### ColdFire Core

6.1	Introduction	6-1
6.1.1	Overview	6-1
6.2	Memory Map/Register Description	6-2
6.2.1	Data Registers (D0–D7)	6-3
6.2.2	Address Registers (A0–A6)	6-4
6.2.3	Supervisor/User Stack Pointers (A7 and OTHER_A7)	6-4
6.2.4	Condition Code Register (CCR)	6-5
6.2.5	Program Counter (PC)	6-6
6.2.6	Vector Base Register (VBR)	6-6
6.2.7	CPU Configuration Register (CPUCR)	6-7
6.2.8	Status Register (SR)	6-8
6.3	Functional Description	6-9

6.3.1	Instruction Set Architecture (ISA_C) . . . . .	6-9
6.3.2	Exception Processing Overview . . . . .	6-10
6.3.2.1	Exception Stack Frame Definition . . . . .	6-12
6.3.2.2	S08 and ColdFire Exception Processing Comparison . . . . .	6-13
6.3.3	Processor Exceptions . . . . .	6-15
6.3.3.1	Access Error Exception . . . . .	6-15
6.3.3.2	Address Error Exception . . . . .	6-15
6.3.3.3	Illegal Instruction Exception . . . . .	6-16
6.3.3.4	Privilege Violation . . . . .	6-17
6.3.3.5	Trace Exception . . . . .	6-17
6.3.3.6	Unimplemented Line-A Opcode . . . . .	6-18
6.3.3.7	Unimplemented Line-F Opcode . . . . .	6-18
6.3.3.8	Debug Interrupt . . . . .	6-18
6.3.3.9	RTE and Format Error Exception . . . . .	6-18
6.3.3.10	TRAP Instruction Exception . . . . .	6-19
6.3.3.11	Unsupported Instruction Exception . . . . .	6-19
6.3.3.12	Interrupt Exception . . . . .	6-19
6.3.3.13	Fault-on-Fault Halt . . . . .	6-19
6.3.3.14	Reset Exception . . . . .	6-20
6.3.4	Instruction Execution Timing . . . . .	6-23
6.3.4.1	Timing Assumptions . . . . .	6-23
6.3.4.2	MOVE Instruction Execution Times . . . . .	6-24
6.3.4.3	Standard One Operand Instruction Execution Times . . . . .	6-25
6.3.4.4	Standard Two Operand Instruction Execution Times . . . . .	6-26
6.3.4.5	Miscellaneous Instruction Execution Times . . . . .	6-27
6.3.4.6	Branch Instruction Execution Times . . . . .	6-28

## Chapter 7

### Multipurpose Clock Generator (S08MCGV3)

7.1	Introduction . . . . .	7-1
7.1.1	Features . . . . .	7-2
7.1.2	Modes of Operation . . . . .	7-4
7.2	External Signal Description . . . . .	7-4
7.3	Register Definition . . . . .	7-4
7.3.1	MCG Control Register 1 (MCGC1) . . . . .	7-4
7.3.2	MCG Control Register 2 (MCGC2) . . . . .	7-6
7.3.3	MCG Trim Register (MCGTRM) . . . . .	7-7
7.3.4	MCG Status and Control Register (MCGSC) . . . . .	7-8
7.3.5	MCG Control Register 3 (MCGC3) . . . . .	7-9
7.3.6	MCG Control Register 4 (MCGC4) . . . . .	7-10
7.3.7	MCG Test Register (MCGT) . . . . .	7-11
7.4	Functional Description . . . . .	7-12
7.4.1	MCG Modes of Operation . . . . .	7-12
7.4.2	MCG Mode State Diagram . . . . .	7-14
7.4.3	Mode Switching . . . . .	7-14
7.4.4	Bus Frequency Divider . . . . .	7-15
7.4.5	Low Power Bit Usage . . . . .	7-15
7.4.6	Internal Reference Clock . . . . .	7-15

7.4.7	External Reference Clock .....	7-15
7.4.8	Fixed Frequency Clock .....	7-16
7.5	Initialization / Application Information .....	7-17
7.5.1	MCG Module Initialization Sequence .....	7-17
7.5.1.1	Initializing the MCG .....	7-17
7.5.2	Using a 32.768 kHz Reference .....	7-19
7.5.3	MCG Mode Switching .....	7-19
7.5.3.1	Example 1: Moving from FEI to PEE Mode: External Crystal = 8 MHz, Bus Frequency = 16 MHz .....	7-20
7.5.3.2	Example 2: Moving from PEE to BLPI Mode: Bus Frequency =16 kHz .....	7-24
7.5.3.3	Example 3: Moving from BLPI to FEE Mode: External Crystal = 8 MHz, Bus Frequency = 16 MHz .....	7-27
7.5.4	Calibrating the Internal Reference Clock (IRC) .....	7-28
7.5.4.1	Example 4: Internal Reference Clock Trim .....	7-29

## Chapter 8 Interrupt Controller (CF1\_INTC)

8.1	Introduction .....	8-1
8.1.1	Overview .....	8-2
8.1.2	Features .....	8-4
8.1.3	Modes of Operation .....	8-5
8.2	External Signal Description .....	8-5
8.3	Memory Map/Register Definition .....	8-5
8.3.1	INTC Force Interrupt Register (INTC_FRC) .....	8-6
8.3.2	INTC Programmable Level 6, Priority {7,6} Registers (INTC_PL6P{7,6}) .....	8-7
8.3.3	INTC Wakeup Control Register (INTC_WCR) .....	8-8
8.3.4	INTC Set Interrupt Force Register (INTC_SFRC) .....	8-9
8.3.5	INTC Clear Interrupt Force Register (INTC_CFRC) .....	8-10
8.3.6	INTC Software and Level- <i>n</i> IACK Registers ( <i>n</i> = 1,2,3,...,7) .....	8-11
8.4	Functional Description .....	8-12
8.4.1	Handling of Non-Maskable Level 7 Interrupt Requests .....	8-12
8.5	Initialization Information .....	8-12
8.6	Application Information .....	8-12
8.6.1	Emulation of the HCS08's 1-Level IRQ Handling .....	8-12
8.6.2	Using INTC_PL6P{7,6} Registers .....	8-13
8.6.3	More on Software IACKs .....	8-14

## Chapter 9 Parallel Input/Output Control

9.1	Port Data and Data Direction .....	9-1
9.2	Pull-up, Slew Rate, and Drive Strength .....	9-2
9.2.1	Port Internal Pull-up Enable .....	9-2
9.2.2	Port Slew Rate Enable .....	9-2
9.2.3	Port Drive Strength Select .....	9-3
9.3	V1 ColdFire Rapid GPIO Functionality .....	9-3
9.4	Keyboard Interrupts .....	9-3
9.4.1	Edge Only Sensitivity .....	9-4

9.4.2	Edge and Level Sensitivity . . . . .	9-4
9.4.3	Pull-up/Pull-down Resistors . . . . .	9-4
9.4.4	Keyboard Interrupt Initialization . . . . .	9-4
9.5	Pin Behavior in Stop Modes . . . . .	9-4
9.6	Parallel I/O, Keyboard Interrupt, and Pin Control Registers . . . . .	9-5
9.6.1	Port A Registers . . . . .	9-5
9.6.1.1	Port A Data Register (PTAD) . . . . .	9-5
9.6.1.2	Port A Data Direction Register (PTADD) . . . . .	9-6
9.6.1.3	Port A Pull Enable Register (PTAPE) . . . . .	9-6
9.6.1.4	Port A Slew Rate Enable Register (PTASE) . . . . .	9-6
9.6.1.5	Port A Drive Strength Selection Register (PTADS) . . . . .	9-7
9.6.1.6	Port A Input Filter Enable Register (PTAIFE) . . . . .	9-7
9.6.2	Port B Registers . . . . .	9-8
9.6.2.1	Port B Data Register (PTBD) . . . . .	9-8
9.6.2.2	Port B Data Direction Register (PTBDD) . . . . .	9-8
9.6.2.3	Port B Pull Enable Register (PTBPE) . . . . .	9-9
9.6.2.4	Port B Slew Rate Enable Register (PTBSE) . . . . .	9-9
9.6.2.5	Port B Drive Strength Selection Register (PTBDS) . . . . .	9-9
9.6.2.6	Port B Input Filter Enable Register (PTBIFE) . . . . .	9-10
9.6.3	Port C Registers . . . . .	9-10
9.6.3.1	Port C Data Register (PTCD) . . . . .	9-10
9.6.3.2	Port C Data Direction Register (PTCDD) . . . . .	9-11
9.6.3.3	Port C Pull Enable Register (PTCPE) . . . . .	9-11
9.6.3.4	Port C Slew Rate Enable Register (PTCSE) . . . . .	9-12
9.6.3.5	Port C Drive Strength Selection Register (PTCDS) . . . . .	9-12
9.6.3.6	Port C Input Filter Enable Register (PTCIFE) . . . . .	9-12
9.6.4	Port D Registers . . . . .	9-13
9.6.4.1	Port D Data Register (PTDD) . . . . .	9-13
9.6.4.2	Port D Data Direction Register (PTDDD) . . . . .	9-13
9.6.4.3	Port D Pull Enable Register (PTDPE) . . . . .	9-14
9.6.4.4	Port D Slew Rate Enable Register (PTDSE) . . . . .	9-14
9.6.4.5	Port D Drive Strength Selection Register (PTDDS) . . . . .	9-15
9.6.4.6	Port D Input Filter Enable Register (PTDIFE) . . . . .	9-15
9.6.5	Port E Registers . . . . .	9-15
9.6.5.1	Port E Data Register (PTED) . . . . .	9-16
9.6.5.2	Port E Data Direction Register (PTEDD) . . . . .	9-16
9.6.5.3	Port E Pull Enable Register (PTEPE) . . . . .	9-16
9.6.5.4	Port E Slew Rate Enable Register (PTESE) . . . . .	9-17
9.6.5.5	Port E Drive Strength Selection Register (PTEDS) . . . . .	9-17
9.6.5.6	Port E Input Filter Enable Register (PTEIFE) . . . . .	9-17
9.6.6	Port F Registers . . . . .	9-18
9.6.6.1	Port F Data Register (PTFD) . . . . .	9-18
9.6.6.2	Port F Data Direction Register (PTFDD) . . . . .	9-19
9.6.6.3	Port F Pull Enable Register (PTFPE) . . . . .	9-19
9.6.6.4	Port F Slew Rate Enable Register (PTFSE) . . . . .	9-19
9.6.6.5	Port F Drive Strength Selection Register (PTFDS) . . . . .	9-20
9.6.6.6	Port F Input Filter Enable Register (PTFIFE) . . . . .	9-20
9.6.7	Port G Registers . . . . .	9-21
9.6.7.1	Port G Data Register (PTGD) . . . . .	9-21

9.6.7.2	Port G Data Direction Register (PTGDD) . . . . .	9-21
9.6.7.3	Port G Pull Enable Register (PTGPE) . . . . .	9-22
9.6.7.4	Port G Slew Rate Enable Register (PTGSE) . . . . .	9-22
9.6.7.5	Port G Drive Strength Selection Register (PTGDS) . . . . .	9-22
9.6.7.6	Port G Input Filter Enable Register (PTGIFE) . . . . .	9-23
9.6.8	Port H Registers . . . . .	9-23
9.6.8.1	Port H Data Register (PTHD) . . . . .	9-23
9.6.8.2	Port H Data Direction Register (PTHDD) . . . . .	9-24
9.6.8.3	Port H Pull Enable Register (PTHPE) . . . . .	9-24
9.6.8.4	Port H Slew Rate Enable Register (PTHSE) . . . . .	9-25
9.6.8.5	Port H Drive Strength Selection Register (PTHDS) . . . . .	9-25
9.6.8.6	Port H Input Filter Enable Register (PTHIFE) . . . . .	9-25
9.6.9	Port J Registers . . . . .	9-26
9.6.9.1	Port J Data Register (PTJD) . . . . .	9-26
9.6.9.2	Port J Data Direction Register (PTJDD) . . . . .	9-26
9.6.9.3	Port J Pull Enable Register (PTJPE) . . . . .	9-27
9.6.9.4	Port J Slew Rate Enable Register (PTJSE) . . . . .	9-27
9.6.9.5	Port J Drive Strength Selection Register (PTJDS) . . . . .	9-28
9.6.9.6	Port J Input Filter Enable Register (PTJIFE) . . . . .	9-28
9.6.10	Keyboard Interrupt 1 (KBI1) Registers . . . . .	9-28
9.6.10.1	KBI1 Interrupt Status and Control Register (KBI1SC) . . . . .	9-29
9.6.10.2	KBI1 Interrupt Pin Select Register (KBI1PE) . . . . .	9-29
9.6.10.3	KBI1 Interrupt Edge Select Register (KBI1ES) . . . . .	9-30

## 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	RGPIO Data Direction (RGPIO_DIR) . . . . .	10-4
10.3.2	RGPIO Data (RGPIO_DATA) . . . . .	10-5
10.3.3	RGPIO Pin Enable (RGPIO_ENB) . . . . .	10-6
10.3.4	RGPIO Clear Data (RGPIO_CLR) . . . . .	10-6
10.3.5	RGPIO Set Data (RGPIO_SET) . . . . .	10-7
10.3.6	RGPIO Toggle Data (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 . . . . .	10-8
10.6.2	Application 2: 16-bit Message Transmission using SPI Protocol . . . . .	10-9

# Chapter 11

## Freescale's Controller Area Network (MSCAN)

11.1	Introduction . . . . .	11-1
11.1.1	Features . . . . .	11-1
11.1.2	Modes of Operation . . . . .	11-2
11.1.3	Block Diagram . . . . .	11-2
11.2	External Signal Description . . . . .	11-2
11.2.1	RXCAN — CAN Receiver Input Pin . . . . .	11-2
11.2.2	TXCAN — CAN Transmitter Output Pin . . . . .	11-3
11.2.3	CAN System . . . . .	11-3
11.3	Register Definition . . . . .	11-3
11.3.1	MSCAN Control Register 0 (CANCTL0) . . . . .	11-3
11.3.2	MSCAN Control Register 1 (CANCTL1) . . . . .	11-6
11.3.3	MSCAN Bus Timing Register 0 (CANBTR0) . . . . .	11-7
11.3.4	MSCAN Bus Timing Register 1 (CANBTR1) . . . . .	11-8
11.3.5	MSCAN Receiver Flag Register (CANRFLG) . . . . .	11-9
11.3.6	MSCAN Receiver Interrupt Enable Register (CANRIER) . . . . .	11-11
11.3.7	MSCAN Transmitter Flag Register (CANTFLG) . . . . .	11-12
11.3.8	MSCAN Transmitter Interrupt Enable Register (CANTIER) . . . . .	11-13
11.3.9	MSCAN Transmitter Message Abort Request Register (CANTARQ) . . . . .	11-14
11.3.10	MSCAN Transmitter Message Abort Acknowledge Register (CANTAAK) . . . . .	11-15
11.3.11	MSCAN Transmit Buffer Selection Register (CANTBSEL) . . . . .	11-15
11.3.12	MSCAN Identifier Acceptance Control Register (CANIDAC) . . . . .	11-16
11.3.13	MSCAN Miscellaneous Register (CANMISC) . . . . .	11-17
11.3.14	MSCAN Receive Error Counter (CANRXERR) . . . . .	11-18
11.3.15	MSCAN Transmit Error Counter (CANTXERR) . . . . .	11-18
11.3.16	MSCAN Identifier Acceptance Registers (CANIDAR0-7) . . . . .	11-19
11.3.17	MSCAN Identifier Mask Registers (CANIDMR0–CANIDMR7) . . . . .	11-20
11.4	Programmer's Model of Message Storage . . . . .	11-21
11.4.1	Identifier Registers (IDR0–IDR3) . . . . .	11-24
11.4.1.1	IDR0–IDR3 for Extended Identifier Mapping . . . . .	11-24
11.4.2	IDR0–IDR3 for Standard Identifier Mapping . . . . .	11-26
11.4.3	Data Segment Registers (DSR0-7) . . . . .	11-27
11.4.4	Data Length Register (DLR) . . . . .	11-28
11.4.5	Transmit Buffer Priority Register (TBPR) . . . . .	11-29
11.4.6	Time Stamp Register (TSRH–TSRL) . . . . .	11-29
11.5	Functional Description . . . . .	11-30
11.5.1	General . . . . .	11-30
11.5.2	Message Storage . . . . .	11-31
11.5.2.1	Message Transmit Background . . . . .	11-32
11.5.2.2	Transmit Structures . . . . .	11-32
11.5.2.3	Receive Structures . . . . .	11-33
11.5.2.4	Identifier Acceptance Filter . . . . .	11-34
11.5.2.5	Identifier Acceptance Filters example . . . . .	11-38
11.5.2.6	Protocol Violation Protection . . . . .	11-39
11.5.2.7	Clock System . . . . .	11-39
11.5.3	Timer Link . . . . .	11-41
11.5.4	Modes of Operation . . . . .	11-42

11.5.4.1	Normal Modes . . . . .	11-42
11.5.4.2	Special Test Modes . . . . .	11-42
11.5.4.3	Emulation Modes . . . . .	11-42
11.5.4.4	Listen-Only Mode . . . . .	11-42
11.5.4.5	Security Modes . . . . .	11-42
11.5.4.6	Loopback Self Test Mode . . . . .	11-42
11.5.5	Low-Power Options . . . . .	11-42
11.5.5.1	Operation in Run Mode . . . . .	11-43
11.5.5.2	Operation in Wait Mode . . . . .	11-43
11.5.5.3	Operation in Stop Mode . . . . .	11-43
11.5.5.4	MSCAN Sleep Mode . . . . .	11-44
11.5.5.5	MSCAN Initialization Mode . . . . .	11-45
11.5.5.6	MSCAN Power Down Mode . . . . .	11-46
11.5.5.7	Programmable Wake-Up Function . . . . .	11-47
11.5.6	Reset Initialization . . . . .	11-47
11.5.7	Interrupts . . . . .	11-47
11.5.7.1	Description of Interrupt Operation . . . . .	11-47
11.5.7.2	Transmit Interrupt . . . . .	11-48
11.5.7.3	Receive Interrupt . . . . .	11-48
11.5.7.4	Wake-Up Interrupt . . . . .	11-48
11.5.7.5	Error Interrupt . . . . .	11-48
11.5.7.6	Interrupt Acknowledge . . . . .	11-48
11.5.7.7	Recovery from Stop or Wait . . . . .	11-49
11.6	Initialization/Application Information . . . . .	11-49
11.6.1	MSCAN initialization . . . . .	11-49
11.6.2	Bus-Off Recovery . . . . .	11-49

## Chapter 12

### Serial Communication Interface (S08SCIV4)

12.1	Introduction . . . . .	12-1
12.1.1	Features . . . . .	12-2
12.1.2	Modes of Operation . . . . .	12-2
12.1.3	Block Diagram . . . . .	12-3
12.2	Register Definition . . . . .	12-5
12.2.1	SCI Baud Rate Registers (SCIxBDH, SCIxBDL) . . . . .	12-5
12.2.2	SCI Control Register 1 (SCIxC1) . . . . .	12-6
12.2.3	SCI Control Register 2 (SCIxC2) . . . . .	12-7
12.2.4	SCI Status Register 1 (SCIxS1) . . . . .	12-8
12.2.5	SCI Status Register 2 (SCIxS2) . . . . .	12-10
12.2.6	SCI Control Register 3 (SCIxC3) . . . . .	12-11
12.2.7	SCI Data Register (SCIxD) . . . . .	12-12
12.3	Functional Description . . . . .	12-12
12.3.1	Baud Rate Generation . . . . .	12-12
12.3.2	Transmitter Functional Description . . . . .	12-13
12.3.2.1	Send Break and Queued Idle . . . . .	12-14
12.3.3	Receiver Functional Description . . . . .	12-14
12.3.3.1	Data Sampling Technique . . . . .	12-15
12.3.3.2	Receiver Wakeup Operation . . . . .	12-15

12.3.4	Interrupts and Status Flags . . . . .	12-16
12.3.5	Additional SCI Functions . . . . .	12-17
12.3.5.1	8- and 9-Bit Data Modes . . . . .	12-17
12.3.5.2	Stop Mode Operation . . . . .	12-18
12.3.5.3	Loop Mode . . . . .	12-18
12.3.5.4	Single-Wire Operation . . . . .	12-18

## Chapter 13

### Inter-Integrated Circuit (S08IICV2)

13.1	Introduction . . . . .	13-1
13.1.1	Features . . . . .	13-2
13.1.2	Modes of Operation . . . . .	13-2
13.1.3	Block Diagram . . . . .	13-3
13.2	External Signal Description . . . . .	13-3
13.2.1	SCL — Serial Clock Line . . . . .	13-3
13.2.2	SDA — Serial Data Line . . . . .	13-3
13.3	Register Definition . . . . .	13-3
13.3.1	IIC Address Register (IICxA) . . . . .	13-4
13.3.2	IIC Frequency Divider Register (IICxF) . . . . .	13-4
13.3.3	IIC Control Register (IICxC1) . . . . .	13-7
13.3.4	IIC Status Register (IICxS) . . . . .	13-8
13.3.5	IIC Data I/O Register (IICxD) . . . . .	13-9
13.3.6	IIC Control Register 2 (IICxC2) . . . . .	13-9
13.4	Functional Description . . . . .	13-10
13.4.1	IIC Protocol . . . . .	13-10
13.4.1.1	Start Signal . . . . .	13-11
13.4.1.2	Slave Address Transmission . . . . .	13-11
13.4.1.3	Data Transfer . . . . .	13-12
13.4.1.4	Stop Signal . . . . .	13-12
13.4.1.5	Repeated Start Signal . . . . .	13-12
13.4.1.6	Arbitration Procedure . . . . .	13-12
13.4.1.7	Clock Synchronization . . . . .	13-13
13.4.1.8	Handshaking . . . . .	13-13
13.4.1.9	Clock Stretching . . . . .	13-13
13.4.2	10-bit Address . . . . .	13-14
13.4.2.1	Master-Transmitter Addresses a Slave-Receiver . . . . .	13-14
13.4.2.2	Master-Receiver Addresses a Slave-Transmitter . . . . .	13-14
13.4.3	General Call Address . . . . .	13-15
13.5	Resets . . . . .	13-15
13.6	Interrupts . . . . .	13-15
13.6.1	Byte Transfer Interrupt . . . . .	13-15
13.6.2	Address Detect Interrupt . . . . .	13-15
13.6.3	Arbitration Lost Interrupt . . . . .	13-15
13.7	Initialization/Application Information . . . . .	13-17

## Chapter 14

### Serial Peripheral Interface (SPI)

14.1	Introduction . . . . .	14-1
14.1.1	Features . . . . .	14-1
14.1.2	Modes of Operation . . . . .	14-1
14.1.3	Block Diagrams . . . . .	14-2
14.1.3.1	SPI System Block Diagram . . . . .	14-2
14.1.3.2	SPI Module Block Diagram . . . . .	14-3
14.2	External Signal Description . . . . .	14-4
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	Register Definition . . . . .	14-5
14.3.1	SPI Control Register 1 (SPIxC1) . . . . .	14-5
14.3.2	SPI Control Register 2 (SPIxC2) . . . . .	14-7
14.3.3	SPI Baud Rate Register (SPIxBR) . . . . .	14-8
14.3.4	SPI Status Register (SPIxS) . . . . .	14-10
14.3.5	SPI Data Registers (SPIxDH:SPIxDL) . . . . .	14-13
14.3.6	SPI Match Registers (SPIxMH:SPIxML) . . . . .	14-13
14.3.7	SPI Control Register 3 (SPIxC3) — Enable FIFO Feature . . . . .	14-14
14.4	Functional Description . . . . .	14-15
14.4.1	General . . . . .	14-15
14.4.2	Master Mode . . . . .	14-16
14.4.3	Slave Mode . . . . .	14-17
14.4.4	SPI FIFO MODE . . . . .	14-19
14.4.5	Data Transmission Length . . . . .	14-20
14.4.6	SPI Clock Formats . . . . .	14-20
14.4.7	SPI Baud Rate Generation . . . . .	14-22
14.4.8	Special Features . . . . .	14-23
14.4.8.1	SS Output . . . . .	14-23
14.4.8.2	Bidirectional Mode (MOMI or SISO) . . . . .	14-23
14.4.9	Error Conditions . . . . .	14-24
14.4.9.1	Mode Fault Error . . . . .	14-24
14.4.10	Low Power Mode Options . . . . .	14-25
14.4.10.1	SPI in Run Mode . . . . .	14-25
14.4.10.2	SPI in Wait Mode . . . . .	14-25
14.4.10.3	SPI in Stop Mode . . . . .	14-26
14.4.10.4	Reset . . . . .	14-26
14.4.10.5	TNEAREF . . . . .	14-26
14.4.10.6	RNFULLF . . . . .	14-26
14.4.10.7	Interrupts . . . . .	14-27
14.4.11	SPI Interrupts . . . . .	14-27
14.4.11.1	MODF . . . . .	14-27
14.4.11.2	SPRF . . . . .	14-27
14.4.11.3	SPTEF . . . . .	14-28
14.4.11.4	SPMF . . . . .	14-28
14.5	Initialization/Application Information . . . . .	14-28

14.5.1 SPI Module Initialization Example . . . . .	14-28
14.5.1.1 Initialization Sequence . . . . .	14-28
14.5.1.2 Pseudo—Code Example . . . . .	14-28

## Chapter 15 Carrier Modulator Timer (CMT)

15.1 Features . . . . .	15-1
15.2 CMT Block Diagram . . . . .	15-1
15.3 Pin Description . . . . .	15-2
15.4 CMT Pad Configuration . . . . .	15-2
15.5 Functional Description . . . . .	15-2
15.5.1 Carrier Generator . . . . .	15-3
15.5.2 Modulator . . . . .	15-5
15.5.2.1 Time Mode . . . . .	15-6
15.5.2.2 Baseband Mode . . . . .	15-7
15.5.2.3 FSK Mode . . . . .	15-7
15.5.3 Extended Space Operation . . . . .	15-8
15.5.3.1 EXSPC Operation in Time Mode . . . . .	15-8
15.5.3.2 EXSPC Operation in FSK Mode . . . . .	15-9
15.5.4 Transmitter . . . . .	15-9
15.5.5 CMT Interrupts . . . . .	15-9
15.5.6 Wait Mode Operation . . . . .	15-10
15.5.7 Stop Mode Operation . . . . .	15-10
15.5.8 Background Mode Operation . . . . .	15-10
15.6 CMT Registers and Control Bits . . . . .	15-11
15.6.1 Carrier Generator Data Registers (CMTCGH1, CMTCGL1, CMTCGH2, and CMTCGL2) . . . . .	15-11
15.6.2 CMT Output Control Register (CMTOC) . . . . .	15-13
15.6.3 CMT Modulator Status and Control Register (CMTMSC) . . . . .	15-14
15.6.4 CMT Modulator Data Registers (CMTCMD1, CMTCMD2, CMTCMD3, and CMTCMD4) . . . . .	15-15

## Chapter 16 Universal Serial Bus, OTG Capable Controller

16.1 Introduction . . . . .	16-1
16.1.1 USB . . . . .	16-1
16.1.2 USB On-The-Go . . . . .	16-3
16.1.3 USB-FS Features . . . . .	16-4
16.1.4 Modes of Operation . . . . .	16-4
16.1.4.1 Run Mode . . . . .	16-4
16.1.4.2 Wait Mode . . . . .	16-4
16.1.4.3 Stop Mode . . . . .	16-4
16.2 External Signal Description . . . . .	16-4
16.2.1 USB Pull-up/Pull-down Connections . . . . .	16-4
16.2.2 USB OTG Connections . . . . .	16-5
16.3 Functional Description . . . . .	16-6
16.3.1 Data Structures . . . . .	16-6

16.4	Programmers Interface . . . . .	16-6
16.4.1	Buffer Descriptor Table . . . . .	16-6
16.4.2	Rx vs. Tx as a USB Target Device or USB Host . . . . .	16-7
16.4.3	Addressing Buffer Descriptor Table Entries . . . . .	16-8
16.4.4	Buffer Descriptor Formats . . . . .	16-9
16.4.5	USB Transaction . . . . .	16-10
16.5	Memory Map/Register Definitions . . . . .	16-12
16.5.1	Capability Registers . . . . .	16-13
16.5.1.1	Peripheral ID Register (PER_ID) . . . . .	16-14
16.5.1.2	Peripheral ID Complement Register (ID_COMP) . . . . .	16-14
16.5.1.3	Peripheral Revision Register (REV) . . . . .	16-15
16.5.1.4	Peripheral Additional Info Register (ADD_INFO) . . . . .	16-15
16.5.1.5	OTG Interrupt Status Register (OTG_INT_STAT) . . . . .	16-16
16.5.1.6	OTG Interrupt Control Register (OTG_INT_EN) . . . . .	16-17
16.5.1.7	Interrupt Status Register (OTG_STAT) . . . . .	16-18
16.5.1.8	OTG Control Register (OTG_CTRL) . . . . .	16-19
16.5.1.9	Interrupt Status Register (INT_STAT) . . . . .	16-20
16.5.1.10	Interrupt Enable Register (INT_ENB) . . . . .	16-21
16.5.1.11	Error Interrupt Status Register (ERR_STAT) . . . . .	16-22
16.5.1.12	Error Interrupt Enable Register (ERR_ENB) . . . . .	16-23
16.5.1.13	Status Register (STAT) . . . . .	16-24
16.5.1.14	Control Register (CTL) . . . . .	16-25
16.5.1.15	Address Register (ADDR) . . . . .	16-26
16.5.1.16	BDT Page Register 1 (BDT_PAGE_01) . . . . .	16-27
16.5.1.17	Frame Number Register Low/High (FRM_NUML, FRM_NUMH) . . . . .	16-28
16.5.1.18	Token Register (TOKEN) . . . . .	16-29
16.5.1.19	SOF Threshold Register (SOF_THLD) . . . . .	16-30
16.5.1.20	BDT Page Register 2 (BDT_PAGE_02) . . . . .	16-31
16.5.1.21	BDT Page Register 3 (BDT_PAGE_03) . . . . .	16-31
16.5.1.22	Endpoint Control Registers 0 – 15 (ENDPT0–15) . . . . .	16-32
16.5.1.23	USB Control Register (USB_CTRL) . . . . .	16-34
16.5.1.24	USB OTG Observe Register (USB_OTG_OBSERVE) . . . . .	16-34
16.5.1.25	USB OTG Control Register (USB_OTG_CONTROL) . . . . .	16-35
16.5.1.26	USB Transceiver and Regulator Control Register 0 (USBTRC0) . . . . .	16-36
16.5.1.27	OTG Pin Control Register (OTGPIN) . . . . .	16-37
16.6	OTG and Host Mode Operation . . . . .	16-38
16.6.1	Configuration of External Pull-up/Pull-down for USB . . . . .	16-38
16.7	Host Mode Operation Examples . . . . .	16-40
16.8	On-The-Go Operation . . . . .	16-42
16.8.1	OTG Dual Role A Device Operation . . . . .	16-42
16.8.2	OTG Dual Role B Device Operation . . . . .	16-44
16.8.3	Power . . . . .	16-45
16.8.4	USB Suspend State . . . . .	16-46

## Chapter 17 Real-Time Counter (S08RTCV1)

17.1	Introduction . . . . .	17-1
17.1.1	Features . . . . .	17-2

17.1.2 Modes of Operation . . . . .	17-2
17.1.2.1 Wait Mode . . . . .	17-2
17.1.2.2 Stop Modes . . . . .	17-2
17.1.2.3 Active Background Mode . . . . .	17-2
17.1.3 Block Diagram . . . . .	17-3
17.2 External Signal Description . . . . .	17-3
17.3 Register Definition . . . . .	17-3
17.3.1 RTC Status and Control Register (RTCSC) . . . . .	17-4
17.3.2 RTC Counter Register (RTCCNT) . . . . .	17-5
17.3.3 RTC Modulo Register (RTCMOD) . . . . .	17-5
17.4 Functional Description . . . . .	17-5
17.4.1 RTC Operation Example . . . . .	17-6
17.5 Initialization/Application Information . . . . .	17-7

## Chapter 18 Cryptographic Acceleration Unit (CAU)

18.1 CAU Registers . . . . .	18-1
18.1.1 CAU Status Register . . . . .	18-1
18.2 CAU Operation . . . . .	18-2
18.3 CAU Commands . . . . .	18-2
18.3.1 CNOP - coprocessor no operation . . . . .	18-3
18.3.2 LDR - load register . . . . .	18-3
18.3.3 STR - store register . . . . .	18-3
18.3.4 ADR - add to register . . . . .	18-4
18.3.5 RADR - reverse and add to register . . . . .	18-4
18.3.6 ADRA - add register to accumulator . . . . .	18-4
18.3.7 XOR - exclusive or . . . . .	18-4
18.3.8 ROTL - rotate left . . . . .	18-4
18.3.9 MVRA - move register to accumulator . . . . .	18-4
18.3.10MVAR - move accumulator to register . . . . .	18-4
18.3.11AESS - AES substitution . . . . .	18-5
18.3.12AESIS - AES inverse substitution . . . . .	18-5
18.3.13AESC - AES column operation . . . . .	18-5
18.3.14AESIC - AES inverse column operation . . . . .	18-5
18.3.15AESR - AES shift rows . . . . .	18-5
18.3.16AESIR - AES inverse shift rows . . . . .	18-5
18.3.17DESR - DES round . . . . .	18-6
18.3.18DESK - DES key setup . . . . .	18-6
18.3.19HASH - hash function . . . . .	18-6
18.3.20SHS - secure hash shift . . . . .	18-7
18.3.21MDS - message digest shift . . . . .	18-7
18.3.22ILL - illegal command . . . . .	18-7
18.4 CAU Equate Values . . . . .	18-7

## Chapter 19 Random Number Generator Accelerator (RNGA)

19.1 Overview . . . . .	19-1
-------------------------	------

19.2	Features	19-2
19.3	Modes of Operation	19-2
19.4	Memory Map/Register Definition	19-3
19.4.1	Register Descriptions	19-3
19.4.1.1	Control Register (RNGCR)	19-3
19.4.1.2	Status Register (RNGSR)	19-4
19.4.1.3	Entropy Register (RNGER)	19-6
19.4.1.4	Output Register (RNGOUT)	19-6
19.5	Functional Description	19-7
19.5.1	Output Register	19-7
19.5.2	Interface Block	19-7
19.5.3	RNGA Core/Control Logic Block	19-7
19.5.3.1	Control	19-8
19.5.3.2	Core Engine	19-8
19.6	Initialization/Application Information	19-8

## Chapter 20 Analog Comparator (S08ACMPV2)

20.1	Introduction	20-1
20.1.1	ACMP Configuration Information	20-1
20.1.2	ACMP/TPM Configuration Information	20-1
20.1.3	ACMP Clock Gating	20-1
20.1.4	Features	20-2
20.1.5	Modes of Operation	20-2
20.1.5.1	ACMP in Wait Mode	20-2
20.1.5.2	ACMP in Stop Modes	20-2
20.1.5.3	ACMP in Active Background Mode	20-2
20.1.6	Block Diagram	20-2
20.2	External Signal Description	20-3
20.3	Memory Map and Register Definition	20-4
20.3.1	Memory Map (Register Summary)	20-4
20.3.2	Register Descriptions	20-4
20.3.2.1	ACMP Status and Control Register (ACMPSC)	20-4
20.4	Functional Description	20-5

## Chapter 21 Analog-to-Digital Converter (S08ADC12V1)

21.1	Introduction	21-1
21.1.1	ADC Clock Gating	21-1
21.1.2	Module Configurations	21-1
21.1.2.1	Channel Assignments	21-1
21.1.2.2	Alternate Clock	21-2
21.1.2.3	Hardware Trigger	21-2
21.1.2.4	Temperature Sensor	21-2
21.1.3	Features	21-4
21.1.4	ADC Module Block Diagram	21-4
21.2	External Signal Description	21-5

21.2.1	Analog Power ( $V_{DDA}$ ) . . . . .	21-6
21.2.2	Analog Ground ( $V_{SSA}$ ) . . . . .	21-6
21.2.3	Voltage Reference High ( $V_{REFH}$ ) . . . . .	21-6
21.2.4	Voltage Reference Low ( $V_{REFL}$ ) . . . . .	21-6
21.2.5	Analog Channel Inputs (ADx) . . . . .	21-6
21.3	Register Definition . . . . .	21-6
21.3.1	Status and Control Register 1 (ADCSC1) . . . . .	21-6
21.3.2	Status and Control Register 2 (ADCSC2) . . . . .	21-8
21.3.3	Data Result High Register (ADCRH) . . . . .	21-9
21.3.4	Data Result Low Register (ADCRL) . . . . .	21-9
21.3.5	Compare Value High Register (ADCCVH) . . . . .	21-10
21.3.6	Compare Value Low Register (ADCCVL) . . . . .	21-10
21.3.7	Configuration Register (ADCCFG) . . . . .	21-10
21.3.8	Pin Control 1 Register (APCTL1) . . . . .	21-12
21.3.9	Pin Control 2 Register (APCTL2) . . . . .	21-13
21.3.10	Pin Control 3 Register (APCTL3) . . . . .	21-14
21.4	Functional Description . . . . .	21-15
21.4.1	Clock Select and Divide Control . . . . .	21-15
21.4.2	Input Select and Pin Control . . . . .	21-16
21.4.3	Hardware Trigger . . . . .	21-16
21.4.4	Conversion Control . . . . .	21-16
21.4.4.1	Initiating Conversions . . . . .	21-16
21.4.4.2	Completing Conversions . . . . .	21-17
21.4.4.3	Aborting Conversions . . . . .	21-17
21.4.4.4	Power Control . . . . .	21-17
	21.4.4.5 Sample Time and Total Conversion Time . . . . .	21-17
21.4.5	Automatic Compare Function . . . . .	21-19
21.4.6	MCU Wait Mode Operation . . . . .	21-20
21.4.7	MCU Stop3 Mode Operation . . . . .	21-20
21.4.7.1	Stop3 Mode With ADACK Disabled . . . . .	21-20
21.4.7.2	Stop3 Mode With ADACK Enabled . . . . .	21-21
21.4.8	MCU Stop2 Mode Operation . . . . .	21-21
21.5	Initialization Information . . . . .	21-21
21.5.1	ADC Module Initialization Example . . . . .	21-21
21.5.1.1	Initialization Sequence . . . . .	21-21
21.5.1.2	Pseudo-Code Example . . . . .	21-22
21.6	Application Information . . . . .	21-23
21.6.1	External Pins and Routing . . . . .	21-23
21.6.1.1	Analog Supply Pins . . . . .	21-23
21.6.1.2	Analog Reference Pins . . . . .	21-24
21.6.1.3	Analog Input Pins . . . . .	21-24
21.6.2	Sources of Error . . . . .	21-25
21.6.2.1	Sampling Error . . . . .	21-25
21.6.2.2	Pin Leakage Error . . . . .	21-25
21.6.2.3	Noise-Induced Errors . . . . .	21-25
21.6.2.4	Code Width and Quantization Error . . . . .	21-26
21.6.2.5	Linearity Errors . . . . .	21-26
21.6.2.6	Code Jitter, Non-Monotonicity, and Missing Codes . . . . .	21-27

## Chapter 22

### Timer/PWM Module

22.1	Introduction	22-1
22.1.1	Features	22-1
22.1.2	Modes of Operation	22-1
22.1.3	Block Diagram	22-2
22.2	Signal Description	22-4
22.2.1	Detailed Signal Descriptions	22-4
22.2.1.1	EXTCLK — External Clock Source	22-5
22.2.1.2	TPMxCHn — TPM Channel n I/O Pin(s)	22-5
22.3	Register Definition	22-8
22.3.1	TPM Status and Control Register (TPMxCSC)	22-8
22.3.2	TPM-Counter Registers (TPMxCNTH:TPMxCNTL)	22-9
22.3.3	TPM Counter Modulo Registers (TPMxMODH:TPMxMODL)	22-10
22.3.4	TPM Channel n Status and Control Register (TPMxCnSC)	22-11
22.3.5	TPM Channel Value Registers (TPMxCnVH:TPMxCnVL)	22-12
22.4	Functional Description	22-14
22.4.1	Counter	22-14
22.4.1.1	Counter Clock Source	22-14
22.4.1.2	Counter Overflow and Modulo Reset	22-15
22.4.1.3	Counting Modes	22-16
22.4.1.4	Manual Counter Reset	22-16
22.4.2	Channel Mode Selection	22-16
22.4.2.1	Input Capture Mode	22-16
22.4.2.2	Output Compare Mode	22-16
22.4.2.3	Edge-Aligned PWM Mode	22-17
22.4.2.4	Center-Aligned PWM Mode	22-18
22.5	Reset Overview	22-19
22.5.1	General	22-19
22.5.2	Description of Reset Operation	22-19
22.6	Interrupts	22-19
22.6.1	General	22-19
22.6.2	Description of Interrupt Operation	22-20
22.6.2.1	Timer Overflow Interrupt (TOF) Description	22-20
22.6.2.2	Channel Event Interrupt Description	22-21

## Chapter 23

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

23.1	Introduction	23-1
23.1.1	Overview	23-2
23.1.2	Features	23-3
23.1.3	Modes of Operations	23-3
23.2	External Signal Descriptions	23-5
23.3	Memory Map/Register Definition	23-6
23.3.1	Configuration/Status Register (CSR)	23-7
23.3.2	Extended Configuration/Status Register (XCSR)	23-10
23.3.3	Configuration/Status Register 2 (CSR2)	23-13

23.3.4 Configuration/Status Register 3 (CSR3) . . . . .	23-16
23.3.5 BDM Address Attribute Register (BAAR) . . . . .	23-17
23.3.6 Address Attribute Trigger Register (AATR) . . . . .	23-18
23.3.7 Trigger Definition Register (TDR) . . . . .	23-19
23.3.8 Program Counter Breakpoint/Mask Registers (PBR0–3, PBMR) . . . . .	23-22
23.3.9 Address Breakpoint Registers (ABLR, ABHR) . . . . .	23-24
23.3.10 Data Breakpoint and Mask Registers (DBR, DBMR) . . . . .	23-25
23.3.11 Resulting Set of Possible Trigger Combinations . . . . .	23-26
<b>23.4 Functional Description . . . . .</b>	<b>23-26</b>
<b>23.4.1 Background Debug Mode (BDM) . . . . .</b>	<b>23-26</b>
23.4.1.1 CPU Halt . . . . .	23-27
23.4.1.2 Background Debug Serial Interface Controller (BDC) . . . . .	23-29
23.4.1.3 BDM Communication Details . . . . .	23-30
23.4.1.4 BDM Command Set Descriptions . . . . .	23-33
23.4.1.5 BDM Command Set Summary . . . . .	23-36
23.4.1.6 Serial Interface Hardware Handshake Protocol . . . . .	23-51
23.4.1.7 Hardware Handshake Abort Procedure . . . . .	23-53
23.4.2 Real-Time Debug Support . . . . .	23-56
23.4.3 Trace Support . . . . .	23-56
23.4.3.1 Begin Execution of Taken Branch (PST = 0x05) . . . . .	23-59
23.4.3.2 PST Trace Buffer (PSTB) . . . . .	23-60
23.4.3.3 PST/DDATA Example . . . . .	23-60
23.4.3.4 Processor Status, Debug Data Definition . . . . .	23-62
23.4.4 Freescale-Recommended BDM Pinout . . . . .	23-66

# Chapter 1

## Device Overview

### 1.1 The MCF51JM128 Series Microcontrollers

#### 1.1.1 Definition

The MCF51JM128 series microcontrollers are systems-on-chip (SoCs) that are based on the V1 ColdFire core and:

- Operate at processor core speeds up to 50.33 MHz (peripherals operate at half of this speed)
- Integrate technologies that are important for today's consumer and industrial applications, such as USB On-The-Go, Controller Area Network, and cryptographic acceleration
- Are the ideal upgrade for designs based on the MC9S08JM60 series microcontrollers

#### 1.1.2 MCF51JM128 Series Devices

There are two members of the MCF51JM128 series, each available in various packages, as shown in [Table 1-1](#).

**Table 1-1. MCF51JM128 Series Package Availability**

	MCF51JM128	MCF51JM64	MCF51JM32
80-pin LQFP	Yes	Yes	Yes
64-pin LQFP	Yes	Yes	Yes
64-pin QFP	Yes	Yes	Yes
44-pin LQFP	Yes	Yes	Yes

### 1.1.3 MCF51JM128 Series Device Comparison

Table 1-2 compares the MCF51JM128 series microcontrollers.

**Table 1-2. MCF51JM128 Series Device Comparison**

Feature	MCF51JM128			MCF51JM64			MCF51JM32		
	80-pin	64-pin	44-pin	80-pin	64-pin	44-pin	80-pin	64-pin	44-pin
Flash memory size (Kbytes)	128			64			32		
RAM size (Kbytes)	16			16			16		
V1 ColdFire core with BDM (background debug module)				Yes					
ACMP (analog comparator)				Yes					
ADC channels (12-bit)	12		8	12		8	12		8
CAN (controller area network)	Yes	Yes	No	Yes	Yes	No	Yes	Yes	No
RNGA + CAU				Yes <sup>1</sup>					
CMT (carrier modulator timer)				Yes					
COP (computer operating properly)				Yes					
IIC1 (inter-integrated circuit)				Yes					
IIC2	Yes	No		Yes	No		Yes	No	
IRQ (interrupt request input)				Yes					
KBI (keyboard interrupts)	8	8	6	8	8	6	8	8	6
LVD (low-voltage detector)				Yes					
MCG (multipurpose clock generator)				Yes					
Port I/O <sup>2</sup>	66	51	33	66	51	33	66	51	33
GPIO (rapid general-purpose I/O)	16	6	0	16	6	0	16	6	0
RTC (real-time counter)				Yes					
SCI1 (serial communications interface)				Yes					
SCI2				Yes					
SPI1 (serial peripheral interface)				Yes					
SPI2				Yes					
TPM1 (timer/pulse-width modulator) channels	6	6	4	6	6	4	6	6	4
TPM2 channels				2					
USBOTG (USB On-The-Go dual-role controller)				Yes					
XOSC (crystal oscillator)				Yes					

<sup>1</sup> Only existed on special parts

<sup>2</sup> Up to 16 pins on Ports A, H, and J are shared with the ColdFire Rapid GPIO module.

## 1.2 MCF51JM128 Series Block Diagram

### 1.2.1 Block Diagram

Figure 1-1 shows the connections between the MCF51JM128 series pins and functional units.

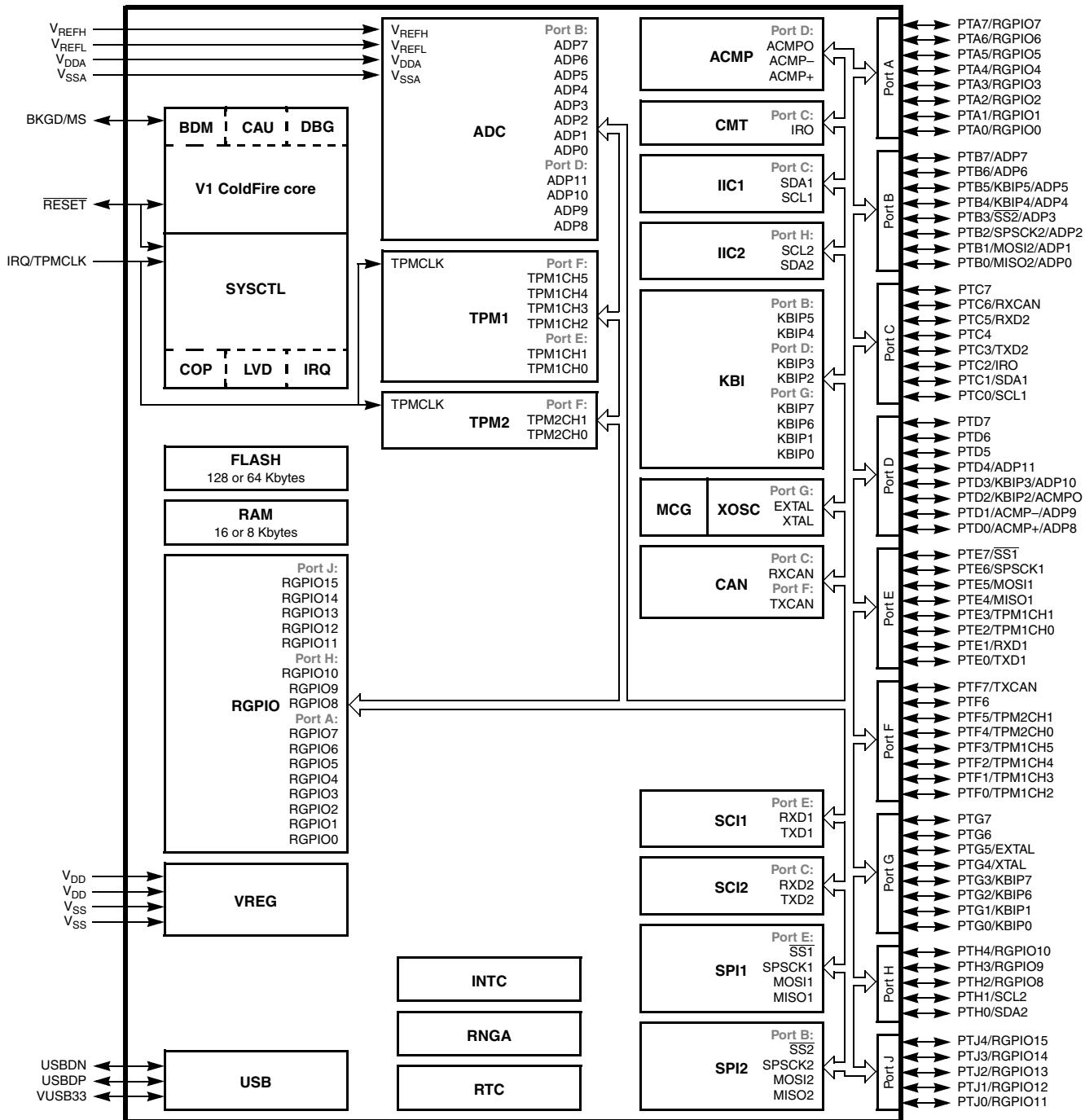


Figure 1-1. MCF51JM128 Series Block Diagram

## 1.2.2 Functional Units

Table 1-3 describes the functional units of the MCF51JM128 series microcontrollers.

**Table 1-3. MCF51JM128 Series Functional Units**

Unit	Function
CF1CORE (V1 ColdFire core)	Executes programs and interrupt handlers.
BDM (background debug module)	Provides single pin debugging interface (part of the V1 ColdFire core)
DBG (debug)	Provides debugging and emulation capabilities (part of the V1 ColdFire core)
SYSCTL (system control)	Provides LVD, COP, external interrupt request, and so on
FLASH (flash memory)	Provides storage for program code, constants and variables
RAM (random-access memory)	Provides storage for data
RGPIO (rapid general-purpose input/output)	Allows for I/O port access at CPU clock speeds
VREG (voltage regulator)	Controls power management across the device
USB (USB On-The-Go)	USB On-The-Go dual-role controller
ADC (analog-to-digital converter)	Measures analog voltages at up to 12 bits of resolution
TPM1, TPM2 (timer/pulse-width modulators)	Provide a variety of timing-based features
CF1_INTC (interrupt controller)	Controls and prioritizes all device interrupts
CAU (cryptographic acceleration unit)	Co-processor support of DES, 3DES, AES, MD5, and SHA-1
RNGA (random number generator accelerator)	32-bit random number generator that complies with FIPS-140
RTC (real-time counter)	Provides a constant time-base with optional interrupt
ACMP (analog comparator)	Compare two analog inputs
CMT (carrier modulator timer)	Infrared output used for Remote Controller
IIC1, IIC2 (inter-integrated circuits)	Supports standard IIC communications protocol
KBI (keyboard interrupt)	Provides pin interrupt capabilities
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
XOSC (crystal oscillator)	Supports low- or high-range crystals
CAN (controller area network)	Supports standard CAN communications protocol
SCI1, SCI2 (serial communications interfaces)	Serial communications UARTs capable of supporting RS-232 and LIN protocols
SPI1, SPI2 (serial peripheral interfaces)	Provide 4-pin synchronous serial interface

## 1.3 V1 ColdFire Core

The MCF51JM128 series devices contain a version of the V1 ColdFire core that is optimized for area and low power. This CPU implements ColdFire instruction set architecture revision C (ISA\_C) with a reduced programming model:

- No hardware support for MAC/EMAC and DIV instructions<sup>1</sup>
- Upward compatibility with all other ColdFire cores (V2–V5)

For more details on the V1 ColdFire core, see [Chapter 6, “ColdFire Core.”](#)

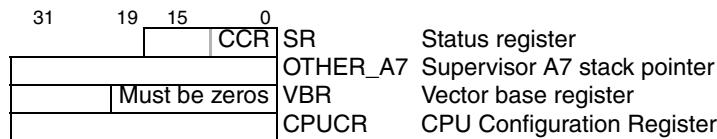
### 1.3.1 User Programming Model

[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 as well as the registers listed in [Figure 1-2](#).



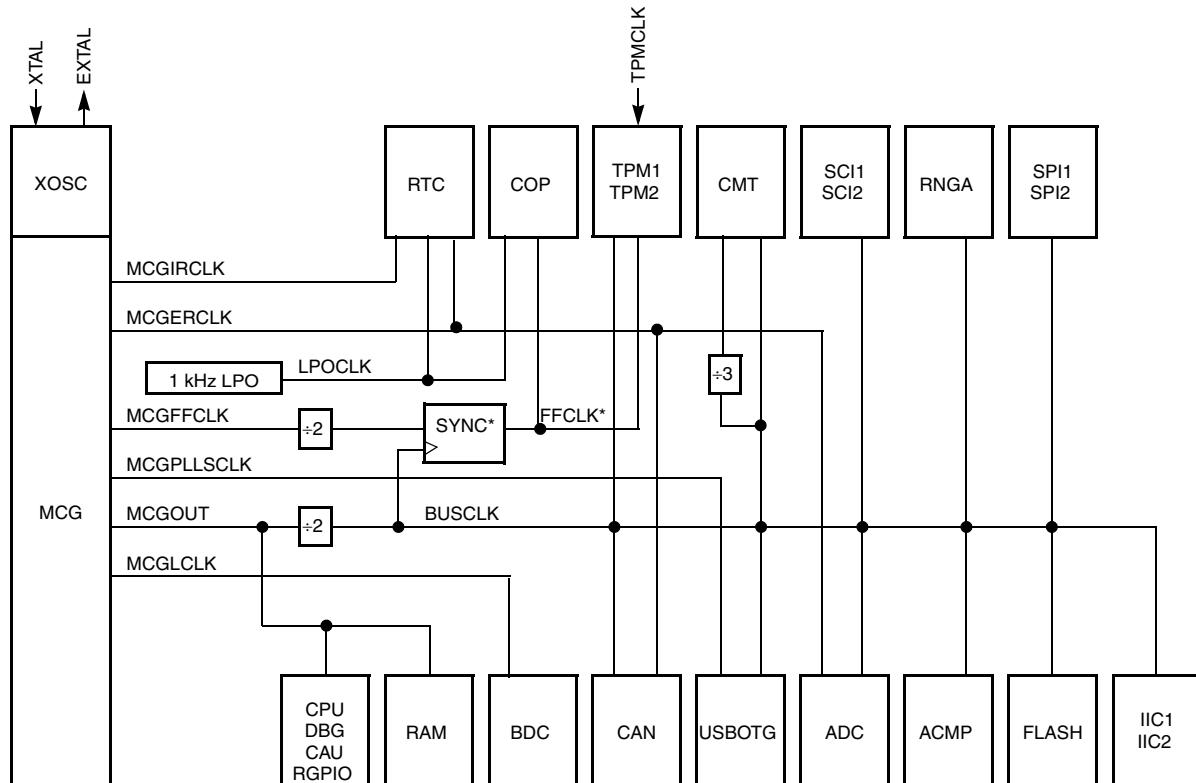
**Figure 1-2. Supervisor Programming Model**

1. These operations can be emulated via software functions.

## 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).



Note: The ADC has minimum and maximum frequency requirements. See the ADC chapter and the *MCF51JM128 Data Sheet*. Flash memory has frequency requirements for program and erase operations. See the *MCF51JM128 Data Sheet*.

\* 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**

## 1.4.2 System Clocks

[Table 1-4](#) describes each of the system clocks.

**Table 1-4. System Clocks**

Clock	Description
MCGOUT	<p>This clock source is used as the CPU clock and is divided by two to generate the peripheral bus clock. Control bits in the MCG control registers determine which of three clock sources is connected:</p> <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> <p>This clock drives the CPU, debug, RAM, GPIO and BDM directly and is divided by two to clock all peripherals (BUSCLK). See <a href="#">Chapter 7, “Multipurpose Clock Generator (S08MCGV3)</a>, for details on configuring the MCGOUT clock.</p>
MCGLCLK	This clock source is derived from the digitally controlled oscillator (DCO) of the MCG. Development tools can select this internal self-clocked source to speed up BDC communications in systems where the bus clock is slow.
MGERCLK	MCG External Reference Clock—This is the external reference clock and can be selected as the real-time counter clock source and/or the alternate clock for the ADC, RTC, and msCAN module.
MGIRCLK	MCG Internal Reference Clock—This is the internal reference clock and can be selected as the real-time counter clock source.
MGFFCLK	MCG Fixed-Frequency Clock—This clock is divided by 2 to generate the fixed frequency clock (FFCLK) after being synchronized to the bus clock. It can be selected as clock source for the TPM modules. The frequency of the FFCLK is determined by the settings of the MCG.
LPOCLK	Low-Power Oscillator Clock—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.
TPMCLK	TPM Clock—An optional external clock source for the TPMs. This clock must be limited to one-quarter the frequency of the bus clock for synchronization.
MGPLLSCLK	This clock has a direct connection to the PLL output clock (running at 48 MHz) and thus allows the user to have the flexibility to run the MCGOUT at lower frequencies to conserve power.
ADACK (not shown)	The ADC module also has an internally generated asynchronous clock that allows it to run in STOP mode (ADACK). This signal is not available externally.

## 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 5.7.8, “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 1-5](#).

**Table 1-5. MCG Modes**

Mode	Related field values <sup>1</sup>	Description
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. The MCG supplies a clock derived from one of the on-chip FLLs, which is sourced by the internal reference clock. Upon exiting reset, the default FLL is that which generates the x MHz bus / y MHz CPU clocks.
FLL Engaged External (FEE)	<ul style="list-style-type: none"> <li>MCGC1[IREFS] = 0</li> <li>MCGC1[CLKS] = 00</li> <li>MCGC3[PLLS] = 0</li> </ul>	The MCG supplies a clock derived from the FLL, which is sourced from an external reference clock (or crystal oscillator).
FLL Bypassed Internal (FBI)	<ul style="list-style-type: none"> <li>MCGC1[IREFS] = 1</li> <li>MCGC1[CLKS] = 01</li> <li>MCGC3[PLLS] = 0</li> <li>XCSM[ENBDM] = 1 or MCGC2[LP] = 0</li> </ul>	The FLL is enabled and sourced by the internal reference clock but is bypassed. The MCGOUT clock is derived from the internal reference clock.
FLL Bypassed External (FBE)	<ul style="list-style-type: none"> <li>MCGC1[IREFS] = 0</li> <li>MCGC1[CLKS] = 10</li> <li>MCGC3[PLLS] = 0</li> <li>XCSM[ENBDM] = 1 or MCGC2[LP] = 0</li> </ul>	The FLL is enabled and controlled by an external reference clock but is bypassed. The MCGOUT clock is derived from the external reference clock. The external reference clock can be an external crystal/resonator supplied by an OSC controlled by the MCG, or it can be another external clock source.
PLL Engaged External (PEE)	<ul style="list-style-type: none"> <li>MCGC1[IREFS] = 0</li> <li>MCGC1[CLKS] = 00</li> <li>MCGC3[PLLS] = 1</li> </ul>	The MCG supplies a clock derived from the PLL, which is sourced from an external reference clock (or crystal oscillator).
PLL Bypassed External (PBE)	<ul style="list-style-type: none"> <li>MCGC1[IREFS] = 0</li> <li>MCGC1[CLKS] = 10</li> <li>MCGC3[PLLS] = 1</li> <li>XCSM[ENBDM] = 1 or MCGC2[LP] = 0</li> </ul>	The MCG supplies a clock MCGOUT derived from the external reference clock (or crystal oscillator). The PLL is also sourced from the external clock source but is bypassed.
Bypassed Low Power Internal (BLPI)	<ul style="list-style-type: none"> <li>MCGC1[IREFS] = 1</li> <li>MCGC1[CLKS] = 01</li> <li>XCSM[ENBDM] = 0 and MCGC2[LP] = 1</li> </ul>	The FLL and PLL are disabled and bypassed, and the MCG supplies MCGOUT derived from the internal reference clock.
Bypassed Low Power External (BLPE)	<ul style="list-style-type: none"> <li>MCGC1[IREFS] = 0</li> <li>MCGC1[CLKS] = 10</li> <li>XCSM[ENBDM] = 0 and MCGC2[LP] = 1</li> </ul>	The FLL and PLL are disabled and bypassed, and the MCG supplies MCGOUT derived from the external reference clock.
STOP	—	The FLL and PLL are disabled, and the internal or external reference clocks can be selected to be enabled or disabled. The microcontroller does not provide a microcontroller clock source unless BDC[ENBDM] is enabled.

<sup>1</sup> For descriptions of the MCGC1, MCGC2, and MCGC3 registers, see Chapter 7, “Multipurpose Clock Generator (S08MCGV3)”. For a description of the XCSM register, see Chapter 23, “Version 1 ColdFire Debug (CF1\_DEBUG)”.

## 1.4.5 MCG Mode State Diagram

Figure 1-4 shows the valid state transitions for the MCG.

The IREFS and CLKS fields are contained within the MCG module definition. The LP bit is part of the on-chip power management controller (PMC) block.

The clock source for the BDC is controlled by the BDC clksw bit. Choices for the BDC clock are MCGOUT and the output from DCO.

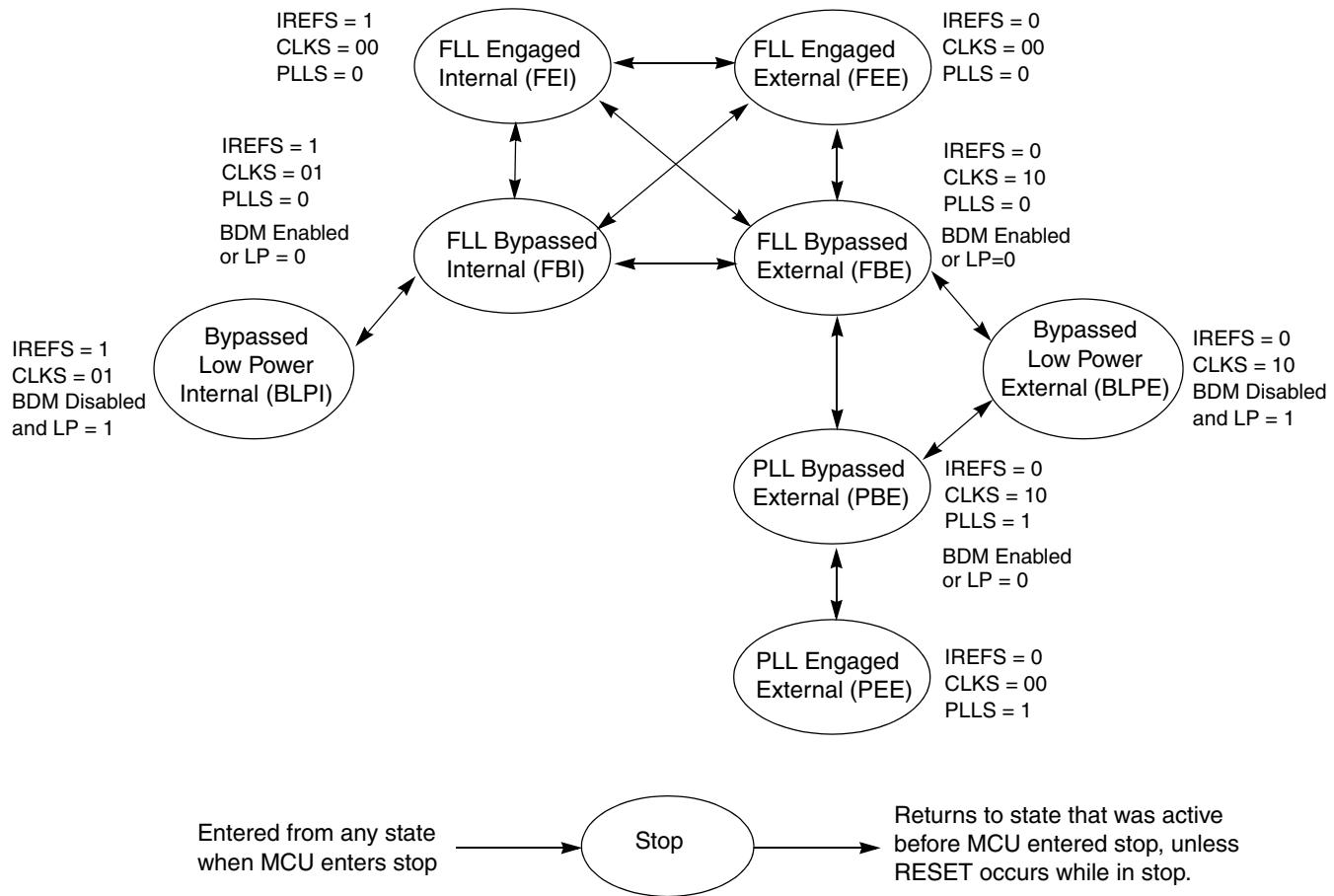


Figure 1-4. MCG Mode State Diagram



# 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 Device Pin Assignment

#### 2.1.1 Pinout: 80-Pin LQFP

Figure 2-1 shows the pinout of the 80-pin LQFP.

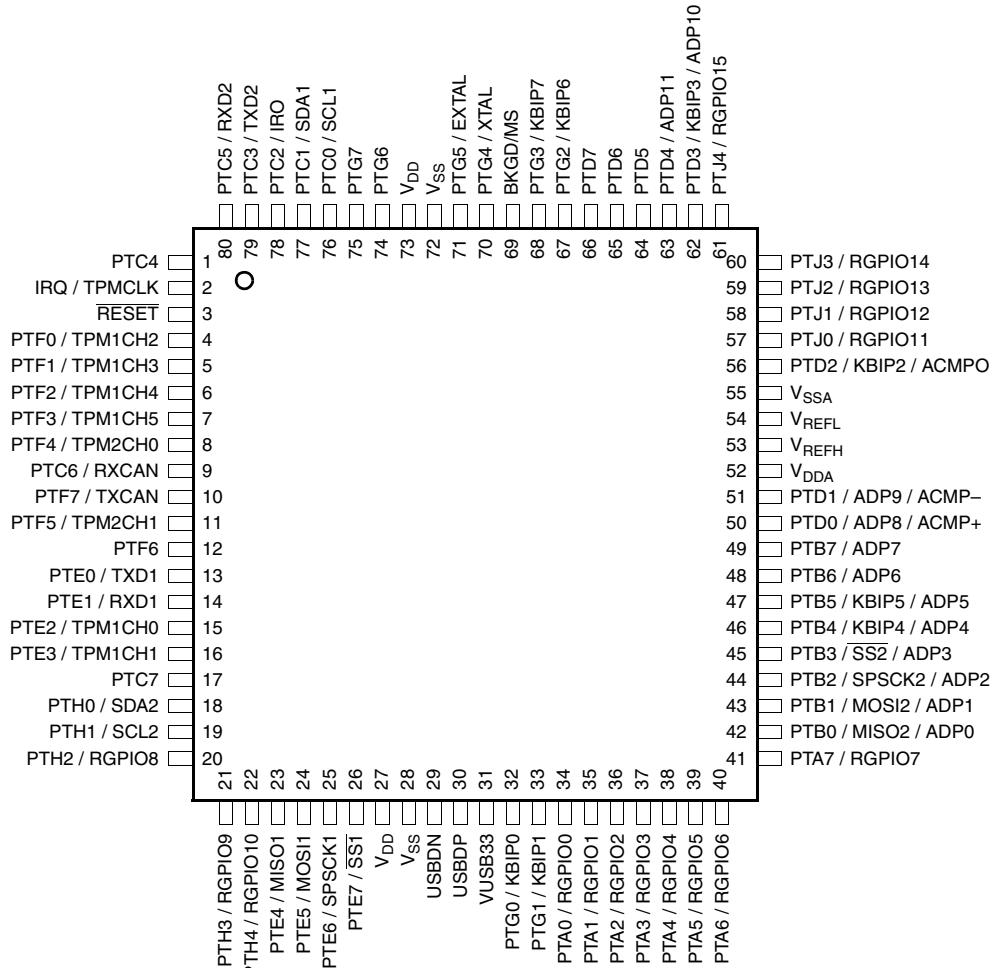


Figure 2-1. 80-Pin LQFP

## 2.1.2 Pinout: 64-Pin LQFP and QFP

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

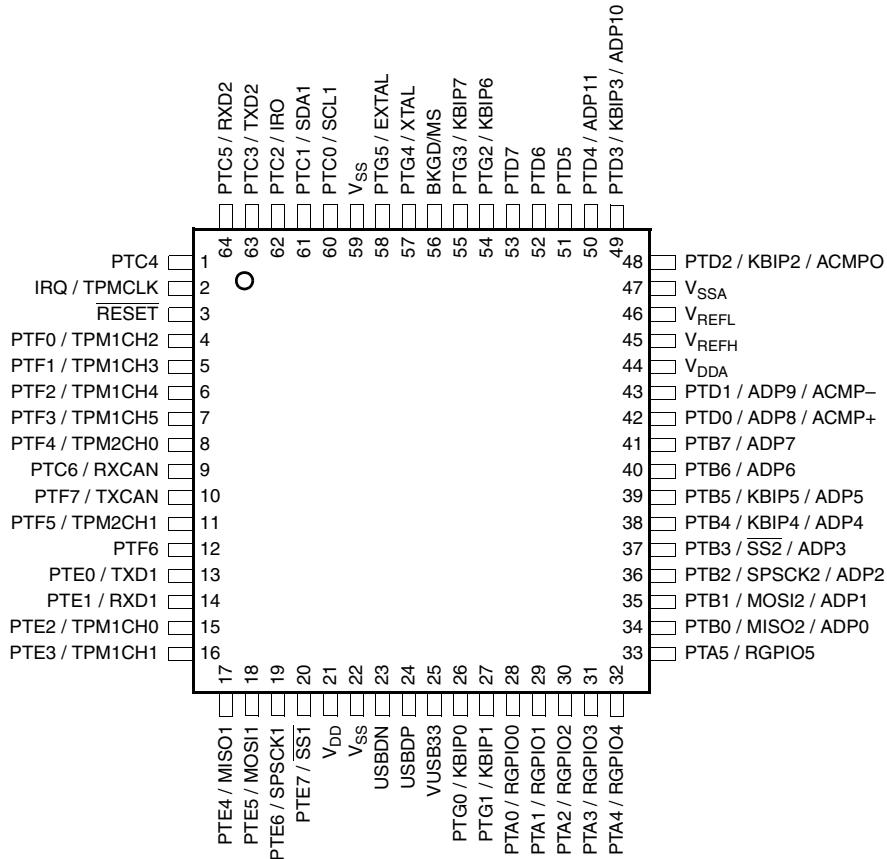


Figure 2-2. 64-Pin LQFP and QFP

### 2.1.3 Pinout: 44-Pin LQFP

Figure 2-3 shows the pinout of the 44-pin LQFP.

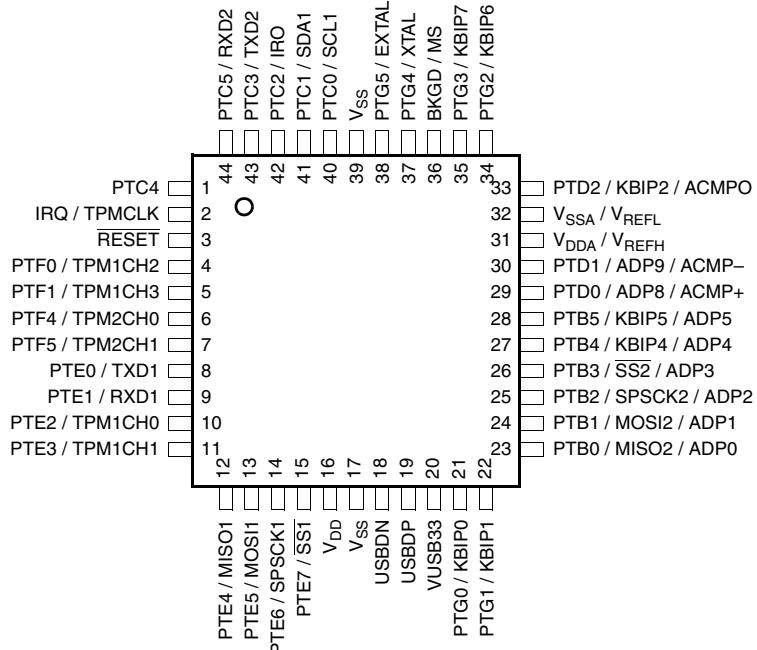


Figure 2-3. 44-Pin LQFP

## 2.1.4 Pin Assignments

**Table 2-1. Pin Assignments by Package and Pin Sharing Priority**

Pin Number			<-- Lowest Priority --> Highest		
80	64	44	Port Pin	Alt 1	Alt 2
1	1	1	PTC4	—	—
2	2	2	—	IRQ	TPMCLK
3	3	3	—	RESET	—
4	4	4	PTF0	TPM1CH2	—
5	5	5	PTF1	TPM1CH3	—
6	6	—	PTF2	TPM1CH4	—
7	7	—	PTF3	TPM1CH5	—
8	8	6	PTF4	TPM2CH0	BUSCLK_OUT
9	9	—	PTC6	RXCAN	—
10	10	—	PTF7	TXCAN	—
11	11	7	PTF5	TPM2CH1	—
12	12	—	PTF6	—	—
13	13	8	PTE0	TXD1	—
14	14	9	PTE1	RXD1	—
15	15	10	PTE2	TPM1CH0	—
16	16	11	PTE3	TPM1CH1	—
17	—	—	PTC7	—	—
18	—	—	PTH0	SDA2	—
19	—	—	PTH1	SCL2	—
20	—	—	PTH2	GPIO8	—
21	—	—	PTH3	GPIO9	—
22	—	—	PTH4	GPIO10	—
23	17	12	PTE4	MISO1	—
24	18	13	PTE5	MOSI1	—
25	19	14	PTE6	SPSCK1	—
26	20	15	PTE7	SS1	—
27	21	16	—	—	V <sub>DD</sub>
28	22	17	—	—	V <sub>SS</sub>
29	23	18	—	—	USBDN
30	24	19	—	—	USBDP

**Table 2-1. Pin Assignments by Package and Pin Sharing Priority (continued)**

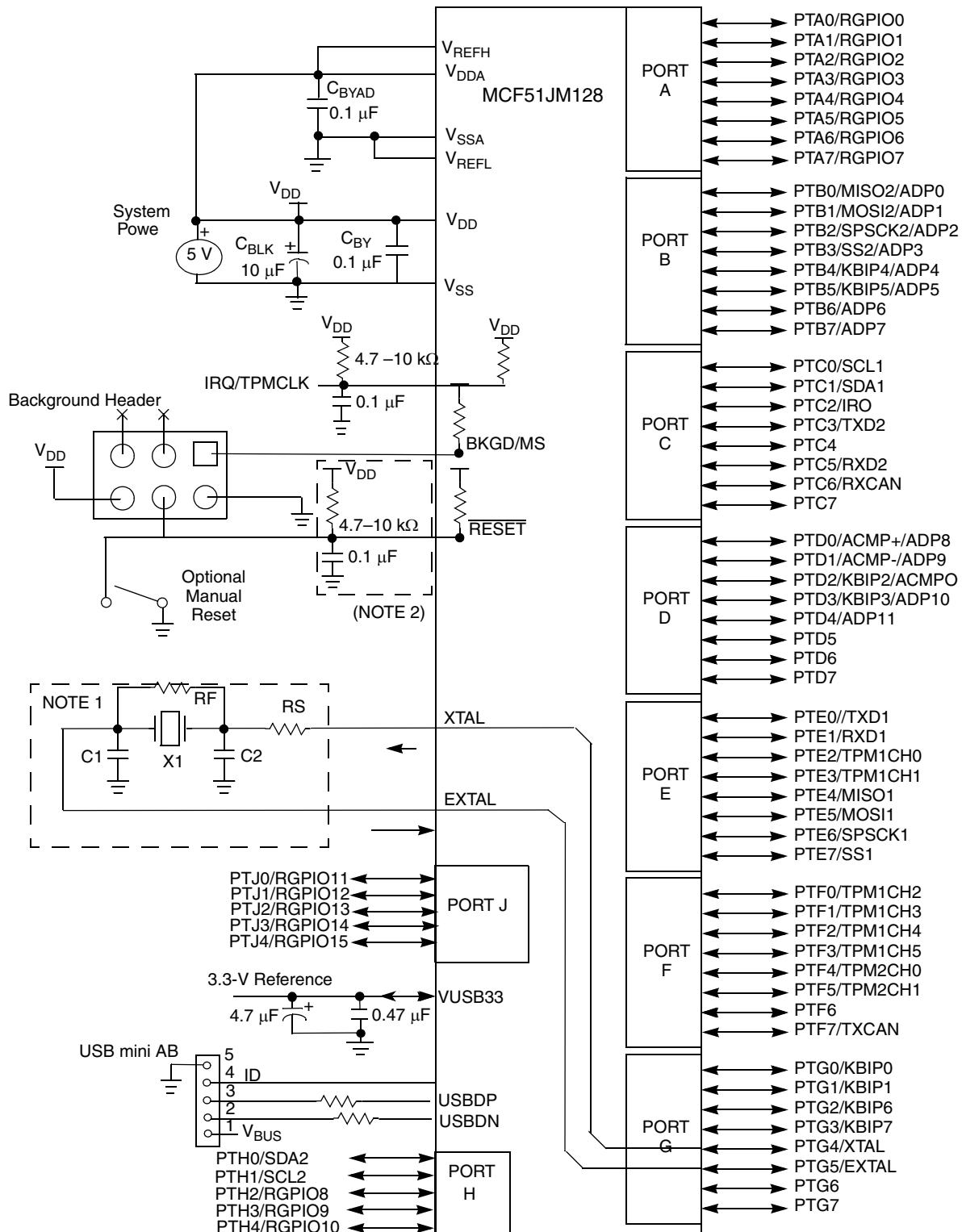
Pin Number			<-- Lowest Priority --> Highest		
80	64	44	Port Pin	Alt 1	Alt 2
31	25	20	—	—	VUSB33
32	26	21	PTG0	KBIP0	USB_ALT_CLK
33	27	22	PTG1	KBIP1	—
34	28	—	PTA0	GPIO0	USB_SESSVLD
35	29	—	PTA1	GPIO1	USB_SESEND
36	30	—	PTA2	GPIO2	USB_VBUSVLD
37	31	—	PTA3	GPIO3	USB_PULLUP(D+)
38	32	—	PTA4	GPIO4	USB_DM_DOWN
39	33	—	PTA5	GPIO5	USB_DP_DOWN
40	—	—	PTA6	GPIO6	USB_ID
41	—	—	PTA7	GPIO7	—
42	34	23	PTB0	MISO2	ADP0
43	35	24	PTB1	MOSI2	ADP1
44	36	25	PTB2	SPSCK2	ADP2
45	37	26	PTB3	SS2	ADP3
46	38	27	PTB4	KBIP4	ADP4
47	39	28	PTB5	KBIP5	ADP5
48	40	—	PTB6	ADP6	—
49	41	—	PTB7	ADP7	—
50	42	29	PTD0	ADP8	ACMP+
51	43	30	PTD1	ADP9	ACMP-
52	44	31	—	—	V <sub>DDA</sub>
53	45		—	—	V <sub>REFH</sub>
54	46	32	—	—	V <sub>REFL</sub>
55	47		—	—	V <sub>SSA</sub>
56	48	33	PTD2	KBIP2	ACMPO
57	—	—	PTJ0	GPIO11	—
58	—	—	PTJ1	GPIO12	—
59	—	—	PTJ2	GPIO13	—
60	—	—	PTJ3	GPIO14	—
61	—	—	PTJ4	GPIO15	—

**Table 2-1. Pin Assignments by Package and Pin Sharing Priority (continued)**

Pin Number			<-- Lowest Priority --> Highest		
80	64	44	Port Pin	Alt 1	Alt 2
62	49	—	PTD3	KBIP3	ADP10
63	50	—	PTD4	ADP11	—
64	51	—	PTD5	—	—
65	52	—	PTD6	—	—
66	53	—	PTD7	—	—
67	54	34	PTG2	KBIP6	—
68	55	35	PTG3	KBIP7	—
69	56	36	—	BKGD	MS
70	57	37	PTG4	XTAL	—
71	58	38	PTG5	EXTAL	—
72	59	39	—	—	V <sub>SS</sub>
73	—	—	—	—	V <sub>DD</sub>
74	—	—	PTG6	—	—
75	—	—	PTG7	—	—
76	60	40	PTC0	SCL1	—
77	61	41	PTC1	SDA1	—
78	62	42	PTC2	IRO	—
79	63	43	PTC3	TXD2	—
80	64	44	PTC5	RXD2	—

## 2.2 Recommended System Connections

Figure 2-4 shows pin connections that are common to MCF51JM128 series application systems.



**Figure 2-4. Basic System Connections**

## 2.2.1 Power

$V_{DD}$  and  $V_{SS}$  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 regulated lower-voltage source to the CPU and other internal circuitry of the microcontroller.

Typically, application systems have two separate capacitors 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 and a 0.1  $\mu F$  ceramic bypass capacitor located as close to the microcontroller power pins as practical to suppress high-frequency noise. The MCF51JM128 has two  $V_{DD}$  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 and ACMP modules. 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.

$V_{USB33}$  maintains an output voltage of 3.3 V and sources enough current for the internal USB transceiver and USB pullup resistor. If using an external 3.3 V regulator as an input to  $V_{USB33}$  (just when  $USBVREN = 0$ ), the supply voltage  $VDD$  must not fall below the input voltage at the  $V_{USB33}$  pin. If using the internal 3.3 V regulator ( $USBVREN = 1$ ):

- do not connect an external supply to the  $V_{USB33}$  pin
- ensure that the supply voltage  $VDD$  falls between 3.9 V and 5.5 V so that the internal 3.3 V regulator operates correctly

Two separate capacitors (4.7  $\mu F$  bulk electrolytic stability capacitor and 0.47  $\mu F$  ceramic bypass capacitors) must be connected across this pin to ground to decrease the output ripple of this voltage regulator when it is enabled.

## 2.2.2 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.  $C1$  and  $C2$  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.

$C1$  and  $C2$  are typically in the 5 pF to 25 pF range and are chosen to match the requirements of a specific crystal or resonator. Be sure to take into account printed circuit board (PCB) capacitance and microcontroller pin capacitance when selecting  $C1$  and  $C2$ . The crystal manufacturer typically specifies a load capacitance that is the series combination of  $C1$  and  $C2$  (which are usually the same size). As a

first-order approximation, use 10 pF as an estimate of combined pin and PCB capacitance for each oscillator pin (EXTAL and XTAL).

### 2.2.3 **RESET**

**RESET** is a dedicated pin with a pullup device built in. It has input hysteresis, a high current output driver, and no output slew rate control. Internal power-on reset and low-voltage reset circuitry typically make external reset circuitry unnecessary. This pin 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).

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).

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

### 2.2.4 **IRQ/TPMCLK**

The IRQ pin is the input source for the IRQ interrupt. If the IRQ function is not enabled, this pin can be used for TPMCLK. In EMC-sensitive applications, an external RC filter is recommended on the IRQ pin. See [Figure 2-4](#) for an example.

### 2.2.5 **Background / Mode Select (BKGD/MS)**

During a power-on-reset (POR) or background debug force reset (see bit ENBDM in [Section 23.3.2, “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.

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 to 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 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.

---

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

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

The  $V_{REFH}$  and  $V_{REFL}$  pins are the voltage reference high and voltage reference low inputs, respectively, for the ADC module.

## 2.2.7 USB Data Pins (USBDP, USBDN)

The USBDP (D+) and USBDN (D-) pins are the analog input/output lines to/from full-speed internal USB transciever. An optional internal pullup resistor for the USBDP pin,  $R_{PUDP}$ , is available. See [Chapter 16, “Universal Serial Bus, OTG Capable Controller”](#) for more details.

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

The MCF51JM128 series microcontrollers support up to 66 general-purpose I/O pins, which are shared with on-chip peripheral functions (timers, serial I/O, ADC, ACMP, 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. Immediately after reset, all of these pins are configured as high-impedance general-purpose inputs with internal pull-up devices enabled.

When an on-chip peripheral system is controlling a pin, data direction control bits 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 or non-bonded pins to outputs so they do not float.

# Chapter 3

## Modes of Operation

### 3.1 Introduction

The operating modes of the MCF51JM128 series MCUs 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 interrelated, variables: debug mode, security mode, power mode and clock mode. Clock modes were discussed in [Section 1.4.4, “MCG Modes of Operation.”](#) This chapter covers the other dimensions of the system operating mode.

### 3.2 Features

- Debug mode for code development. For devices based on the V1 ColdFire core, such as those in the MCF51JM128 series, debug mode and secure mode are mutually exclusive.
- 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.
- Wait mode—The CPU shuts down to conserve power; peripheral clocks are running and full regulation is maintained.
- Stop modes—System (CPU and peripheral) clocks are stopped.
  - Stop4—All internal circuits are powered (full regulation mode) and internal clock sources at max frequency for fastest recovery, LVD enabled, or XCSR[ENBDM] = 1.
  - Stop3—All internal circuits are loosely regulated and clocks sources are at minimal values. LVD disabled, and XCSR[ENBDM] = 0. Providing a good compromise between power and recovery time.
  - Stop2—Partial power-down of internal circuits; RAM content is retained. The lowest power mode for this device. After wakeup from Stop2 mode, the MCU goes through a reset sequence, even if the source of wakeup was an interrupt. LVD is not active, XCSR[ENBDM] = 0, and PPDC = 1.

On the MCF51JM128 series MCUs, Wait, Stop2, Stop3 and Stop4 are all entered via the CPU STOP instruction. See [Table 3-1](#) and subsequent sections of this chapter for details.

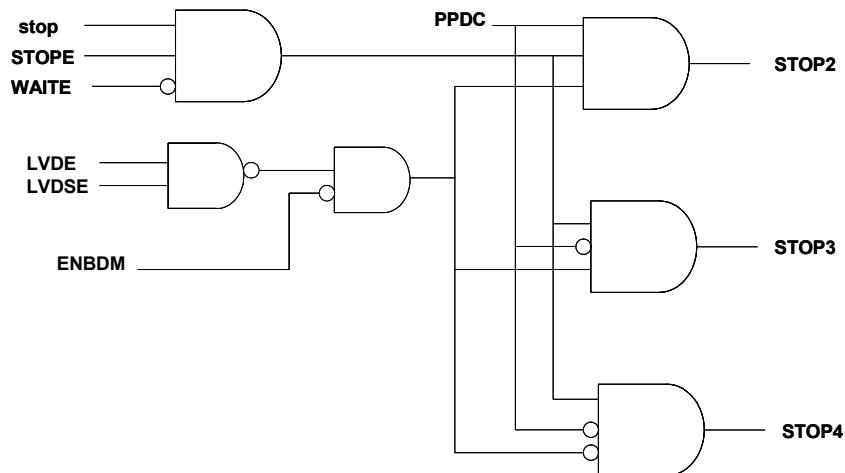
### 3.3 Overview

The ColdFire CPU has two primary modes of operation: Run and Stop. The CPU also supports a halt mode used strictly for debugging operations. The STOP instruction is used to invoke Stop and Wait modes for this family of devices.

If the SOPT1[WAITE] control bit is set when STOP is executed, the wait mode is entered. Otherwise, if the SOPT1[STOPE] bit is set, one of the Stop modes is entered. It is illegal to execute a STOP instruction if neither STOPE or WAITE are set. This results in reset assertion if the instruction-related reset disable bit in the CPU Control Register (CPUCR[IRD]) is cleared or an illegal instruction exception if CPUCR[IRD] is set.

During The MCF51AG 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.

During partial power-down mode, the regulator is in standby mode and most of the digital logic on the chip 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. MCF51JM128 Stop Modes**

Table 3-1. CPU / Power Mode Selections

Mode of Operation	Register and Bit names						CPU and Peripheral Clocks	Affects on Sub-System
	SOPT1		XCSR	SPMSC1		SPMS C2		
	STOPE	WAITE	ENBDM <sup>1</sup>	LVDE	LVDSE	PPDC		
RUN mode — processor and peripherals clocked normally.	x	x	x	x	x	x	on. MCG in any mode	x
WAIT mode — processor clock nominally inactive, but peripherals are clocked.	x	1	x	x	x	x	Peripherals on and clocked. MCG in any mode	x
Stop modes disabled; illegal opcode reset if STOP instruction executed and CPUCR[IRD] is cleared, else an illegal instruction exception is generated.	0	0	function of BKGD /MS at reset	x	x	x	on. MCG in any mode	function of BKGD/MS at reset
STOP4 — low voltage detects are enabled or ENBDM = 1.	1	0	x	1	1	x	CPU clock on and peripheral clocks off if XCSR[ENBDM]=1	LVD enabled
			1	x	x			BDC clock enabled only if ENBDM=1 prior to entering stop.
STOP3 — Low voltage detect in STOP is disabled. If BDC is enabled, STOP4 will be invoked rather than STOP3.	1	0	0	x	0	0	MCG in stop mode. CPU and bus clocks are off. LPO, internal or external Ref clock can be enabled for RTC wakeup.	LVD disabled in stop only.
				0	x			LVD disabled in all modes.
STOP2 — Low Voltage Detect in stop is disabled. If BDC is enabled, STOP4 will be invoked rather than STOP2.	1	0	0	x	0	1	ICS powered off. LPO clock can be enabled for RTC wakeup.	LVD disabled in stop only. Only RAM powered.
				0	x			LVD disabled. Only RAM powered.

<sup>1</sup> ENBDM is located in the upper byte of the XCSR register which is write accessible only through BDC commands, see Debug module [Section 23.3.3, “Configuration/Status Register 2 \(CSR2\)”.](#)

### 3.4 Debug Mode

The debug interface is used to program a bootloader or user application program into the flash program memory before the MCU is operated in Run mode for the first time. When a MCF51JM128 series MCU is shipped from the Freescale Semiconductor factory, the flash program memory is erased by default unless

specifically noted, so there is no program that could 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.

For additional information about the debug interface, refer to [Chapter 23, “Version 1 ColdFire Debug \(CF1\\_DEBUG\).”](#)

## 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 23, “Version 1 ColdFire Debug \(CF1\\_DEBUG\),”](#) for details.

## 3.6 Run Mode

Run mode is the normal operating mode for the MCF51JM128 series MCUs. 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 in the memory map and executes code starting at the newly set PC value.

## 3.7 Wait Mode

Wait mode is entered by executing a STOP instruction after configuring the device as per [Table 3-1](#). If the WAITE control bit is set when STOP is executed, the Wait mode is entered. Upon 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. 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 instructions after in Wait mode.

The low voltage detector, if enabled, can be configured to interrupt the CPU and exit Wait mode into Run mode.

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

## 3.8 Stop Modes

One of three Stop modes are entered upon execution of a STOP instruction when SOPT1[STOPE] is set. SOPT1[WAITE] must be clear. In Stop3 mode, the bus and CPU clocks are halted. If the ENBDM bit 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 7, “Multipurpose Clock Generator \(S08MCGV3\)”](#) for more information.

## NOTE

If neither the WAITE nor STOPE bit is set when the CPU executes a STOP instruction, the MCU does not enter either of the stop modes. Instead, it initiates an illegal opcode reset (if CPUCR[IRD]=0) or generates an illegal instruction exception.

The Stop modes are selected by setting the appropriate bits in the SPMSC2 register. [Table 3-1](#) shows all of 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 be used to wake the MCU from Stop mode and enter halt mode if the ENBDM bit is set prior to entering Stop mode. After entering background debug mode, all background commands are available.

### 3.8.1 Stop2 Mode

Stop2 mode is entered by executing a STOP instruction under the conditions as shown in [Table 3-1](#). Most of the MCU internal circuitry is powered off in Stop2 mode, with the exception of the RAM. After entering Stop2 mode, all I/O pin control signals are latched so that the pins retain their states during Stop2 mode. Exiting from Stop2 mode is performed by asserting either wake-up pin: RESET or IRQ.

## NOTE

- The IRQ pin enable (IRQPE) control bit in IRQSC must be set, before entering stop2 mode for the IRQ pin to act as the wakeup source from stop2 mode.
- IRQ/TPMCLK always functions as an active-low wakeup input when the MCU is in stop2 mode, regardless of how the pin is configured before entering stop2 mode. The pullup on this pin is always disabled in stop2 mode. This pin must be driven or pulled high externally while in stop2 mode.

In addition, the RTC interrupt can wake the MCU from Stop2 mode, if enabled and using the low power oscillator (LPO). If the RTC is using the external clock source (EREFSTEN = 1) or internal clock source (IREFSTEN = 1), then the RTC is disabled in Stop2 mode.

After wake-up from Stop2 mode, the MCU starts up as from a power-on reset (POR):

- All module control and status registers are reset.
- The LVD reset function is enabled. The MCU remains in the reset state if VDD is below the LVD trip point (low trip point selected due to POR).
- The CPU takes the reset vector.

After waking up from Stop2 mode, 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]. To maintain I/O states for pins that were configured as general-purpose I/O before

entering Stop2 mode, software must restore the contents of the I/O port registers, which have been saved in RAM, to the port registers before writing to SPMSC2[PPDACK]. If the port registers are not restored from RAM before writing to SPMSC2[PPDACK], the pins switch to their reset states when SPMSC2[PPDACK] is written.

For pins that were configured as peripheral I/O, software must reconfigure the peripheral module that interfaces to the pin before writing to SPMSC2[PPDACK]. If the peripheral module is not enabled before writing to SPMSC2[PPDACK], the pins are controlled by their associated port control registers when the I/O latches are opened.

### 3.8.2 Stop3 Mode

Stop3 mode is entered by executing a STOP instruction under the conditions as shown in [Table 3-1](#) ( $\text{SPMSC2[PPDC]} = 0$  and  $\text{SPMSC1[LVDSE]} = 0$ ). The states of all of the internal registers and logic, RAM contents, and I/O pin states are maintained. Stop3 mode can be exited by asserting RESET, or by an interrupt from one of the following sources: the real-time clock (RTC) interrupt, the USB resume interrupt, the MSCAN wake-up interrupt, ADC, IRQ, KBI, or the ACMP. If Stop3 mode is exited by means of the RESET pin, the MCU is then reset and operation resumes after taking the reset vector. Exit by means of one of the internal interrupt sources results in the MCU taking the appropriate interrupt vector.

### 3.8.3 Stop4 Mode

Stop4 mode is entered by executing a STOP instruction shown in [Table 3-1](#). The states of all of the internal registers and logic, RAM contents, and I/O pin states are maintained. The MCG is configured to be running with the PLL or FLL engaged. Stop4 can be exited by asserting RESET, or by an interrupt from one of the following sources: the real-time clock (RTC) interrupt, the USB resume interrupt, the MSCAN wake-up interrupt, LVD, ADC, IRQ, KBI, or the ACMP. If stop4 is exited by means of the RESET pin, the MCU is then reset and operation resumes after taking the reset vector. Exit by means of one of the internal interrupt sources results in the MCU taking the appropriate interrupt vector.

#### 3.8.3.1 LVD Enabled in Stop Mode

The LVD is capable of generating an interrupt or a reset when the supply voltage drops below the LVD voltage. If the LVD is enabled in stop ( $\text{SPMSC1[LVDE]} \&\& \text{SPMSC1[LVDSE]} = 1$ ) at the time the CPU executes a STOP instruction, the voltage regulator then remains active during Stop mode. If the user attempts to enter Stop2 mode with the LVD enabled for Stop mode, the MCU enters Stop4 mode instead.

## 3.9 On-Chip Peripheral Modules in Stop and Wait Modes

When the MCU enters any stop mode (WAIT not included), system clocks to the internal peripheral modules stop. Even in the exception case 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 wait mode, system clocks to the internal peripheral modules continue based on the settings of the clock gating control registers (SCGCx).

**Table 3-2** defines terms used in **Table 3-3** to describe operation of components on the chip in the various low power modes.

**Table 3-2. Abbreviations used in Table 3-3**

Voltage Regulator	Clocked <sup>1</sup>	Not Clocked
Full Regulation	FullOn	FullNoClk FullADACK <sup>2</sup>
Loose 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 and 2 (SCGC1 and SCGC2).

<sup>2</sup> This ADC-specific mode defines the case where the device is fully regulated and the normal peripheral clock is stopped. In this case, the ADC can continue to run using its internally generated asynchronous ADACK clock.

<sup>3</sup> Analog modules must be in their low power mode when the device is operated in this state.

<sup>4</sup> This ADC-specific mode defines the case where the device is in soft regulation and the normal peripheral clock is stopped. In this case, the ADC can only be run using its low power mode and internally generated asynchronous ADACK clock.

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

Peripheral	Mode			
	STOP2	STOP3	STOP4	WAIT
<b>ADC<sup>1,2</sup></b>	Off	SoftADACK (Wake Up)	FULLADACK (Wake Up)	FullOn
<b>ACMP</b>	Off	SoftNoClk (Wake Up)	FullNoClk (Wake Up)	FullOn
<b>BDC</b>	Off	SoftOn	On	FullOn
<b>CAN</b>	Off	SoftNoClk	FullNoClk	FullOn
<b>CF1CORE</b>	Off	SoftNoClk	FullNoClk	FullNoClk
<b>CMT</b>	Off	SoftNoClk	FullNoClk	FullOn
<b>Crystal Oscillator (MCG)</b>	RANGE=0 HGO=0	RANGE=0 HGO=0	All Modes	All Modes
<b>COP</b>	Off	SoftNoClk	FullNoClk	FullOn
<b>Flash</b>	Off	SoftNoClk	FullNoClk	FullNoClk
<b>GPI/O Pins</b>	States Held	SoftNoClk	FullNoClk	FullOn
<b>IICx</b>	Off	SoftNoClk	FullNoClk	FullOn

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

Peripheral	Mode			
	STOP2	STOP3	STOP4	WAIT
<b>IRQ</b>	Off (Wake Up via POR) <sup>3</sup>	SoftNoClk (Wake Up)	FullNoClk (Wake Up)	FullOn
<b>KBI</b>	Off	SoftNoClk	FullNoClk	FullOn
<b>LVD/LVW</b>	Off	Disabled	On (Wake Up)	FullOn
<b>MCG</b>	Off	STOP or BLPE <sup>4</sup>	STOP or any mode	Any mode
<b>Port I/O Registers</b>	Off	SoftNoClk	FullNoClk	FullOn
<b>RAM</b>	SoftNoClk	SoftNoClk	FullNoClk	FullNoClk
<b>RTC</b>	Soft Regulation, LPOCLK if enabled (Wake Up via POR)	SoftOn LPOCLK or ICSERCLK (Wake Up)	Full Regulation LPOCLK, ICSERCLK or ICSIIRCLK only (Wake Up)	FullOn
<b>SCIx</b>	Off	SoftNoClk (Wake Up)	FullNoClk (Wake Up)	FullOn
<b>SPIx</b>	Off	SoftNoClk	FullNoClk	FullOn
<b>TPMx</b>	Off	SoftNoClk	FullNoClk	FullOn
<b>USB</b>	Off	SoftNoClk	FullNoClk	FullNoClk
<b>USB VREG</b>	Off	OptionallyOn <sup>5</sup>	OptionallyOn <sup>5</sup>	FullOn
<b>Voltage Regulator / PMC</b>	Partial Shutdown. 1kHz osc if enabled	Loose Regulation. 1kHz osc if enabled	Full Regulation 1kHz osc on	FullOn 1kHz osc on

<sup>1</sup> LP mode for the ADC is invoked by setting ADLPC=1. ADACK is selected via the ADCCFG[ADICLK] field in the ADC. See [Chapter 21, “Analog-to-Digital Converter \(S08ADC12V1\),”](#) for details.

<sup>2</sup> LVD must be enabled to run in stop if converting the bandgap channel.

<sup>3</sup> The RESET pin also has a direct connection to the on-chip regulator wakeup input. Asserting a low on this pin while in STOP2 will trigger the PMC to wakeup. As a result, the device will undergo a power-on-reset sequence.

<sup>4</sup> BLPE refers to the MCG “Bypassed Low Power External” state.

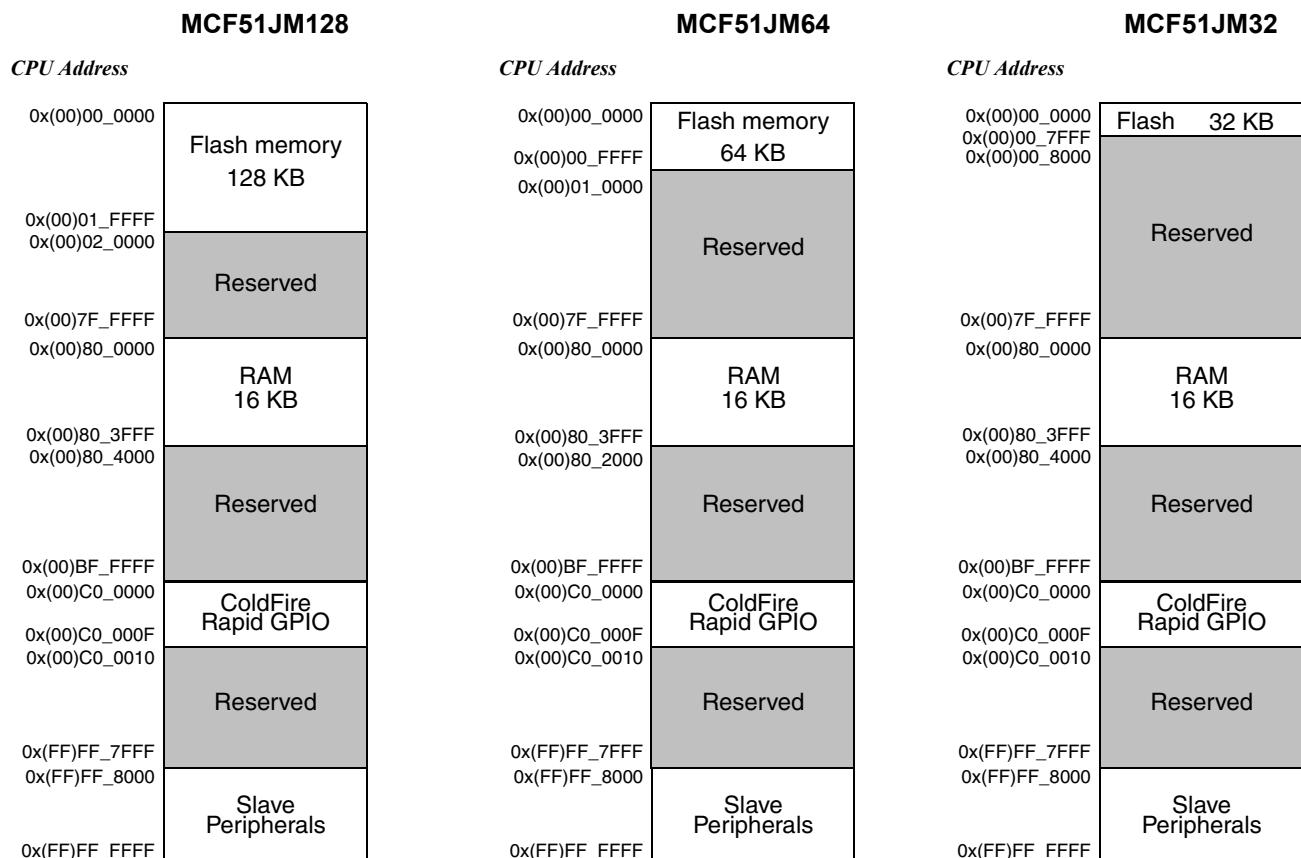
<sup>5</sup> USBVREN in USBTRC0 is set to enable USB 3.3V Regulator, else off

# Chapter 4

## Memory

### 4.1 MCF51JM128 Series Memory Map

As shown in [Figure 4-1](#), on-chip memory in the MCF51JM128 series microcontrollers consists of RAM and flash program memory for nonvolatile data storage, plus I/O and control/status registers.



**Figure 4-1. MCF51JM128 Series Memory Maps**

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 would typically generate a system reset in response to the error termination).

**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	Peripherals	x	x	x	x	x	x

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 into the following sub-sections:

0x(FF) FF\_8000 – 0x(FF) FF\_807F Direct-page peripheral regs

0x(FF) FF\_9800 – 0x(FF) FF\_98FF High-page peripheral regs

0x(FF) FF\_FFC0 – 0x(FF) FF\_FFFF Interrupt controller

The section of memory at 0x(00)C0\_0000 is assigned for use by the ColdFire Rapid GPIO module. See [Table 4-7](#) for the rapid GPIO memory map and [Chapter 10, “Rapid GPIO \(RGPIO\),”](#) for further details on the module.

The MCF51JM128 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. 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 Kbytes 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 Register Addresses and Bit Assignments

Peripheral registers in the MCF51JM128 series microcontrollers are divided into two groups:

- Direct-page registers are located at 0x(FF)FF\_8000 in the memory map.
- High-page registers are located at 0x(FF)FF\_9800 in the memory map.

There is no functional advantage to locating peripherals in the direct page versus the high page peripheral space for an MCF51JM128 series microcontroller. Both sets of registers may be efficiently accessed using the ColdFire absolute short addressing mode. The areas are differentiated to maintain documentation compatibility with the MC9S08JM60.

Peripheral register addresses for the MCF51JM128 series microcontrollers are shifted 0x(FF)FF\_8000 compared with the MC9S08JM60 devices.

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 as well as 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:

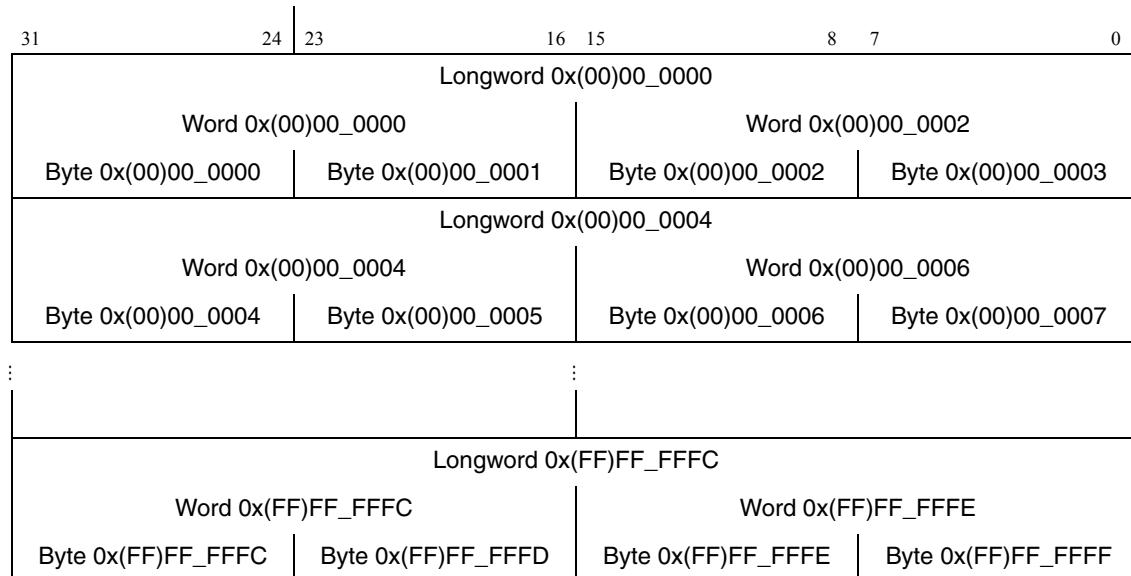
- NVPROT and NVOPT are loaded into working registers at reset
- An 8-byte backdoor comparison key that optionally allows a user to gain controlled access to secure memory

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

**Table 4-2** is a summary of all user-accessible direct-page registers and control bits.

In [Table 4-2](#), [Table 4-3](#), [Table 4-7](#) and [Table 4-8](#), the register names in column two are shown in bold to set them apart from the bit names to the right. Cells that are not associated with named bits are shaded. A shaded cell with a 0 indicates this unused bit always reads as a 0. Shaded cells with dashes indicate unused or reserved bit locations that could read as 1s or 0s. When writing to these bits, write a 0 unless otherwise specified.

Recall that ColdFire has a big endian byte addressable memory architecture. The most significant byte of each address is the lowest number as shown in [Figure 4-2](#). Multi-byte operands (16-bit words and 32-bit longwords) are referenced using an address pointing to the most significant (first) byte.



**Figure 4-2. ColdFire Memory Organization**

**Table 4-2. Direct-Page Register Summary**

Address	Name	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8000	<b>PTAD</b>	PTAD7	PTAD6	PTAD5	PTAD4	PTAD3	PTAD2	PTAD1	PTAD0
0x(FF)FF_8001	<b>PTADD</b>	PTADD7	PTADD6	PTADD5	PTADD4	PTADD3	PTADD2	PTADD1	PTADD0

**Table 4-2. Direct-Page Register Summary (continued)**

Address	Name	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8002	PTBD	PTBD7	PTBD6	PTBD5	PTBD4	PTBD3	PTBD2	PTBD1	PTBDO
0x(FF)FF_8003	PTBDD	PTBDD7	PTBDD6	PTBDD5	PTBDD4	PTBDD3	PTBDD2	PTBDD1	PTBDD0
0x(FF)FF_8004	PTCD	PTCD7	PTCD6	PTCD5	PTCD4	PTCD3	PTCD2	PTCD1	PTCD0
0x(FF)FF_8005	PTCDD	PTCDD7	PTCDD6	PTCDD5	PTCDD4	PTCDD3	PTCDD2	PTCDD1	PTCDD0
0x(FF)FF_8006	PTDD	PTDD7	PTDD6	PTDD5	PTDD4	PTDD3	PTDD2	PTDD1	PTDD0
0x(FF)FF_8007	PTDDD	PTDDD7	PTDDD6	PTDDD5	PTDDD4	PTDDD3	PTDDD2	PTDDD1	PTDDD0
0x(FF)FF_8008	PTED	PTED7	PTED6	PTED5	PTED4	PTED3	PTED2	PTED1	PTED0
0x(FF)FF_8009	PTEDD	PTEDD7	PTEDD6	PTEDD5	PTEDD4	PTEDD3	PTEDD2	PTEDD1	PTEDD0
0x(FF)FF_800A	PTFD	PTFD7	PTFD6	PTFD5	PTFD4	PTFD3	PTFD2	PTFD1	PTFDO
0x(FF)FF_800B	PTFDD	PTFDD7	PTFDD6	PTFDD5	PTFDD4	PTFDD3	PTFDD2	PTFDD1	PTFDD0
0x(FF)FF_800C	PTGD	PTGD7	PTGD6	PTGD5	PTGD4	PTGD3	PTGD2	PTGD1	PTGDO
0x(FF)FF_800D	PTGDD	PTGDD7	PTGDD6	PTGDD5	PTGDD4	PTGDD3	PTGDD2	PTGDD1	PTGDD0
0x(FF)FF_800E	ACMPSC	ACME	ACBGS	ACF	ACIE	ACO	ACOPE	ACMOD1	ACMOD0
0x(FF)FF_800F	Reserved	0	0	0	0	0	0	0	0
0x(FF)FF_8010	ADCSC1	COCO	AIEN	ADCO	ADCH				
0x(FF)FF_8011	ADCSC2	ADACT	ADTRG	ACFE	ACFGT	—	—	—	—
0x(FF)FF_8012	ADCRH	0	0	0	0	ADR11	ADR10	ADR9	ADR8
0x(FF)FF_8013	ADCRL	ADR7	ADR6	ADR5	ADR4	ADR3	ADR2	ADR1	ADR0
0x(FF)FF_8014	ADCCVH	0	0	0	0	ADCV11	ADCV10	ADCV9	ADCV8
0x(FF)FF_8015	ADCCVL	ADCV7	ADCV6	ADCV5	ADCV4	ADCV3	ADCV2	ADCV1	ADCV0
0x(FF)FF_8016	ADCCFG	ADLPC	ADIV		ADLSMP	MODE		ADICLK	
0x(FF)FF_8017	APCTL1	ADPC7	ADPC6	ADPC5	ADPC4	ADPC3	ADPC2	ADPC1	ADPC0
0x(FF)FF_8018	APCTL2	—	—	—	—	ADPC11	ADPC10	ADPC9	ADPC8
0x(FF)FF_8019	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_801A	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_801B	IRQSC	0	IRQPDD	IRQEDG	IRQPE	IRQF	IRQACK	IRQIE	IRQMOD
0x(FF)FF_801C	KBI1SC	0	0	0	0	KB1F	KB1ACK	KB1IE	KBI1MOD
0x(FF)FF_801D	KBI1PE	KBI1PE7	KBI1PE6	KBI1PE5	KBI1PE4	KBI1PE3	KBI1PE2	KBI1PE1	KBI1PE0
0x(FF)FF_801E	KBI1ES	KB1EDG7	KB1EDG6	KB1EDG5	KB1EDG4	KB1EDG3	KB1EDG2	KB1EDG1	KB1EDG0
0x(FF)FF_801F	Reserved	0	0	0	0	0	0	0	0
0x(FF)FF_8020	TPM1SC	TOF	TOIE	CPWMS	CLKSB	CLKSA	PS2	PS1	PS0
0x(FF)FF_8021	TPM1CNTH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_8022	TPM1CNTL	Bit 7	6	5	4	3	2	1	Bit 0

**Table 4-2. Direct-Page Register Summary (continued)**

Address	Name	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8023	TPM1MODH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_8024	TPM1MODL	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8025	TPM1C0SC	CH0F	CH0IE	MS0B	MS0A	ELS0B	ELS0A	0	0
0x(FF)FF_8026	TPM1C0VH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_8027	TPM1C0VL	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8028	TPM1C1SC	CH1F	CH1IE	MS1B	MS1A	ELS1B	ELS1A	0	0
0x(FF)FF_8029	TPM1C1VH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_802A	TPM1C1VL	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_802B	TPM1C2SC	CH2F	CH2IE	MS2B	MS2A	ELS2B	ELS2A	0	0
0x(FF)FF_802C	TPM1C2VH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_802D	TPM1C2VL	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_802E	TPM1C3SC	CH3F	CH3IE	MS3B	MS3A	ELS3B	ELS3A	0	0
0x(FF)FF_802F	TPM1C3VH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_8030	TPM1C3VL	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8031	TPM1C4SC	CH4F	CH4IE	MS4B	MS4A	ELS4B	ELS4A	0	0
0x(FF)FF_8032	TPM1C4VH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_8033	TPM1C4VL	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8034	TPM1C5SC	CH5F	CH5IE	MS5B	MS5A	ELS5B	ELS5A	0	0
0x(FF)FF_8035	TPM1C5VH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_8036	TPM1C5VL	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8037	Reserved	0	0	0	0	0	0	0	0
0x(FF)FF_8038	SCI1BDH	LBKDI	RXEDGIE	0	SBR12	SBR11	SBR10	SBR9	SBR8
0x(FF)FF_8039	SCI1BDL	SBR7	SBR6	SBR5	SBR4	SBR3	SBR2	SBR1	SBR0
0x(FF)FF_803A	SCI1C1	LOOPS	SCISWAI	RSRC	M	WAKE	ILT	PE	PT
0x(FF)FF_803B	SCI1C2	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
0x(FF)FF_803C	SCI1S1	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF
0x(FF)FF_803D	SCI1S2	LBKDIF	RXEDGIF	0	RXINV	RWUID	BRK13	LBKDE	RAF
0x(FF)FF_803E	SCI1C3	R8	T8	TXDIR	TXINV	ORIE	NEIE	FEIE	PEIE
0x(FF)FF_803F	SCI1D	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8040	SCI2BDH	LBKDI	RXEDGIE	0	SBR12	SBR11	SBR10	SBR9	SBR8
0x(FF)FF_8041	SCI2BDL	SBR7	SBR6	SBR5	SBR4	SBR3	SBR2	SBR1	SBR0
0x(FF)FF_8042	SCI2C1	LOOPS	SCISWAI	RSRC	M	WAKE	ILT	PE	PT
0x(FF)FF_8043	SCI2C2	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK

Table 4-2. Direct-Page Register Summary (continued)

Address	Name	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8044	SCI2S1	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF
0x(FF)FF_8045	SCI2S2	LBKDIF	RXEDGIF	0	RXINV	RWUID	BRK13	LBKDE	RAF
0x(FF)FF_8046	SCI2C3	R8	T8	TXDIR	TXINV	ORIE	NEIE	FEIE	PEIE
0x(FF)FF_8047	SCI2D	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8048	MCGC1	CLKS		RDIV			IREFS	IRCLKEN	IREFSTEN
0x(FF)FF_8049	MCGC2	BDIV		RANGE	HGO	LP	EREFs	ERCLKEN	EREFSTEN
0x(FF)FF_804A	MCGTRM	TRIM							
0x(FF)FF_804B	MCGSC	LOLS	LOCK	PLLST	IREFST	CLKST		OSCINIT	FTRIM
0x(FF)FF_804C	MCGC3	LOLIE	PLLS	CME	DIV32	VDIV			
0x(FF)FF_804D	MCGC4	0	0	DMX32	0	0	0	DRST / DRS	
0x(FF)FF_804E- 0x(FF)FF_804F	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_8050	SPI1C1	SPIE	SPE	SPTIE	MSTR	CPOL	CPHA	SSOE	LSBFE
0x(FF)FF_8051	SPI1C2	SPMIE	SPIMODE	0	MODFEN	BIDIROE	0	SPISWAI	SPC0
0x(FF)FF_8052	SPI1BR	0	SPPR2	SPPR1	SPPR0	0	SPR2	SPR1	SPR0
0x(FF)FF_8053	SPI1S	SPRF	0	SPTEF	MODF	0	0	0	0
0x(FF)FF_8054	SPI1DH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_8055	SPI1DL	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8056	SPI1MH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_8057	SPI1ML	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8058	IIC1A	AD7	AD6	AD5	AD4	AD3	AD2	AD1	0
0x(FF)FF_8059	IIC1F	MULT		ICR					
0x(FF)FF_805A	IIC1C1	IICEN	IICIE	MST	TX	TXAK	RSTA	0	0
0x(FF)FF_805B	IIC1S	TCF	IAAS	BUSY	ARBL	0	SRW	IICIF	RXAK
0x(FF)FF_805C	IIC1D	DATA							
0x(FF)FF_805D	IIC1C2	GCAEN	ADEXT	0	0	0	AD10	AD9	AD8
0x(FF)FF_805E	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_805F	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_8060	TPM2SC	TOF	TOIE	CPWMS	CLKSB	CLKSA	PS2	PS1	PS0
0x(FF)FF_8061	TPM2CNTH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_8062	TPM2CNTL	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8063	TPM2MODH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_8064	TPM2MODL	Bit 7	6	5	4	3	2	1	Bit 0

**Table 4-2. Direct-Page Register Summary (continued)**

Address	Name	Bit 7	6	5	4	3	2	1	Bit 0						
0x(FF)FF_8065	TPM2C0SC	CH0F	CH0IE	MS0B	MS0A	ELS0B	ELS0A	0	0						
0x(FF)FF_8066	TPM2C0VH	Bit 15	14	13	12	11	10	9	Bit 8						
0x(FF)FF_8067	TPM2C0VL	Bit 7	6	5	4	3	2	1	Bit 0						
0x(FF)FF_8068	TPM2C1SC	CH1F	CH1IE	MS1B	MS1A	ELS1B	ELS1A	0	0						
0x(FF)FF_8069	TPM2C1VH	Bit 15	14	13	12	11	10	9	Bit 8						
0x(FF)FF_806A	TPM2C1VL	Bit 7	6	5	4	3	2	1	Bit 0						
0x(FF)FF_806B	Reserved	—	—	—	—	—	—	—	—						
0x(FF)FF_806C	RTCSC	RTIF	RTCLKS		RTIE	RTCPS									
0x(FF)FF_806D	RTCCNT	RTCCNT													
0x(FF)FF_806E	RTCMOD	RTCMOD													
0x(FF)FF_806F	Reserved	—	—	—	—	—	—	—	—						
0x(FF)FF_8070	SPI2C1	SPIE	SPE	SPTIE	MSTR	CPOL	CPHA	SSOE	LSBFE						
0x(FF)FF_8071	SPI2C2	SPMIE	SPIMODE	0	MODFEN	BIDIROE	0	SPISWAI	SPC0						
0x(FF)FF_8072	SPI2BR	0	SPPR2	SPPR1	SPPR0	0	SPR2	SPR1	SPR0						
0x(FF)FF_8073	SPI2S	SPRF	SPMF	SPTEF	MODF	RNFULLF	TNEAREF	TXFULLF	RFIFOFEF						
0x(FF)FF_8074	SPI2DH	Bit 15	14	13	12	11	10	9	Bit 8						
0x(FF)FF_8075	SPI2DL	Bit 7	6	5	4	3	2	1	Bit 0						
0x(FF)FF_8076	SPI2MH	Bit 15	14	13	12	11	10	9	Bit 8						
0x(FF)FF_8077	SPI2ML	Bit 7	6	5	4	3	2	1	Bit 0						
0x(FF)FF_8078	SPI2C3	0	0	TNEAREF MARK	RNFULL MARK	0	TNEARIE N	RNFULLI EN	FIFOMODE						
0x(FF)FF_8079– 0x(FF)FF_80FF	Reserved	—	—	—	—	—	—	—	—						

**Table 4-3. High-Page Register Summary**

Address	Name	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_9800	SRS	POR	PIN	COP	ILOP	ILAD	LOC	LVD	0
0x(FF)FF_9801	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_9802	SOPT1	COPT1	COPT0	STOPE	WAITE	0	0	0	0
0x(FF)FF_9803	SOPT2	COPCLK S	COPW	USB_BIG END	CLKOUT _EN	CMT_CL K_SEL	SPI1FE	SPI2FE	ACIC
0x(FF)FF_9804– 0x(FF)FF_9805	Reserved	—	—	—	—	—	—	—	—

**Table 4-3. High-Page Register Summary (continued)**

Address	Name	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_9806	SDIDH	REV				ID11	ID10	ID9	ID8
0x(FF)FF_9807	SDIDL	ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0
0x(FF)FF_9808	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_9809	SPMSC1	LWVF	LWVACK	LWVIE	LVDRE	LVDSE	LVDE	0	BGBE
0x(FF)FF_980A	SPMSC2	0	0	LVDV	LVWV	PPDF	PPDACK	0	PPDC
0x(FF)FF_980B	SCGC1	CMT	TPM2	TPM1	ADC	IIC2	IIC1	SCI2	SCI1
0x(FF)FF_980C	SCGC2	USB	FLS	IRQ	KBI	ACMP	RTC	SPI2	SPI1
0x(FF)FF_980D	SCGC3	—	—	—	—	—	—	—	RNGA
0x(FF)FF_980E	SOPT3	—	—	—	—	—	—	—	CMT_PA D
0x(FF)FF_980F	SOPT4	—	—	—	—	PMC_LVD_TRIM[3:0]			
0x(FF)FF_9810- 0x(FF)FF_981F	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_9820	FCDIV	FDIVLD	PRDIV8	FDIV					
0x(FF)FF_9821	FOPT	KEYEN		0	0	0	0	SEC	
0x(FF)FF_9822	FRSV0 (Reserved)	0	0	0	0	0	0	0	0
0x(FF)FF_9823	FCNFG	0	0	KEYACC	0	0	0	0	0
0x(FF)FF_9824	FPROT	FPS						FPOOPEN	
0x(FF)FF_9825	FSTAT	FCBEF	FCCF	FPVIOL	FACCERR	0	FBLANK	0	0
0x(FF)FF_9826	FCMD	0	FCMD						
0x(FF)FF_9827	FRSV1 (Reserved)	0	0	0	0	0	0	0	0
0x(FF)FF_9828	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_9829	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_982A	FRSV2 (Reserved)	0	0	0	0	0	0	0	0
0x(FF)FF_982B	FRSV3 (Reserved)	0	0	0	0	0	0	0	0
0x(FF)FF_982C- 0x(FF)FF_9837	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_9838	IIC2A	AD7	AD6	AD5	AD4	AD3	AD2	AD1	0
0x(FF)FF_9839	IIC2F	MULT				ICR			
0x(FF)FF_983A	IIC2C1	IICEN	IICIE	MST	TX	TXAK	RSTA	0	0
0x(FF)FF_983B	IIC2S	TCF	IAAS	BUSY	ARBL	0	SRW	IICIF	RXAK

**Table 4-3. High-Page Register Summary (continued)**

Address	Name	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_983C	IIC2D								
0x(FF)FF_983D	IIC2C2	GCAEN	ADEXT	0	0	0	AD10	AD9	AD8
0x(FF)FF_983E- 0x(FF)FF_983F	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_9840	PTAPE	PTAPE7	PTAPE6	PTAPE5	PTAPE4	PTAPE3	PTAPE2	PTAPE1	PTAPE0
0x(FF)FF_9841	PTASE	PTASE7	PTASE6	PTASE5	PTASE4	PTASE3	PTASE2	PTASE1	PTASE0
0x(FF)FF_9842	PTADS	PTADS7	PTADS6	PTADS5	PTADS4	PTADS3	PTADS2	PTADS1	PTADS0
0x(FF)FF_9843	PTAIFE	PTAIFE7	PTAIFE6	PTAIFE5	PTAIFE4	PTAIFE3	PTAIFE2	PTAIFE1	PTAIFE0
0x(FF)FF_9844	PTBPE	PTBPE7	PTBPE6	PTBPE5	PTBPE4	PTBPE3	PTBPE2	PTBPE1	PTBPE0
0x(FF)FF_9845	PTBSE	PTBSE7	PTBSE6	PTBSE5	PTBSE4	PTBSE3	PTBSE2	PTBSE1	PTBSE0
0x(FF)FF_9846	PTBDS	PTBDS7	PTBDS6	PTBDS5	PTBDS4	PTBDS3	PTBDS2	PTBDS1	PTBDS0
0x(FF)FF_9847	PTBIFE	PTBIFE7	PTBIFE6	PTBIFE5	PTBIFE4	PTBIFE3	PTBIFE2	PTBIFE1	PTBIFE0
0x(FF)FF_9848	PTCPE	PTCPE7	PTCPE6	PTCPE5	PTCPE4	PTCPE3	PTCPE2	PTCPE1	PTCPE0
0x(FF)FF_9849	PTCSE	PTCSE7	PTCSE6	PTCSE5	PTCSE4	PTCSE3	PTCSE2	PTCSE1	PTCSE0
0x(FF)FF_984A	PTCDS	PTCDS7	PTCDS6	PTCDS5	PTCDS4	PTCDS3	PTCDS2	PTCDS1	PTCDS0
0x(FF)FF_984B	PTCIFE	PTCIFE7	PTCIFE6	PTCIFE5	PTCIFE4	PTCIFE3	PTCIFE2	PTCIFE1	PTCIFE0
0x(FF)FF_984C	PTDPE	PTDPE7	PTDPE6	PTDPE5	PTDPE4	PTDPE3	PTDPE2	PTDPE1	PTDPE0
0x(FF)FF_984D	PTDSE	PTDSE7	PTDSE6	PTDSE5	PTDSE4	PTDSE3	PTDSE2	PTDSE1	PTDSE0
0x(FF)FF_984E	PTDDS	PTDDS7	PTDDS6	PTDDS5	PTDDS4	PTDDS3	PTDDS2	PTDDS1	PTDDS0
0x(FF)FF_984F	PTDIFE	PTDIFE7	PTDIFE6	PTDIFE5	PTDIFE4	PTDIFE3	PTDIFE2	PTDIFE1	PTDIFE0
0x(FF)FF_9850	PTEPE	PTEPE7	PTEPE6	PTEPE5	PTEPE4	PTEPE3	PTEPE2	PTEPE1	PTEPE0
0x(FF)FF_9851	PTESE	PTESE7	PTESE6	PTESE5	PTESE4	PTESE3	PTESE2	PTESE1	PTESE0
0x(FF)FF_9852	PTEDS	PTEDS7	PTEDS6	PTEDS5	PTEDS4	PTEDS3	PTEDS2	PTEDS1	PTEDS0
0x(FF)FF_9853	PTEIFE	PTEIFE7	PTEIFE6	PTEIFE5	PTEIFE4	PTEIFE3	PTEIFE2	PTEIFE1	PTEIFE0
0x(FF)FF_9854	PTFPE	PTFPE7	PTFPE6	PTFPE5	PTFPE4	PTFPE3	PTFPE2	PTFPE1	PTFPE0
0x(FF)FF_9855	PTFSE	PTFSE7	PTFSE6	PTFSE5	PTFSE4	PTFSE3	PTFSE2	PTFSE1	PTFSE0
0x(FF)FF_9856	PTFDS	PTFDS7	PTFDS6	PTFDS5	PTFDS4	PTFDS3	PTFDS2	PTFDS1	PTFDS0
0x(FF)FF_9857	PTFIFE	PTFIFE7	PTFIFE6	PTFIFE5	PTFIFE4	PTFIFE3	PTFIFE2	PTFIFE1	PTFIFE0
0x(FF)FF_9858	PTGPE	PTGPE7	PTGPE6	PTGPE5	PTGPE4	PTGPE3	PTGPE2	PTGPE1	PTGPE0
0x(FF)FF_9859	PTGSE	PTGSE7	PTGSE6	PTGSE5	PTGSE4	PTGSE3	PTGSE2	PTGSE1	PTGSE0
0x(FF)FF_985A	PTGDS	PTGDS7	PTGDS6	PTGDS5	PTGDS4	PTGDS3	PTGDS2	PTGDS1	PTGDS0
0x(FF)FF_985B	PTGIFE	PTGIFE7	PTGIFE6	PTGIFE5	PTGIFE4	PTGIFE3	PTGIFE2	PTGIFE1	PTGIFE0
0x(FF)FF_985C	PTHPE	—	—	—	PTHPE4	PTHPE3	PTHPE2	PTHPE1	PTHPE0
DATA									

**Table 4-3. High-Page Register Summary (continued)**

Address	Name	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_985D	PTHSE	—	—	—	PTHSE4	PTHSE3	PTHSE2	PTHSE1	PTHSE0
0x(FF)FF_985E	PTHDS	—	—	—	PTHDS4	PTHDS3	PTHDS2	PTHDS1	PTHDS0
0x(FF)FF_985F	PTHIFE	—	—	—	PTHIFE4	PTHIFE3	PTHIFE2	PTHIFE1	PTHIFE0
0x(FF)FF_9860	PTJPE	—	—	—	PTJPE4	PTJPE3	PTJPE2	PTJPE1	PTJPE0
0x(FF)FF_9861	PTJSE	—	—	—	PTJSE4	PTJSE3	PTJSE2	PTJSE1	PTJSE0
0x(FF)FF_9862	PTJDS	—	—	—	PTJDS4	PTJDS3	PTJDS2	PTJDS1	PTJDS0
0x(FF)FF_9863	PTJIFE	—	—	—	PTJIFE4	PTJIFE3	PTJIFE2	PTJIFE1	PTJIFE0
0x(FF)FF_9864– 0x(FF)FF_9867	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_9868	CMTCGH1	PH7	PH6	PH5	PH4	PH3	PH2	PH1	PH0
0x(FF)FF_9869	CMTCGL1	PL7	PL6	PL5	PL4	PL3	PL2	PL1	PL0
0x(FF)FF_986A	CMTCGH2	SH7	SH6	SH5	SH4	SH3	SH2	SH1	SH0
0x(FF)FF_986B	CMTCGL2	SL7	SL6	SL5	SL4	SL3	SL2	SL1	SL0
0x(FF)FF_986C	CMTOC	IROL	CMTPOL	IROPEN	0	0	0	0	0
0x(FF)FF_986D	CMTMSC	EOCF	CMTDIV1	CMTDIV0	EXSPC	BASE	FSK	EOCIE	MCGEN
0x(FF)FF_986E	CMTCMD1	MB15	MB14	MB13	MB12	MB11	MB10	MB9	MB8
0x(FF)FF_986F	CMTCMD2	MB7	MB6	MB5	MB4	MB3	MB2	MB1	MB0
0x(FF)FF_9870	CMTCMD3	SB15	SB14	SB13	SB12	SB11	SB10	SB9	SB8
0x(FF)FF_9871	CMTCMD4	SB7	SB6	SB5	SB4	SB3	SB2	SB1	SB0
0x(FF)FF_9872	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_9873	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_9874	PTHD	—	—	—	PTHD4	PTHD3	PTHD2	PTHD1	PTHD0
0x(FF)FF_9875	PTHDD	—	—	—	PTHDD4	PTHDD3	PTHDD2	PTHDD1	PTHDD0
0x(FF)FF_9876	PTJD	—	—	—	PTJD4	PTJD3	PTJD2	PTJD1	PTJD0
0x(FF)FF_9877	PTJDD	—	—	—	PTJDD4	PTJDD3	PTJDD2	PTJDD1	PTJDD0
0x(FF)FF_9878– 0x(FF)FF_987F	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_9880	CANCTL0	RXFRM	RXACT	CSWAI	SYNCH	TIME	WUPE	SLPRQ	INITRQ
0x(FF)FF_9881	CANCTL1	CANE	CLKSRC	LOOPB	LISTEN	BORM	WUPM	SLPAK	INITAK
0x(FF)FF_9882	CANBTR0	SJW1	SJW0	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0
0x(FF)FF_9883	CANBTR1	SAMP	TSEG22	TSEG21	TSEG20	TESEG13	TESEG12	TSEG11	TSEG10
0x(FF)FF_9884	CANRFLG	WUPIF	CSCIF	RSTAT1	RSTAT0	TSTAT1	TSTAT0	OVRIF	RXF
0x(FF)FF_9885	CANRIER	WUPIE	CSCIE	RSTATE1	RSTATE0	TSTATE1	TSTATE0	OVRIE	RXFIE

**Table 4-3. High-Page Register Summary (continued)**

Address	Name	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_9886	CANTFLG	—	—	—	—	—	TXE2	TXE1	TXE0
0x(FF)FF_9887	CANTIER	—	—	—	—	—	TXEIE2	TXEIE1	TXEIE0
0x(FF)FF_9888	CANTARQ	—	—	—	—	—	ABTRQ2	ABTRQ1	ABTRQ0
0x(FF)FF_9889	CANTAAK	—	—	—	—	—	ABTAK2	ABTAK1	ABTAK0
0x(FF)FF_988A	CANTBSEL	—	—	—	—	—	TX2	TX1	TX0
0x(FF)FF_988B	CANIDAC	—	—	IDAM1	IDAM0	—	IDHIT2	IDHIT1	IDHIT0
0x(FF)FF_988C	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_988D	CANMISC	—	—	—	—	—	—	—	BOHOLD
0x(FF)FF_988E	CANRXERR	RXERR7	RXERR6	RXERR5	RXERR4	RXERR3	RXERR2	RXERR1	RXERR0
0x(FF)FF_988F	CANTXERR	TXERR7	TXERR6	TXERR5	TXERR4	TXERR3	TXERR2	TXERR1	TXERR0
0x(FF)FF_9890	CANIDAR0	Acceptance Code Bits [7:0]							
0x(FF)FF_9891	CANIDAR1	Acceptance Code Bits [7:0]							
0x(FF)FF_9892	CANIDAR2	Acceptance Code Bits [7:0]							
0x(FF)FF_9893	CANIDAR3	Acceptance Code Bits [7:0]							
0x(FF)FF_9894	CANIDMR0	Acceptance Mask Bits [7:0]							
0x(FF)FF_9895	CANIDMR1	Acceptance Mask Bits [7:0]							
0x(FF)FF_9896	CANIDMR2	Acceptance Mask Bits [7:0]							
0x(FF)FF_9897	CANIDMR3	Acceptance Mask Bits [7:0]							
0x(FF)FF_9898	CANIDAR4	Acceptance Code Bits [7:0]							
0x(FF)FF_9899	CANIDAR5	Acceptance Code Bits [7:0]							
0x(FF)FF_989A	CANIDAR6	Acceptance Code Bits [7:0]							
0x(FF)FF_989B	CANIDAR7	Acceptance Code Bits [7:0]							
0x(FF)FF_989C	CANIDMR4	Acceptance Mask Bits [7:0]							
0x(FF)FF_989D	CANIDMR5	Acceptance Mask Bits [7:0]							
0x(FF)FF_989E	CANIDMR6	Acceptance Mask Bits [7:0]							
0x(FF)FF_989F	CANIDMR7	Acceptance Mask Bits [7:0]							
0x(FF)FF_98A0	CANRIDR0	Receiver Identifier Register 0							
0x(FF)FF_98A1	CANRIDR1	Receiver Identifier Register 1							
0x(FF)FF_98A2	CANRIDR2	Receiver Identifier Register 2							
0x(FF)FF_98A3	CANRIDR3	Receiver Identifier Register 3							
0x(FF)FF_98A4	CANRDSR0	Receiver Data Segment Register 0							
0x(FF)FF_98A5	CANRDSR1	Receiver Data Segment Register 1							
0x(FF)FF_98A6	CANRDSR2	Receiver Data Segment Register 2							

**Table 4-3. High-Page Register Summary (continued)**

Address	Name	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_98A7	CANRDSR3	Receiver Data Segment Register 3							
0x(FF)FF_98A8	CANRDSR4	Receiver Data Segment Register 4							
0x(FF)FF_98A9	CANRDSR5	Receiver Data Segment Register 5							
0x(FF)FF_98AA	CANRDSR6	Receiver Data Segment Register 6							
0x(FF)FF_98AB	CANRDSR7	Receiver Data Segment Register 7							
0x(FF)FF_98AC	CANRDLR	—	—	—	—	DLC3	DLC2	DLC1	DLC0
0x(FF)FF_98AD	CANTBPR	PRI07	PRI06	PRI05	PRI04	PRI03	PRI02	PRI01	PRI00
0x(FF)FF_98AE	CANRTSRH	TSR15	TSR14	TSR13	TSR12	TSR11	TSR10	TSR9	TSR8
0x(FF)FF_98AF	CANRTSRL	TSR7	TSR6	TSR5	TSR4	TSR3	TSR2	TSR1	TSR0
0x(FF)FF_98B0	CANTIDR0	Transmit Identifier Register 0							
0x(FF)FF_98B1	CANTIDR1	Transmit Identifier Register 1							
0x(FF)FF_98B2	CANTIDR2	Transmit Identifier Register 2							
0x(FF)FF_98B3	CANTIDR3	Transmit Identifier Register 3							
0x(FF)FF_98B4	CANTIDSR0	Transmit Data Segment Register 0							
0x(FF)FF_98B5	CANTIDSR1	Transmit Data Segment Register 1							
0x(FF)FF_98B6	CANTIDSR2	Transmit Data Segment Register 2							
0x(FF)FF_98B7	CANTIDSR3	Transmit Data Segment Register 3							
0x(FF)FF_98B8	CANTIDSR4	Transmit Data Segment Register 4							
0x(FF)FF_98B9	CANTIDSR5	Transmit Data Segment Register 5							
0x(FF)FF_98BA	CANTIDSR6	Transmit Data Segment Register 6							
0x(FF)FF_98BB	CANTIDSR7	Transmit Data Segment Register 7							
0x(FF)FF_98BC	CANTDLR	—	—	—	—	DLC3	DLC2	DLC1	DLC0
0x(FF)FF_98BD	CANTTBPR	PRI07	PRI06	PRI05	PRI04	PRI03	PRI02	PRI01	PRI00
0x(FF)FF_98BE	CANTTSRH	TSR15	TSR14	TSR13	TSR12	TSR11	TSR10	TSR9	TSR8
0x(FF)FF_98BF	CANTTSRL	TSR7	TSR6	TSR5	TSR4	TSR3	TSR2	TSR1	TSR0
0x(FF)FF_98C0	RNGCR [31:24]	0	0	0	0	0	0	0	0
0x(FF)FF_98C1	RNGCR [23:16]	0	0	0	0	0	0	0	0
0x(FF)FF_98C2	RNGCR [15:8]	0	0	0	0	0	0	0	0
0x(FF)FF_98C3	RNGCR [7:0]	0	0	0	SLM	CI	IM	HA	GO
0x(FF)FF_98C4	RNGSR [31:24]	OD	0	0	0	0	0	0	0

**Table 4-3. High-Page Register Summary (continued)**

Address	Name	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_98C5	RNGSR [23:16]								ORS register
0x(FF)FF_98C6	RNGSR [15:8]								ORL register
0x(FF)FF_98C7	RNGSR [7:0]	0	0	0	SLP	EI	OUF	LRS	SV
0x(FF)FF_98C8	RNGER [31:24]								ENT [31:24]
0x(FF)FF_98C9	RNGER [23:16]								ENT [23:16]
0x(FF)FF_98CA	RNGER [15:8]								ENT [15:8]
0x(FF)FF_98CB	RNGER [7:0]								ENT [7:0]
0x(FF)FF_98CC	RNGOUT [31:24]								RANDOM_OUTPUT [31:24]
0x(FF)FF_98CD	RNGOUT [23:16]								RANDOM_OUTPUT [23:16]
0x(FF)FF_98CE	RNGOUT [15:8]								RANDOM_OUTPUT [15:8]
0x(FF)FF_98CF	RNGOUT [7:0]								RANDOM_OUTPUT [7:0]
0x(FF)FF_98D0– 0x(FF)FF_98E8	Reserved	—	—	—	—	—	—	—	—

**Table 4-4. USB Register Summary**

Address	Name	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_9A00	PER_ID	0	0	ID5	ID4	ID3	ID2	ID1	ID0
0x(FF)FF_9A04	ID_COMP	1	1	NID5	NID4	NID3	NID2	NID1	NID0
0x(FF)FF_9A08	REV	REV7	REV6	REV5	REV4	REV3	REV2	REV1	REV0
0x(FF)FF_9A0C	ADD_INFO				IRQ_NUM		0	0	IEHOST
0x(FF)FF_9A10	OTG_INT_STAT	ID_CHG	1_MSEC	LINE_STATE_CHG	—	SESS_VLD_CHG	B_SESS_CHG	—	A_VBUS_CHG
0x(FF)FF_9A14	OTG_INT_EN	ID_EN	1_MSEC_EN	LINE_STATE_EN	—	SESS_VLD_EN	B_SESS_EN	—	A_VBUS_EN

Table 4-4. USB Register Summary (continued)

Address	Name	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_9A18	OTG_STAT	ID	1_MSEC_EN	LINE_STATE_STABLE	—	SESS_VLD	B_SESS_END	—	A_VBUS_VLD
0x(FF)FF_9A1C	OTG_CTRL	DP_HIGH	—	DP_LOW	DM_LOW	—	OTG_EN	—	—
0x(FF)FF_9A80	INT_STAT	STALL	ATTACH	RESUME	SLEEP	TOK_DNE	SOF_TOK	ERROR	USB_RST
0x(FF)FF_9A84	INT_ENB	STALL_EN	ATTACH_EN	RESUME_EN	SLEEP_EN	TOK_DNE_EN	SOF_TOK_EN	ERROR_EN	USB_RST_EN
0x(FF)FF_9A88	ERR_STAT	BTS_ERR	—	DMA_ERR	BTO_ERR	DFN8	CRC16	CRC5_EOF	PID_ERR
0x(FF)FF_9A8C	ERR_ENB	BTS_ERR_EN	—	DMA_ERR_EN	BTO_ERR_EN	DFN8_EN	CRC16_EN	CRC5_EOF_EN	PID_ERR_EN
0x(FF)FF_9A90	STAT	ENDP [3:0]				TX	ODD	—	—
0x(FF)FF_9A94	CTL	JSTATE	SE0	TXSUSP END/TOKEN BUSY	RESET	HOST_MODE_EN	RESUME	ODD_RST	USB_EN/SOF_EN
0x(FF)FF_9A98	ADDR	LS_EN	ADDR [6:0]						
0x(FF)FF_9A9C	BDT_PAGE_01	BDT_BA15	BDT_BA14	BDT_BA13	BDT_BA12	BDT_BA11	BDT_BA10	BDT_BA9	NOT USED
0x(FF)FF_9AA0	FRM_NUML	FRM7	FRM8	FRM5	FRM4	FRM3	FRM2	FRM1	FRM0
0x(FF)FF_9AA4	FRM_NUMH	0	0	0	0	0	FRM10	FRM9	FRM8
0x(FF)FF_9AA8	TOKEN	TOKEN_PID				TOKEN_ENDPT			
0x(FF)FF_9AAC	SOF_THLD	CNT7	CNT6	CNT5	CNT4	CNT3	CNT2	CNT1	CNT0
0x(FF)FF_9AB0	BDT_PAGE_02	BDT_BA23	BDT_BA22	BDT_BA21	BDT_BA20	BDT_BA19	BDT_BA18	BDT_BA17	BDT_BA16
0x(FF)FF_9AB4	BDT_PAGE_03	BDT_BA31	BDT_BA30	BDT_BA29	BDT_BA28	BDT_BA27	BDT_BA26	BDT_BA25	BDT_BA24
0x(FF)FF_9AB8-0x(FF)FF_9ABF	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_9AC0	ENDPT0	HOST_WO_HUB	RETRY_DIS	0	EP_CTL_DIS	EP_RX_EN	EP_TX_EN	EP_STALL	EP_HSHK
0x(FF)FF_9AC4	ENDPT1	HOST_WO_HUB	RETRY_DIS	0	EP_CTL_DIS	EP_RX_EN	EP_TX_EN	EP_STALL	EP_HSHK
0x(FF)FF_9AC8	ENDPT2	HOST_WO_HUB	RETRY_DIS	0	EP_CTL_DIS	EP_RX_EN	EP_TX_EN	EP_STALL	EP_HSHK
0x(FF)FF_9ACC	ENDPT3	HOST_WO_HUB	RETRY_DIS	0	EP_CTL_DIS	EP_RX_EN	EP_TX_EN	EP_STALL	EP_HSHK
0x(FF)FF_9AD0	ENDPT4	HOST_WO_HUB	RETRY_DIS	0	EP_CTL_DIS	EP_RX_EN	EP_TX_EN	EP_STALL	EP_HSHK

Table 4-4. USB Register Summary (continued)

Address	Name	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_9AD4	ENDPT5	HOST_WO_HUB	RETRY_DIS	0	EP_CTL_DIS	EP_RX_EN	EP_TX_EN	EP_STALL	EP_HSHK
0x(FF)FF_9AD8	ENDPT6	HOST_WO_HUB	RETRY_DIS	0	EP_CTL_DIS	EP_RX_EN	EP_TX_EN	EP_STALL	EP_HSHK
0x(FF)FF_9ADC	ENDPT7	HOST_WO_HUB	RETRY_DIS	0	EP_CTL_DIS	EP_RX_EN	EP_TX_EN	EP_STALL	EP_HSHK
0x(FF)FF_9AE0	ENDPT8	HOST_WO_HUB	RETRY_DIS	0	EP_CTL_DIS	EP_RX_EN	EP_TX_EN	EP_STALL	EP_HSHK
0x(FF)FF_9AE4	ENDPT9	HOST_WO_HUB	RETRY_DIS	0	EP_CTL_DIS	EP_RX_EN	EP_TX_EN	EP_STALL	EP_HSHK
0x(FF)FF_9AE8	ENDPT10	HOST_WO_HUB	RETRY_DIS	0	EP_CTL_DIS	EP_RX_EN	EP_TX_EN	EP_STALL	EP_HSHK
0x(FF)FF_9AEC	ENDPT11	HOST_WO_HUB	RETRY_DIS	0	EP_CTL_DIS	EP_RX_EN	EP_TX_EN	EP_STALL	EP_HSHK
0x(FF)FF_9AF0	ENDPT12	HOST_WO_HUB	RETRY_DIS	0	EP_CTL_DIS	EP_RX_EN	EP_TX_EN	EP_STALL	EP_HSHK
0x(FF)FF_9AF4	ENDPT13	HOST_WO_HUB	RETRY_DIS	0	EP_CTL_DIS	EP_RX_EN	EP_TX_EN	EP_STALL	EP_HSHK
0x(FF)FF_9AF8	ENDPT14	HOST_WO_HUB	RETRY_DIS	0	EP_CTL_DIS	EP_RX_EN	EP_TX_EN	EP_STALL	EP_HSHK
0x(FF)FF_9AFC	ENDPT15	HOST_WO_HUB	RETRY_DIS	0	EP_CTL_DIS	EP_RX_EN	EP_TX_EN	EP_STALL	EP_HSHK
0x(FF)FF_9B00	USB_CTRL	SUSP	PDE	—	—	—	—	—	CLK_SRC
0x(FF)FF_9B04	USB_OTG_OBSERVE	DP_PU	DP_PD	0	DM_PD	—	—	—	1
0x(FF)FF_9B08	USB_OTG_CONTROL	—	—	—	DPPULL_UP_NONOTG	ID	VBUS_VLD	SESS_VLD	SESS_END
0x(FF)FF_9B0C	USBTRC0	USB_RESET	USBPU	USBRESMEN	—	—	USBVREN	—	USB_RESUME_INT
0x(FF)FF_9B10	OTGPIN	—	USBID	DM_DOWN	DP_DOWN	PULLUP	VBUS_VLD	SESS_END	SESS_VLD
0x(FF)FF_9B0F-0x(FF)FF_9BFF	Reserved	—	—	—	—	—	—	—	—

#### 4.2.1 Flash Module Reserved Memory Locations

Several reserved flash memory locations, shown in Table 4-5, 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 4-5. 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 4-6. 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 <b>FTRIM</b>	0	0	0	0	0	0	0	FTRIM
0x(00)00_03FF	Storage of <b>MCGTRM</b>	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	<b>NVPROT</b>	FPS							FPOOPEN
0x(00)00_040E	Reserved	—	—	—	—	—	—	—	—
0x(00)00_040F	<b>NVOPT</b>	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.

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.

**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.

### 4.2.2 ColdFire Rapid GPIO Memory Map

The rapid GPIO module is mapped into a 16-byte area starting at location 0xC0\_0000. Its memory map is shown below in [Table 4-7](#).

**Table 4-7. V1 ColdFire Rapid GPIO Memory Map**

Address	Register Name	Bit 7	6	5	4	3	2	1	Bit 0
0x(00)C0_0000	GPIO_DIR	15	14	13	12	11	10	9	8
0x(00)C0_0001	GPIO_DIR	7	6	5	4	3	2	1	0
0x(00)C0_0002	GPIO_DATA	15	14	13	12	11	10	9	8
0x(00)C0_0003	GPIO_DATA	7	6	5	4	3	2	1	0
0x(00)C0_0004	GPIO_ENB	15	14	13	12	11	10	9	8
0x(00)C0_0005	GPIO_ENB	7	6	5	4	3	2	1	0
0x(00)C0_0006	GPIO_CLR	15	14	13	12	11	10	9	8
0x(00)C0_0007	GPIO_CLR	7	6	5	4	3	2	1	0
0x(00)C0_0008	Reserved	—	—	—	—	—	—	—	—
0x(00)C0_0009	Reserved	—	—	—	—	—	—	—	—
0x(00)C0_000A	GPIO_SET	15	14	13	12	11	10	9	8
0x(00)C0_000B	GPIO_SET	7	6	5	4	3	2	1	0
0x(00)C0_000C	Reserved	—	—	—	—	—	—	—	—
0x(00)C0_000D	Reserved	—	—	—	—	—	—	—	—
0x(00)C0_000E	GPIO_TOG	15	14	13	12	11	10	9	8
0x(00)C0_000F	GPIO_TOG	7	6	5	4	3	2	1	0

### 4.2.3 ColdFire Interrupt Controller Memory Map

The V1 ColdFire interrupt controller (CF1\_INTC) register map is sparsely-populated, but retains compatibility with earlier ColdFire interrupt controller definitions. The CF1\_INTC occupies the upper 64 bytes of the 4 GB address space and all memory locations are accessed as 8-bit (byte) operands.

**Table 4-8. V1 ColdFire Interrupt Controller Memory Map**

Address	Register Name	msb	Bit Number						lsb
0x(FF)FF_FFC0– 0x(FF)FF_FFCF	<b>Reserved</b>	—	—	—	—	—	—	—	—
0x(FF)FF_FFD0	<b>INTC_FRC</b>	0	LVL1	LVL2	LVL3	LVL4	LVL5	LVL6	LVL7
0x(FF)FF_FFD1– 0x(FF)FF_FFD7	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_FFD8	<b>INTC_PL6P7</b>	0	0	REQN					
0x(FF)FF_FFD9	<b>INTC_PL6P6</b>	0	0	REQN					
0x(FF)FF_FFDA	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_FFDB	<b>INTC_WCR</b>	ENB	0	0	0	0	MASK		
0x(FF)FF_FFDC– 0x(FF)FF_FFDD	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_FFDE	<b>INT_SFRC</b>	0	0	SET					
0x(FF)FF_FFDF	<b>INT_CFR</b>	0	0	CLR					
0x(FF)FF_FFE0	<b>INTC_SWIACK</b>	0	VECN						—
0x(FF)FF_FFE1– 0x(FF)FF_FFE3	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_FFE4	<b>INTC_LVL1IACK</b>	0	VECN						—
0x(FF)FF_FFE5– 0x(FF)FF_FFE7	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_FFE8	<b>INTC_LVL2IACK</b>	0	VECN						—
0x(FF)FF_FFE9– 0x(FF)FF_Ffeb	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_FFEC	<b>INTC_LVL3IACK</b>	0	VECN						—
0x(FF)FF_FFED– 0x(FF)FF_FFEF	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_FFF0	<b>INTC_LVL4IACK</b>	0	VECN						—
0x(FF)FF_FFF1– 0x(FF)FF_FFF3	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_FFF4	<b>INTC_LVL5IACK</b>	0	VECN						—
0x(FF)FF_FFF5– 0x(FF)FF_FFF7	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_FFF8	<b>INTC_LVL6IACK</b>	0	VECN						—
0x(FF)FF_FFF9– 0x(FF)FF_FFFB	Reserved	—	—	—	—	—	—	—	—
0x(FF)FF_FFFC	<b>INTC_LVL7IACK</b>	0	VECN						—
0x(FF)FF_FFFD– 0x(FF)FFFF	Reserved	—	—	—	—	—	—	—	—

## 4.3 RAM

An MCF51JM128 series microcontroller includes up to 16 Kbytes 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 ( $V_{RAM}$ ).

## 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 4-9](#). Failure to adhere to these guidelines may result in a partially programmed flash array.

**Table 4-9. 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

**Table 4-9. Low-Voltage Programming Sequence Example (continued)**

Addresses	Desired Value	Values Programmed
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:

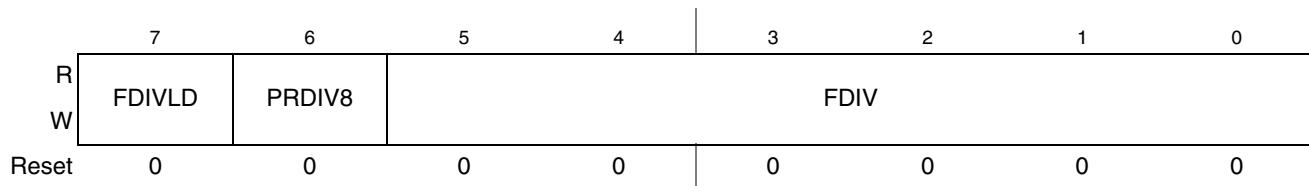
- Flash size
  - MCF51JM128: 131,072 bytes (128 sectors of 1024 bytes each)
  - MCF51JM64: 65,536 bytes (64 sectors of 1024 bytes each)
  - MCF51JM32: 32,5768 bytes (32 sectors of 1024 bytes each)
- Automated program and erase algorithm
- Fast program and sector erase operation
- Burst program command 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-Kbyte 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 module contains a set of 16 control and status registers located between 0x(00)00\_0000 and 0x(00)00\_000F. Detailed descriptions of each register bit are provided in the following sections.

### 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 4-3. Flash Clock Divider Register (FCDIV)**

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

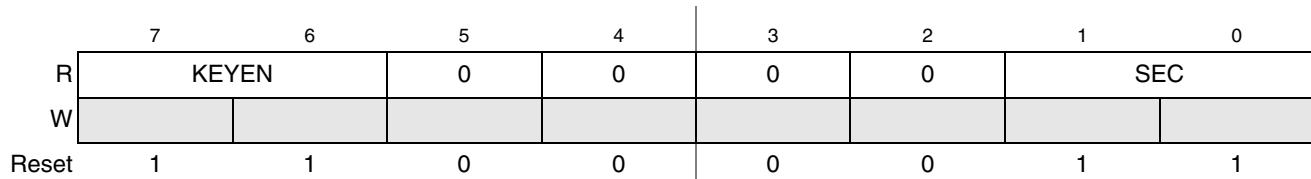
Field	Description
7 FDIVLD	<b>Clock Divider Load Control.</b> When writing to the FCDIV register for the first time after a reset, the value of the FDIVLD bit written controls the future ability to write to the FCDIV register: 0 Writing a 0 to FDIVLD locks the FCDIV register contents; all future writes to FCDIV are ignored. 1 Writing a 1 to FDIVLD 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 down 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. All bits in the FOPT register are readable but are not writable.

The FOPT register is loaded from the flash configuration field during the reset sequence, indicated by F in [Figure 4-4](#).

The security feature in the flash module is described in [Section 4.6, “Security”](#).

**Figure 4-4. Flash Options Register (FOPT)****Table 4-11. 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, should 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	0
W									
Reset	0	0		0	0	0	0	0	0

Figure 4-5. Flash Configuration Register (FCNFG)

Table 4-12. FCNFG Field Descriptions

Field	Description
7–6	Reserved, should 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, should 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 as long as 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 [Table 4-6](#). 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				FPS					FPOOPEN
W									
Reset	1	1	1	1		1	1	1	1

Figure 4-6. Flash Protection Register (FPROT)

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

<b>Field</b>	<b>Description</b>
7–1 FPS	<b>Flash Protection Size.</b> With FPOOPEN set, the FPS bits determine the size of the protected flash address range as shown in <a href="#">Table 4-14</a> .
0 FPOOPEN	Flash Protection Open 0 Flash array fully protected. 1 Flash array protected address range determined by FPS bits.

**Table 4-14. Flash Protection Address Range**

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

**Table 4-14. Flash Protection Address Range (continued)**

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

#### 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 readable and not writable. 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
Reset	1	1	0	0	0	0	0	0	0

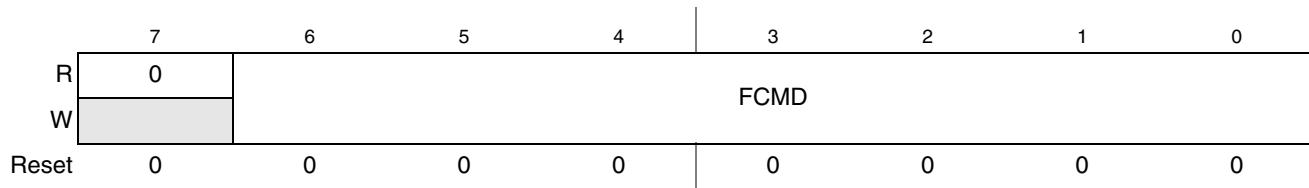
**Figure 4-7. Flash Status Register (FSTAT)**

**Table 4-15. 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 be 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 upon 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 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, should 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, should 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 4-8. Flash Command Register (FCMD)**

**Table 4-16. FCMD Field Descriptions**

Field	Description
7	Reserved, should 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 Function Description

### 4.5.1 Flash Command Operations

Flash command operations execute program, erase, and erase verify algorithms described in this section. The program and erase algorithms are controlled by the flash memory controller whose time base, FCLK, is derived from the bus clock via a programmable divider.

The next sections describe:

1. How to write the FCDIV register to set FCLK
2. Command write sequences to program, erase, and erase verify operations on the flash memory
3. Valid flash commands
4. Effects resulting from illegal flash command write sequences or aborting flash operations

#### 4.5.1.1 Writing the FCDIV Register

Prior to issuing any flash command after a reset, write the FCDIV register to divide the bus clock within 150–200 kHz. The FCDIV[PRDIV8, FDIV] bits must be set as described in [Figure 4-9](#).

For example, if the bus clock frequency is 25 MHz, FCDIV[FDIV] should be set to 0x0F (001111) and the FCDIV[PRDIV8] bit set to 1. The resulting FCLK frequency is then 195 kHz. In this case, the flash program and erase algorithm timings are increased over the optimum target by:

$$(200 - 195) \div 200 = 3\%$$

*Eqn. 4-1*

#### CAUTION

Program and erase command execution time increase proportionally with the period of FCLK. Programming or erasing the flash memory with FCLK less than 150 kHz should be avoided. Setting FCDIV to a value such that FCLK is less than 150 kHz can destroy the flash memory due to overstress. Setting FCDIV to a value where FCLK is greater than 200 kHz can result in incomplete programming or erasure of the flash memory cells.

If the FCDIV register is written, the FDIVLD bit is automatically set. If the FDIVLD bit is 0, the FCDIV register has not been written since the last reset. If the FCDIV register has not been written to, the flash command loaded during a command write sequence does not execute and FSTAT[FACCERR] is set.

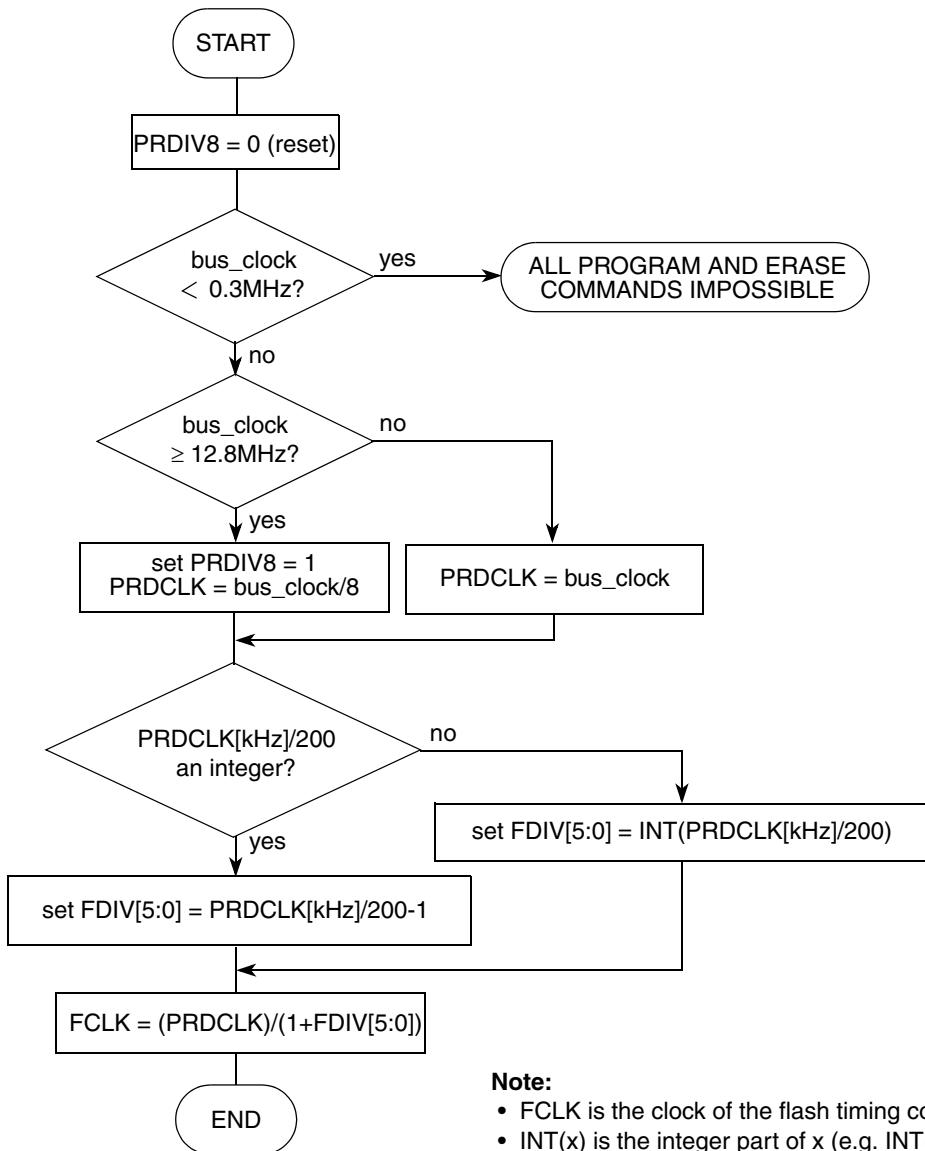


Figure 4-9. Determination Procedure for PRDIV8 and FDIV Bits

#### 4.5.1.2 Command Write Sequence

The flash command controller supervises the command write sequence to execute program, erase, and erase verify algorithms.

Before starting a command write sequence, the FACCERR and FPVIOL flags in the FSTAT register must be clear and the FCBEF flag must be set (see [Section 4.4.2.5](#)).

A command write sequence consists of three steps that must be strictly adhered to with writes to the flash module not permitted between the steps. However, flash register and array reads are allowed during a command write sequence. The basic command write sequence is as follows:

1. Write to a valid address in the flash array memory.
2. Write a valid command to the FCMD register.
3. Clear the FCBEF flag in the FSTAT register by writing a 1 to FCBEF to launch the command.

After a command is launched, the completion of the command operation is indicated by the setting of FSTAT[FCCF]. The FCCF flag sets upon completion of all active and buffered burst program commands.

## 4.5.2 Flash Commands

**Table 4-17** summarizes the valid flash commands along with the effects of the commands on the flash block.

**Table 4-17. Flash Command Description**

FCMD	NVM Command	Function on Flash Memory
0x05	Erase Verify	Verify all memory bytes in the flash array memory are erased. If the flash array memory is erased, FSTAT[FBLANK] sets upon command completion.
0x20	Program	Program an address in the flash array.
0x25	Burst Program	Program an address in the flash array with the internal address incrementing after the program operation.
0x40	Sector Erase	Erase all memory bytes in a sector of the flash array.
0x41	Mass Erase	Erase all memory bytes in the flash array. A mass erase of the full flash array is only possible when no protection is enabled prior to launching the command.

### 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.

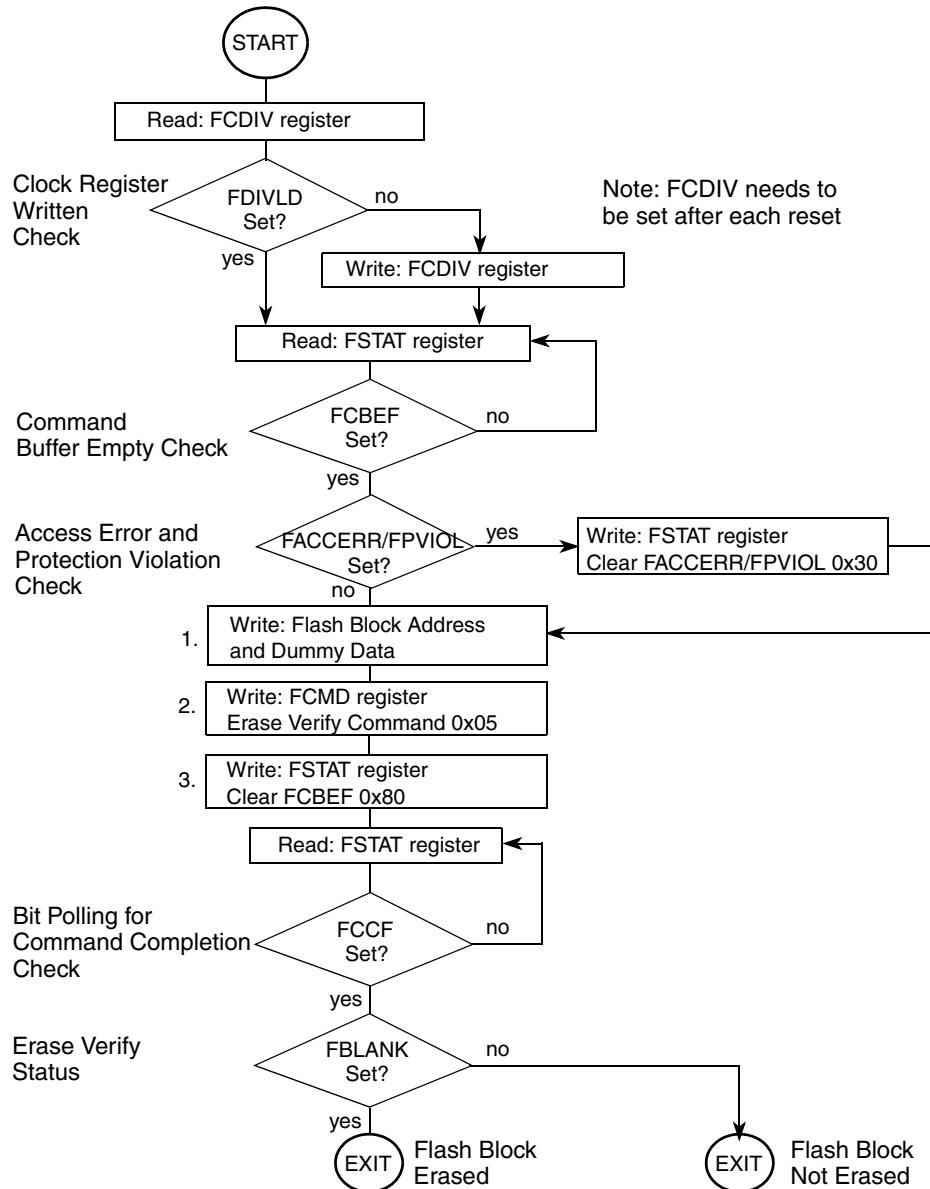
### 4.5.2.1 Erase Verify Command

The erase verify operation verifies that the entire flash array memory is erased.

An example flow to execute the erase verify operation is shown in [Figure 4-10](#). The erase verify command write sequence is as follows:

1. Write to a flash block address to start the command write sequence for the erase verify command. The address and data written are ignored.
2. Write the erase verify command, 0x05, to the FCMD register.
3. Clear the FCBEF flag in the FSTAT register by writing a 1 to FCBEF to launch the erase verify command.

After launching the erase verify command, FSTAT[FCCF] sets after the operation has completed. The number of bus cycles required to execute the erase verify operation is equal to the number of addresses in the flash array memory plus several bus cycles as measured from the time the FCBEF flag is cleared until the FCCF flag is set. Upon completion of the erase verify operation, FSTAT[FBLANK] is set if all addresses in the flash array memory are verified to be erased. If any address in the flash array memory is not erased, the erase verify operation terminates and FSTAT[FBLANK] remains cleared.



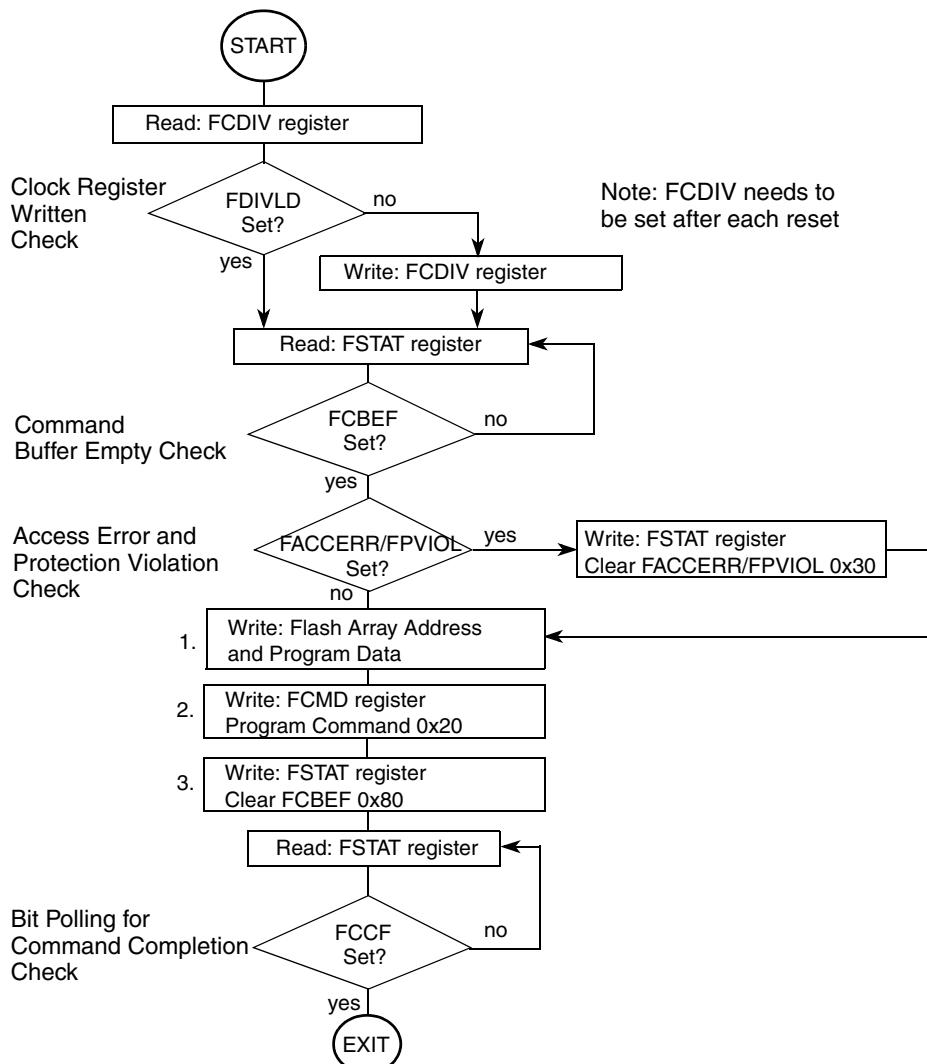
**Figure 4-10. Example Erase Verify Command Flow**

### 4.5.2.2 Program Command

The program operation programs a previously erased address in the flash memory using an embedded algorithm. An example flow to execute the program operation is shown in [Figure 4-11](#). The program command write sequence is as follows:

1. Write to a flash block address to start the command write sequence for the program command. The data written is programmed to the address written.
2. Write the program command, 0x20, to the FCMD register.
3. Clear the FCBEF flag in the FSTAT register by writing a 1 to FCBEF to launch the program command.

If an address to be programmed is in a protected area of the flash block, FSTAT[FPVIOL] sets and the program command does not launch. After the program command has successfully launched and the program operation has completed, FSTAT[FCCF] is set.



**Figure 4-11. Example Program Command Flow**

### 4.5.2.3 Burst Program Command

The burst program operation programs previously erased data in the flash memory using an embedded algorithm.

While burst programming, two internal data registers operate as a buffer and a register (2-stage FIFO) so that a second burst programming command along with the necessary data can be stored to the buffers while the first burst programming command remains in progress. This pipelined operation allows a time optimization when programming more than one consecutive address on a specific row in the flash array as the high voltage generation can be kept active in between two programming commands.

An example flow to execute the burst program operation is shown in [Figure 4-12](#). The burst program command write sequence is as follows:

1. Write to a flash block address to start the command write sequence for the burst program command. The data written is programmed to the address written.
2. Write the program burst command, 0x25, to the FCMD register.
3. Clear the FCBEF flag in the FSTAT register by writing a 1 to FCBEF to launch the program burst command.
4. After the FCBEF flag in the FSTAT register returns to a 1, repeat steps 1 through 3. The address written is ignored but is incremented internally.

The burst program procedure can be used to program the entire flash memory even while crossing row boundaries within the flash array. If data to be burst programmed falls within a protected area of the flash array, FSTAT[FPVIOL] is set and the burst program command does not launch. After the burst program command has successfully launched and the burst program operation has completed, FSTAT[FCCF] is set unless a new burst program command write sequence has been buffered. By executing a new burst program command write sequence on sequential addresses after the FCBEF flag in the FSTAT register has been set, a greater than 50% faster programming time for the entire flash array can be effectively achieved when compared to using the basic program command.

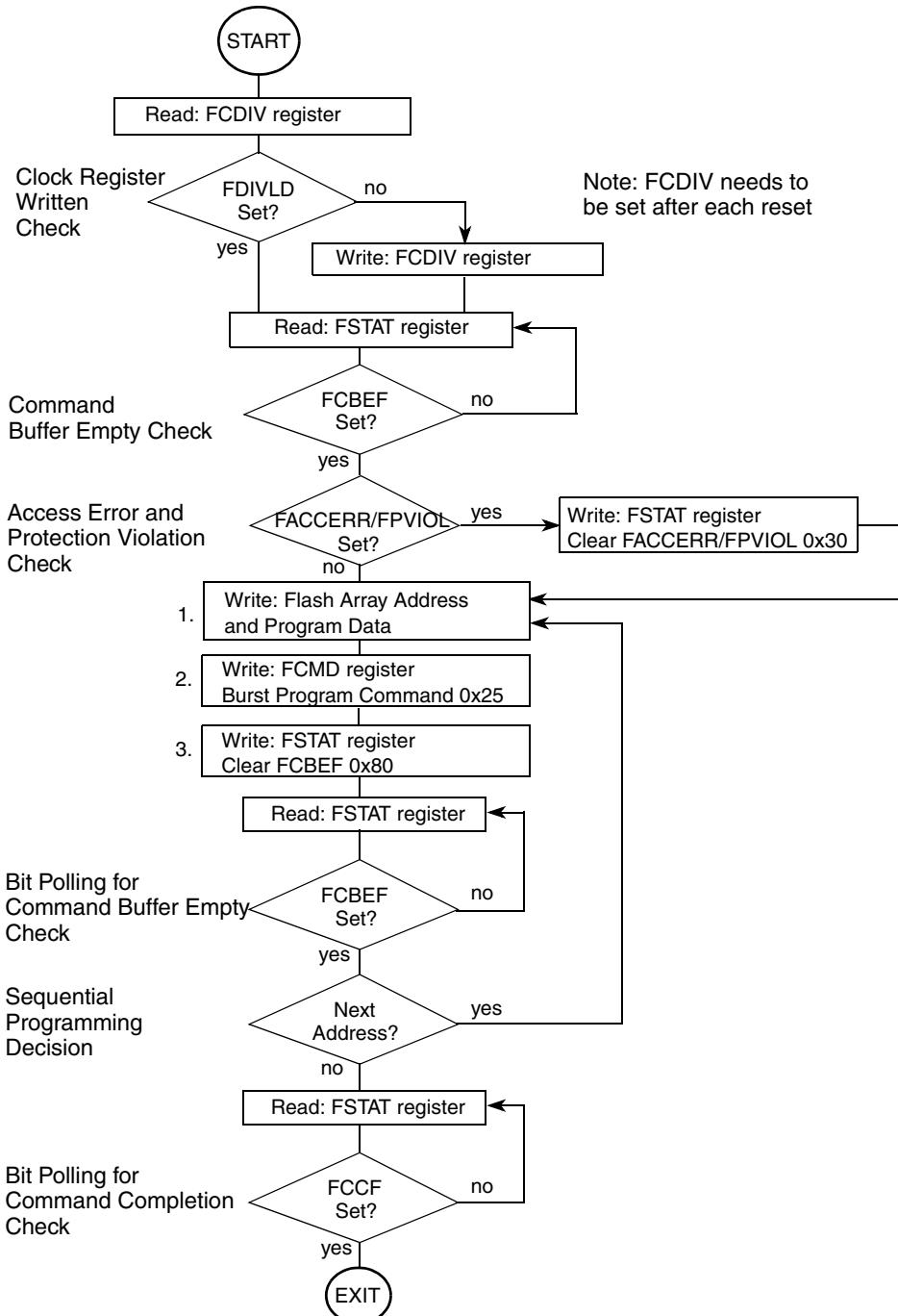


Figure 4-12. Example Burst Program Command Flow

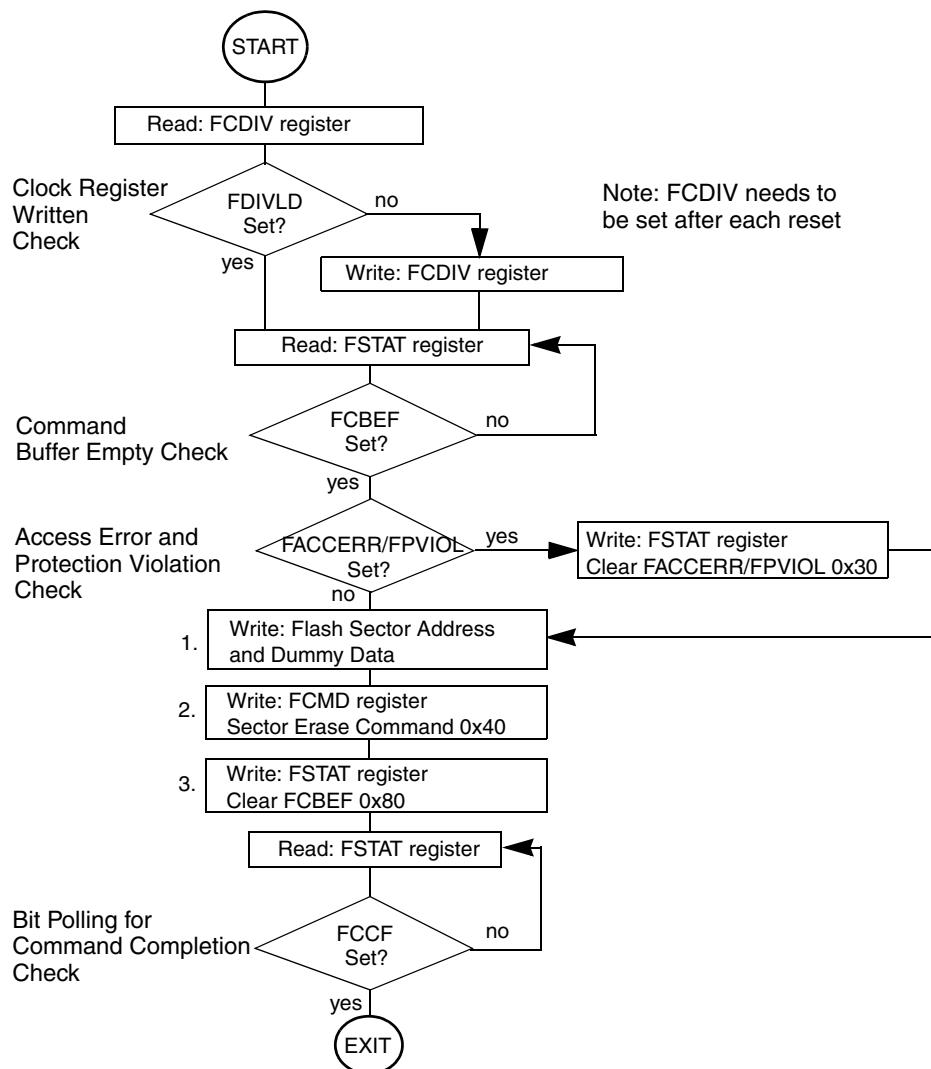
#### 4.5.2.4 Sector Erase Command

The sector erase operation erases all addresses in a 2Kbyte sector of flash memory using an embedded algorithm.

An example flow to execute the sector erase operation is shown in [Figure 4-13](#). The sector erase command write sequence is as follows:

1. Write to a flash block address to start the command write sequence for the sector erase command. The flash address written determines the sector to be erased while the data written is ignored.
2. Write the sector erase command, 0x40, to the FCMD register.
3. Clear the FCBEF flag in the FSTAT register by writing a 1 to FCBEF to launch the sector erase command.

If a flash sector to be erased is in a protected area of the flash block, FSTAT[FPVIOL] is set and the sector erase command does not launch. After the sector erase command has successfully launched and the sector erase operation has completed, FSTAT[FCCF] is set.



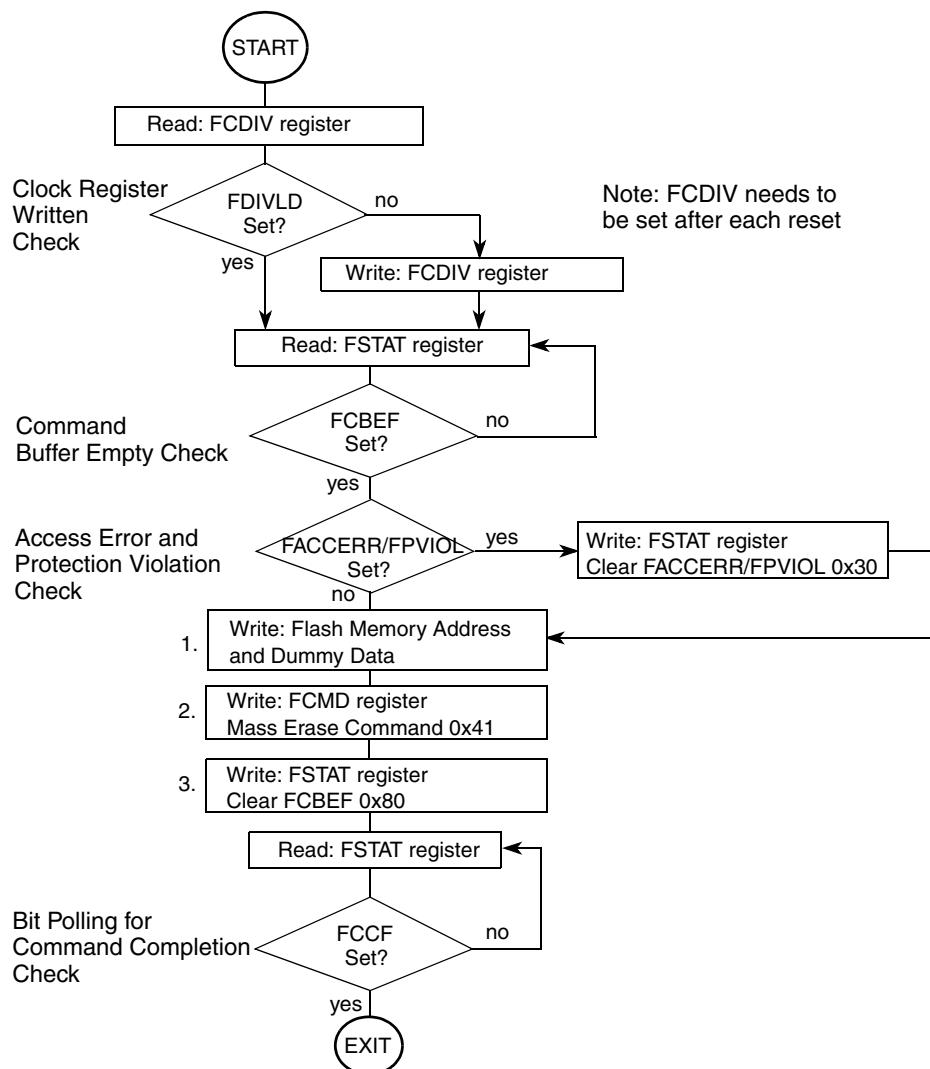
**Figure 4-13. Example Sector Erase Command Flow**

### 4.5.2.5 Mass Erase Command

The mass erase operation erases the entire flash array memory using an embedded algorithm. An example flow to execute the mass erase operation is shown in [Figure 4-14](#). The mass erase command write sequence is as follows:

1. Write to a flash block address to start the command write sequence for the mass erase command. The address and data written is ignored.
2. Write the mass erase command, 0x41, to the FCMD register.
3. Clear the FCBEF flag in the FSTAT register by writing a 1 to FCBEF to launch the mass erase command.

If the flash array memory to be mass erased contains any protected area, FSTAT[FPVIOL] is set and the mass erase command does not launch. After the mass erase command has successfully launched and the mass erase operation has completed, FSTAT[FCCF] is set.



**Figure 4-14. Example Mass Erase Command Flow**

## NOTE

The BDM can also perform a mass erase and verify command. See [Chapter 23, “Version 1 ColdFire Debug \(CF1\\_DEBUG\),”](#) for details.

### 4.5.3 Illegal Flash Operations

#### 4.5.3.1 Flash Access Violations

The FACCERR flag is set during the command write sequence if any of the following illegal steps are performed, causing the command write sequence to immediately abort:

1. Writing to a flash address before initializing the FCMD register.
2. This scenario can cause an illegal address reset as this operation is not supported per the memory map. To cause an exception instead of a system reset, set the ADR bit of core register CPUCR. After setting the ADR bit, performing a write operation sets the FACCERR flag.
3. Writing to any flash register other than FCMD after writing to a flash address.
4. Writing to a second flash address in the same command write sequence.
5. Writing an invalid command to the FCMD register.
6. Writing a command other than burst program while FCBEF is set and FCCF is clear.
7. When security is enabled, writing a command other than erase verify or mass erase to the FCMD register when the write originates from a non-secure memory location or from the background debug mode.
8. Writing to a flash address after writing to the FCMD register.
9. Writing to any flash register other than FSTAT (to clear FCBEF) after writing to the FCMD register.
10. Writing a 0 to the FCBEF flag in the FSTAT register to abort a command write sequence.

The FACCERR flag is also set if the MCU enters stop mode while any command is active (FCCF=0). The operation is aborted immediately and, if burst programming, any pending burst program command is purged (see [Section 4.5.4.2, “Stop Modes”](#)).

The FACCERR flag does not set if any flash register is read during a valid command write sequence.

If the flash memory is read during execution of an algorithm (FCCF = 0), the read operation returns invalid data and the FACCERR flag is not set.

If the FACCERR flag is set in the FSTAT register, clear the FACCERR flag before starting another command write sequence (see [Section 4.4.2.5, “Flash Status Register \(FSTAT\)”](#)).

#### 4.5.3.2 Flash Protection Violations

The FPVIOL flag is set after the command is written to the FCMD register if any of the following illegal operations are attempted:

1. Writing the program command if the address written in the command write sequence was in a protected area of the flash array.

2. Writing the sector erase command if the address written in the command write sequence was in a protected area of the flash array.
3. Writing the mass erase command while any flash protection is enabled.

As a result of any of the above, the command write sequence immediately aborts. If FSTAT[FPVIOL] is set, clear the FPVIOL flag before starting another command write sequence (see [Section 4.4.2.5, “Flash Status Register \(FSTAT\)”](#)).

## 4.5.4 Operating Modes

### 4.5.4.1 Wait Mode

If a command is active (FCCF = 0) when the MCU enters wait mode, the active command and any buffered command is completed.

### 4.5.4.2 Stop Modes

If a command is active (FCCF = 0) when the MCU enters any stop mode, the operation is aborted. If the operation is program or erase, the flash array data being programmed or erased may be corrupted and the FCCF and FACCERR flags are set. If active, the high voltage circuitry to the flash array is immediately switched off when entering stop mode. Upon exit from stop mode, the FCBEF flag is set and any buffered command is not launched. The FACCERR flag must be cleared before starting a command write sequence (see [Section 4.5.1.2, “Command Write Sequence”](#)).

#### NOTE

As active commands are immediately aborted when the MCU enters stop mode, do not use the STOP instruction during program or erase operations.

Active commands continue when the MCU enters wait mode. Use of the STOP instruction when SOPT1[WAITE] is set is acceptable.

### 4.5.4.3 Background Debug Mode

In background debug mode, the FPROT register is writable without restrictions. If the MCU is unsecured, all flash commands listed in [Table 4-12](#) can be executed. If the MCU is secured, only a compound mass erase and erase verify command can be executed. See [Chapter 23, “Version 1 ColdFire Debug \(CF1\\_DEBUG\),”](#) for details.

## 4.5.5 Security

The flash module provides the necessary security information to the MCU. During each reset sequence, the flash module determines the security state of the MCU as defined in [Section 4.2.1, “Flash Module Reserved Memory Locations”](#).

The contents of the flash security byte in the flash configuration field (see [Section 4.4.2.3](#)) must be changed directly by programming the flash security byte location when the MCU is unsecured and the

sector containing the flash security byte is unprotected. If the flash security byte is left in a secured state, any reset causes the MCU to initialize into a secure operating mode.

#### 4.5.5.1 Unsecuring the MCU using Backdoor Key Access

The MCU may be unsecured by using the backdoor key access feature that requires knowledge of the contents of the backdoor keys (see [Section 4.2.1](#)). If the KEYEN[1:0] bits are in the enabled state (see [Section 4.4.2.2](#)) and the KEYACC bit is set, a write to a backdoor key address in the flash memory triggers a comparison between the written data and the backdoor key data stored in the flash memory. If all backdoor keys are written to the correct addresses in the correct order and the data matches the backdoor keys stored in the flash memory, the MCU is unsecured. The data must be written to the backdoor keys sequentially. Values `0` and `1` are not permitted as backdoor keys. While the KEYACC bit is set, the flash memory reads return valid data.

The user code stored in the flash memory must have a method of receiving the backdoor keys from an external stimulus. This external stimulus would typically be through one of the on-chip serial ports.

If the KEYEN[1:0] bits are in the enabled state (see [Section 4.4.2.2](#)), the MCU can be unsecured by the backdoor key access sequence described below:

1. Set FCNFG[KEYACC].
2. Execute three NOP instructions to provide time for the backdoor state machine starting address and number of keys required into the flash state machine
3. Clear the KEYACC bit. Depending on the user code used to write the backdoor keys, a wait cycle (NOP) may be required before clearing the KEYACC bit.
4. If all data written match the backdoor keys, the MCU is unsecured and the SEC[1:0] bits in the NVOPT register are forced to an unsecured state.

The backdoor key access sequence is monitored by an internal security state machine. An illegal operation during the backdoor key access sequence causes the security state machine to lock, leaving the MCU in the secured state. A reset of the MCU causes the security state machine to exit the lock state and allows a new backdoor key access sequence to be attempted. The following operations during the backdoor key access sequence lock the security state machine:

1. If any of the keys written does not match the backdoor keys programmed in the flash array.
2. If the keys are written in the wrong sequence.
3. If any of the keys written are all 0's or all 1's.
4. If the KEYACC bit does not remain set while the keys are written.
5. If any of the keys are written on successive MCU clock cycles.
6. Executing a STOP instruction before all keys have been written.

After the backdoor keys have been correctly matched, the MCU is unsecured. After the MCU is unsecured, the flash security byte can be programmed to the unsecure state, if desired.

In the unsecure state, you have full control of the contents of the backdoor keys by programming the associated addresses in the flash configuration field (see [Section 4.2.1, “Flash Module Reserved Memory Locations”](#)).

The security as defined in the flash security byte is not changed by using the backdoor key access sequence to unsecure. The stored backdoor keys are unaffected by the backdoor key access sequence. After the next reset of the MCU, the security state of the flash module is determined by the flash security byte. The backdoor key access sequence has no effect on the program and erase protections defined in the flash protection register (FPROT).

## 4.5.6 Resets

### 4.5.6.1 Flash Reset Sequence

On each reset, the flash module executes a reset sequence to hold CPU activity while reading the following resources from the flash block:

- MCU control parameters (see [Section 4.2.1](#))
- Flash protection byte (see [Section 4.2.1](#) and [Section 4.4.2.4](#))
- Flash nonvolatile byte (see [Section 4.2.1](#))
- Flash security byte (see [Section 4.2.1](#) and [Section 4.4.2.2](#))

### 4.5.6.2 Reset While Flash Command Active

If a reset occurs while any flash command is in progress, that command is immediately aborted. The state of the flash array address being programmed or the sector/block being erased is not guaranteed.

### 4.5.6.3 Program and Erase Times

Before any program or erase command can be accepted, the flash clock divider (FCDIV) must be written to set the internal clock for the flash module to a frequency ( $f_{FCLK}$ ) between 150 kHz and 200 kHz.

If the initial flash event is a mass erase and verify from BDM, then CSR3[31:24] must be loaded before the XCSR is written to initiate the erase and verify. The data in the XCSR and CSR3 is then loaded into the flash's FCDIV register. (See [Section 23.3.4, “Configuration/Status Register 3 \(CSR3\)”](#)). However, if the first flash event is executed by the processor directly, the flash's FCDIV register is written directly, and the XCSR and CSR3 are not involved.

One period of the resulting clock ( $1/f_{FCLK}$ ) is used by the command processor to time program and erase pulses. An integer number of these timing pulses are used by the command processor to complete a program or erase command.

Program and erase times are given in the *MCF51JM128 Data Sheet*, order number MCF51JM128DS.

## 4.6 Security

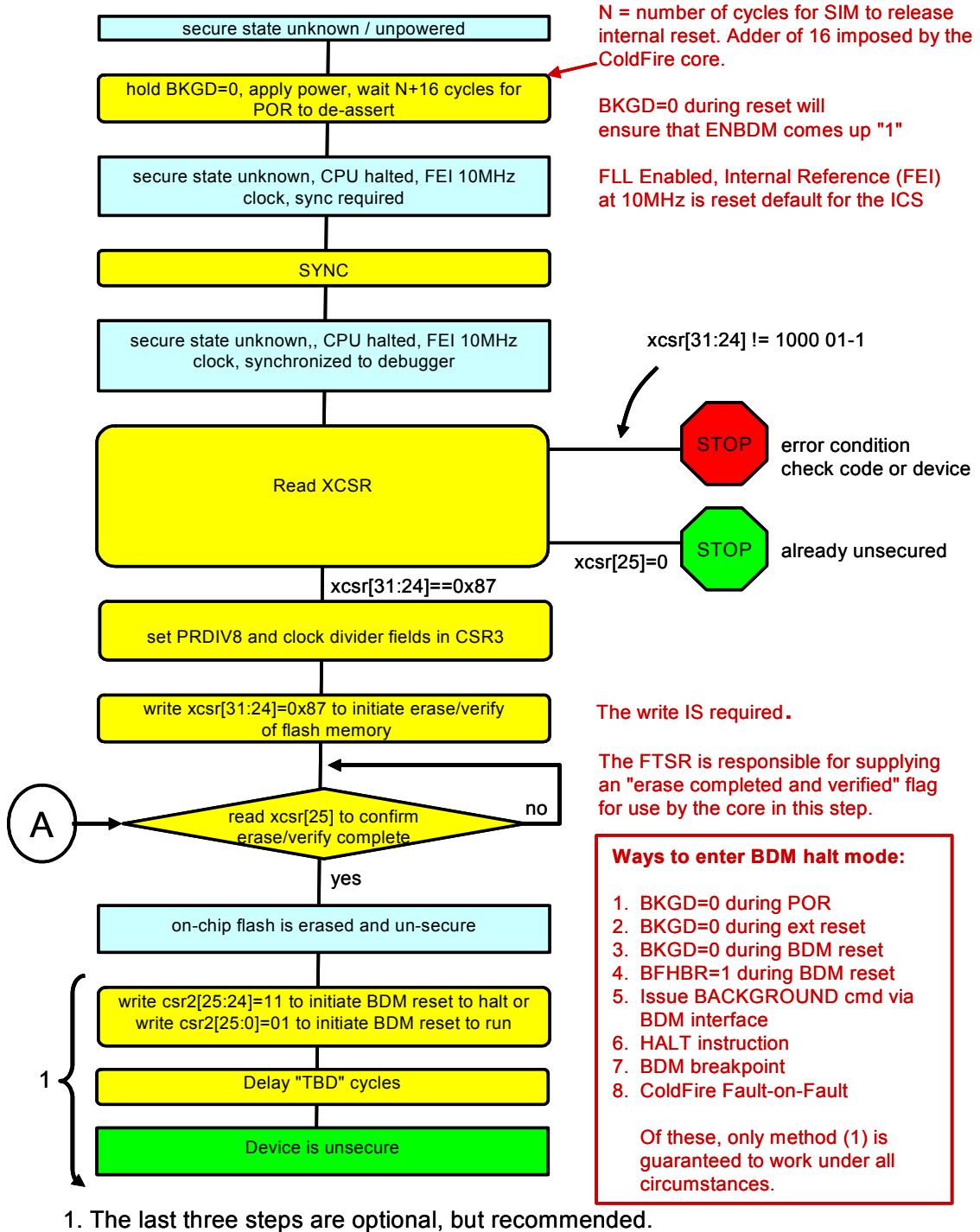
The MCF51JM128 series microcontroller includes circuitry to prevent unauthorized access to the contents of flash and RAM memory. When security is engaged, BDM access is restricted to the upper byte of the ColdFire CSR, XCSR, and CSR2 registers. RAM, flash memory, peripheral registers and most of the CPU register set are not available via BDM. Programs executing from internal memory have normal access to all microcontroller memory locations and resources.

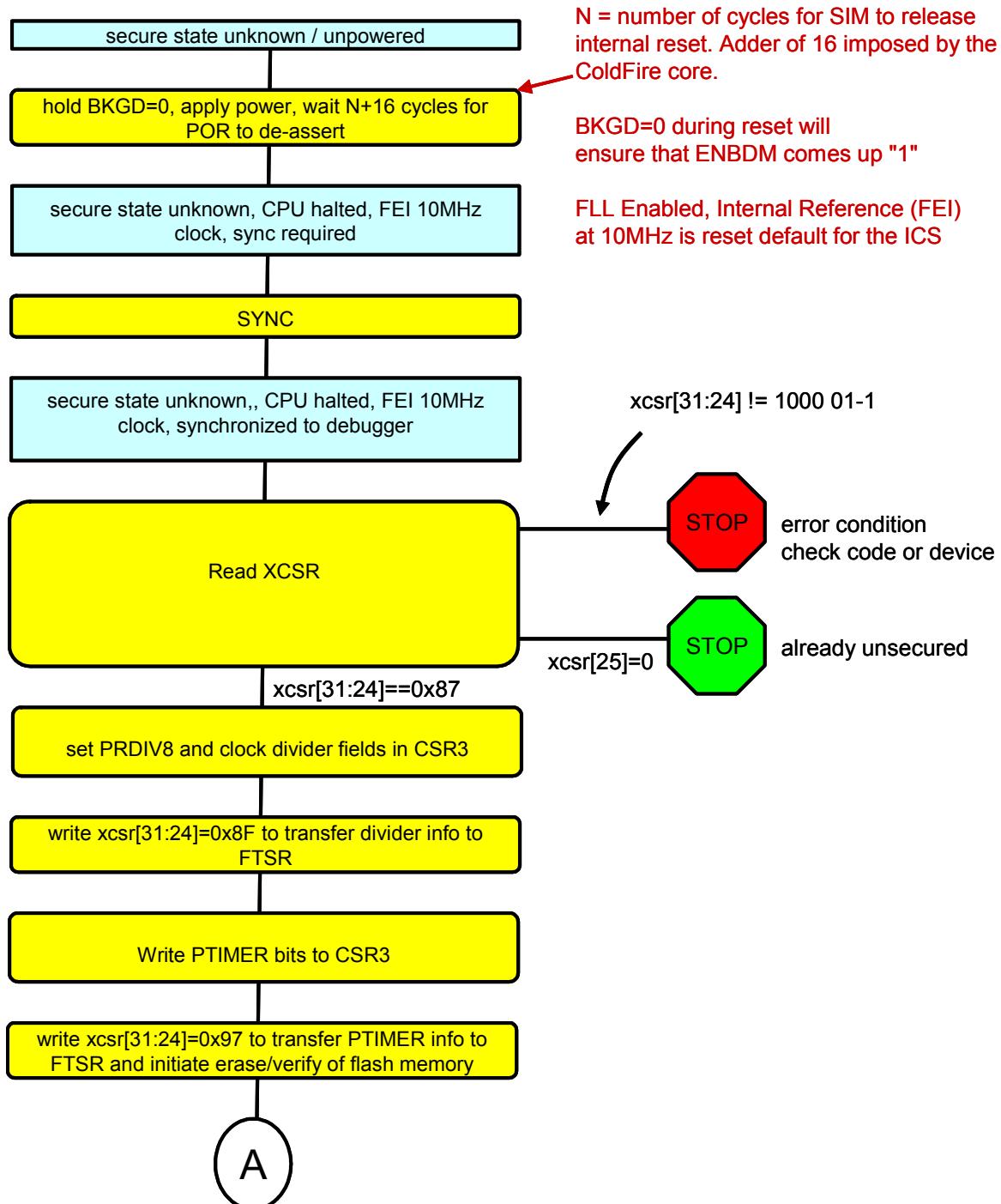
Security is engaged or disengaged based on the state of two nonvolatile register bits (SEC01, SEC00) in the FOPT register. During reset, the contents of the nonvolatile location, NVOPT, are copied from flash into the working FOPT register in high-page register space. A user engages security by programming the NVOPT location that can be done at the same time the flash memory is programmed. The 1:0 state engages security and the other three combinations disengage security. Security is implemented differently than on the pin-compatible MC9S08JM60 family of devices. This is a result of differences inherent in the S08 and ColdFire microcontroller architectures.

Upon exiting reset, the XCSR[25] bit in the ColdFire CPU is initialized to one if the device is secured, zero otherwise.

A user 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. The procedure for this is detailed in the section entitled “Unsecuring the MCU using Backdoor Key Access” in the flash module user guide.

Development tools unsecure devices via an alternate BDM-based methodology shown in [Figure 4-15](#). Because RESET and BKGD pins can be reprogrammed via software, a power-on-reset is required to be absolutely certain of obtaining control of the device via BDM, which is a required prerequisite for clearing security. Other methods (outlined in red in [Figure 4-15](#)) can also be used, but may not work under all circumstances.





**Figure 4-15. Procedure for Clearing Security on MCF51JM128 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 MCF51JM128 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 MCF51JM128 series microcontrollers have the following sources for reset:

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

Each of these sources, with the exception of 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.

After any reset, the COP watchdog is enabled (see [Section 5.7.3, “System Options 1 \(SOPT1\) Register,”](#) for additional information). 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 timeout period. Writes do not affect the data in the read-only SRS. As soon as the write sequence is done, the COP timeout period is restarted. If the program fails to do this during the time-out period, the microcontroller resets. Also, if any value other than 0x55 or 0xAA is written to SRS, the microcontroller is immediately reset.

The SOPT2[COPCLKS] field (see [Section 5.7.4, “System Options 2 \(SOPT2\) Register,”](#) 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 LPOCLK source. With each clock source, there are three associated time-outs controlled by SOPT1[COPT]. [Table 5-9](#) summarizes the control functions of the COPCLKS and COPT bits. The COP watchdog defaults to operation from the 1 kHz LPOCLK source and the longest time-out ( $2^{10}$  cycles).

When the bus clock source is selected, windowed COP operation is available by setting SOPT2[COPW]. In this mode, writes to the SRS register to clear the COP timer must occur in the last 25% of the selected timeout period. A premature write immediately resets the microcontroller. When the 1 kHz LPOCLK 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. Even if the application uses the reset default settings of the COPT, COPCLKS, and COPW bits, the user should write to the write-once SOPT1 and SOPT2 registers during reset initialization to lock in the settings. This prevents accidental changes if the application program gets lost.

The write to SRS that services (clears) the COP counter should not be placed in an interrupt service routine (ISR) because the ISR could continue to be executed periodically even if the main application program fails.

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 LPOCLK source is selected, the COP counter is re-initialized to zero upon entry to background debug mode or stop mode and begins from zero upon exit from background debug mode or stop mode.

### 5.3.2 Illegal Opcode Detect (IOP)

The default configuration of 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] = 0 && SOPT[WAITE] = 0) is treated as an illegal instruction.

The attempted execution of the HALT instruction with XCSR[ENBDM] = 0 is treated as an illegal instruction.

The processor generates a reset in response to any of these events if CPUCR[IRD] = 0. If this configuration bit is set, the processor generates the appropriate exception instead of forcing a reset.

### 5.3.3 Illegal Address Detect (ILAD)

The default configuration of 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] = 0. 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 and 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, after 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[IACK] = 0) or explicitly fetches an 8-bit vector from the INTC (if CPUCR[IACK] = 1).

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 that 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 manage reset, error conditions (access, address), arithmetic faults, system calls, etc. Vectors 64–255 are reserved for interrupt service routines. The MCF51JM128 series microcontrollers support 37 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] = 1.

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 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, therefore the IRQ pin (if enabled) can wake the microcontroller.

### 5.4.1.1 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, the user 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 that can be polled by software (IRQIE).

The IRQ pin, when enabled, defaults to use an internal pull device (IRQPDD = 0), configured as a pull-up or pull-down depending on the polarity chosen. If the user desires to use an external pull-up or pull-down, the IRQPDD can be set to turn off the internal device.

**NOTE**

This pin does not contain a clamp diode to V<sub>DD</sub> and should not be driven above V<sub>DD</sub>.

The voltage measured on the internally pulled up RESET pin is not pulled to V<sub>DD</sub>. The internal gates connected to this pin are pulled to V<sub>DD</sub>. The RESET pullup should not be used to pull up components external to the microcontroller.

### 5.4.1.2 Edge and Level Sensitivity

The IRQMOD control bit re-configures the detection logic so it detects edge events and pin levels. In the edge and level detection mode, the IRQF status flag becomes set 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) as long as the IRQ pin remains at the asserted level.

### 5.4.2 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 MCF51JM128 series microcontrollers. The table is sorted by priority of the sources, with higher-priority sources at the top of the table. The force\_lvl entries do not follow the address and vector number order of the surrounding vectors.

**Table 5-1. Reset and Interrupt Vectors**

Address	Level	Priority	Vector Number	Vector Description	Vector Name
0x000			0	Initial Supervisor Stack Pointer	Vreset
0x004			1	Initial PC	—
Same as standard ColdFire exception table, see <a href="#">Table 5-2</a> .					
0x100	7	Mid-point	64	IRQ	Virq
0x104	7	3	65	Low Voltage Detect	Vlvd
0x108	7	2	66	Loss Of Lock	Vlol
0x10C	6	5	67	SPI1	Vspi1
0x110	6	4	68	SPI2	Vspi2
0x114	6	3	69	USB_Status	Vusb
0x118	6	2	70	—	—
0x11C	5	7	71	TPM1 Channel 0	Vtpm1ch0
0x120	5	6	72	TPM1 Channel 1	Vtpm1ch1
0x124	5	5	73	TPM1 Channel 2	Vtpm1ch2

**Table 5-1. Reset and Interrupt Vectors (continued)**

<b>Address</b>	<b>Level</b>	<b>Priority</b>	<b>Vector Number</b>	<b>Vector Description</b>	<b>Vector Name</b>
0x128	5	4	74	TPM1 Channel 3	Vtpm1ch3
0x12C	5	3	75	TPM1 Channel 4	Vtpm1ch4
0x130	5	2	76	TPM1 Channel 5	Vtpm1ch5
0x134	5	1	77	TPM1 Overflow	Vtpm1ov6
0x138	4	7	78	TPM2 Channel 0	Vtpm2ch0
0x13C	4	6	79	TPM2 Channel 1	Vtpm2ch1
0x140	4	1	80	TPM2 Overflow	Vtpm2ovf
0x144	3	7	81	SCI1 Error	Vsci1err
0x148	3	6	82	SCI1 Receive	Vsci1rx
0x14C	3	5	83	SCI1 Transmit	Vsci1tx
0x150	3	4	84	SCI2 Error	Vsci2err
0x154	3	3	85	SCI2 Receive	Vsci2rx
0x158	3	2	86	SCI2 Transmit	Vsci2tx
0x15C	2	7	87	KBI Interrupt	Vkeyboard
0x160	2	6	88	ADC Conversion	Vadc
0x164	2	5	89	ACMP	Vacmpx
0x168	2	4	90	IIC1	Vii1cx
0x16C	2	3	91	RTC	Vrtc
0x170	2	2	92	IIC2	Vii2cx
0x174	2	1	93	CMT	Vcmrt
0x178	1	7	94	CAN Wakeup	Vcanwu
0x17C	1	6	95	CAN error	Vcanerr
0x180	1	5	96	CAN Receive	Vcanrx
0x184	1	2	97	CAN Transmit	Vcantx
0x188	1	1	98	RNGA Error	Vmga
0x1A0	7	0	104	Force_Lvl7	Force_Lvl7
0x1A4	6	0	105	Force_Lvl6	Force_Lvl6
0x1A8	5	0	106	Force_Lvl5	Force_Lvl5
0x1AC	4	0	107	Force_Lvl4	Force_Lvl4
0x1B0	3	0	108	Force_Lvl3	Force_Lvl3
0x1B4	2	0	109	Force_Lvl2	Force_Lvl2
0x1B8	1	0	110	Force_Lvl1	Force_Lvl1

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 5-2](#).

The CPU configuration register (CPUCR) within the supervisor programming model allows the users 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 5-2](#).

**Table 5-2. ColdFire Exception Vector Table<sup>1</sup>**

Vector	Exception	Reset Disabled via CPUCR	Reported using SRS
64-98	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

**Table 5-2. ColdFire Exception Vector Table<sup>1</sup> (continued)**

<b>Vector</b>	<b>Exception</b>	<b>Reset Disabled via CPUCR</b>	<b>Reported using SRS</b>
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].

### 5.4.3 Interrupt Request Level and Priority Assignments

The CF1\_INTC module implements a sparsely-populated 7 x 9 matrix of levels (7) and priorities within each level (9). In this representation, the leftmost top cell (level 7, priority 7) is the highest interrupt request while the rightmost lowest cell (level 1, priority 0) is the lowest interrupt request. The following legend is used for this table.

**Table 5-3. Legend for Table 5-3**

Interrupt Request Source	
Interrupt Source Number	Vector Number

#### NOTE

For remapped and forced interrupts, the interrupt source number entry indicates the register or register field that enables the corresponding interrupt.

**Table 5-4. ColdFire V1[Level][Priority within Level] Matrix Interrupt Assignments**

	Priority within Level									
	7	6	5	4	midpoint	3	2	1	0	
7	—		—		—		IRQ		LVD	
	0		64		1		65		2	
6	remppped		remppped		SPI1		SPI2		—	
	PL6P 7		PL6P 6		3 67		4 68		5 69	
5	TPM1_ch0		TPM1_ch1		TPM1_ch2		TPM1_ch3		—	
	7 71		8 72		9 73		10 74		11 75	
4	TPM2_ch0		TPM2_ch1		—		—		—	
	14 78		15 79		—		—		TPM2_ovfl	
3	SCI1_error		SCI1_rx		SCI1_tx		SCI2_err		—	
	17 81		18 82		19 83		20 84		21 85	
2	KBI		ADC		ACMP		IIC1		—	
	23 87		24 88		25 89		26 90		27 91	
1	CAN_wu		CAN_err		CAN_rx		—		—	
	30 94		31 95		32 96		—		CAN_tx	
									RNGA_err	
									force_lvl1	
									FRC LVL1	
									110	

## 5.5 Low-Voltage Detect (LVD) System

The MCF51JM128 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. The LVD is disabled upon entering Stop2 or Stop3 modes unless the LVDSE bit is set. 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,  $V_{LVDL}$ . 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 to 1. 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 LVD bit in the SRS register is set following an LVD reset or POR.

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

The LVD system has a low voltage warning flag (LVWF) to indicate to the user 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 MCF51JM128 series microcontroller includes a clock gating system to manage the bus clock sources to the individual peripherals. Using this system, the user can enable or disable the bus clock to each of the peripherals at the clock source, eliminating unnecessary clocks to peripherals 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, user software 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, SCGC2). 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 has no effect.

### NOTE

User software should 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.

### NOTE

The clock for the MSCAN module can be disabled in wait mode by setting CANCTL0[CSWAI] (see [Section 11.3.1, “MSCAN Control Register 0 \(CANCTL0\)”](#)).

In stop modes, the bus clock is disabled for all gated peripherals, regardless of the settings in the SCGC1 and SCGC2 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 high-page register space are related to reset and interrupt systems.

Refer to [Section 4.2, “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 used to 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 5-5. IRQSC Register Field Descriptions**

Field	Description
7	Reserved, should be cleared.
6 IRQPDD	<b>Interrupt Request (IRQ) Pull Device Disable</b> — This read/write control bit is used to disable 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 enabled if IRQPE = 1. 1 IRQ pull device disabled if IRQPE = 1.
5 IRQEDG	<b>Interrupt Request (IRQ) Edge Select</b> — This read/write control bit is used to select the polarity of edges or levels on the IRQ pin that cause IRQF to be set. The IRQMOD control bit determines whether the IRQ pin is sensitive to both edges and levels or only edges. When IRQEDG = 1 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. 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.

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

Field	Description
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 when IRQF = 1.
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 5.4.1.2, “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.

## 5.7.2 System Reset Status Register (SRS)

This high page 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 is 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	Note <sup>1</sup>	Note <sup>1</sup>	Note <sup>1</sup>	Note <sup>1</sup>	0	0	0

<sup>1</sup> Any of these reset sources that are active at the time of reset entry causes the corresponding bit(s) to be set. Bits corresponding to sources that are not active at the time of reset entry are cleared.

**Figure 5-2. System Reset Status (SRS)****Table 5-6. SRS Register Field Descriptions**

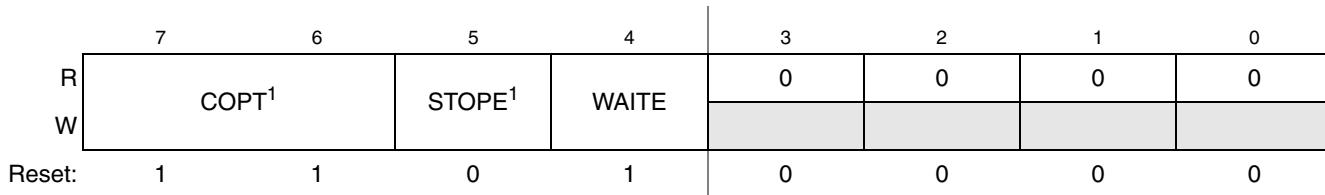
Field	Description
7 POR	<b>Power-On Reset</b> — Reset was caused by the power-on detection logic. Because the internal supply voltage was ramping up at the time, the low-voltage reset (LVD) status bit is also set to indicate that the reset occurred while the internal supply was below the LVD threshold. 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 SOPT1[COPT] = 0b00. 0 Reset not caused by COP timeout. 1 Reset caused by COP timeout.

**Table 5-6. SRS Register Field Descriptions (continued)**

Field	Description
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 ((SOPT[STOPE] = 0) && (SOPT[WAITE] = 0)). The HALT instruction is considered illegal if the BDM interface is disabled by XCSR[ENBDM] = 0. 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] = 0. When CPUCR[ARD] = 1, 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.
2 LOC	<b>Loss-of-Clock Reset</b> — Reset was caused by a loss of external clock. 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 (SOPT1) Register

This high-page 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 should be written to 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 (SOPT1) Register****Table 5-7. SOPT1 Field Descriptions**

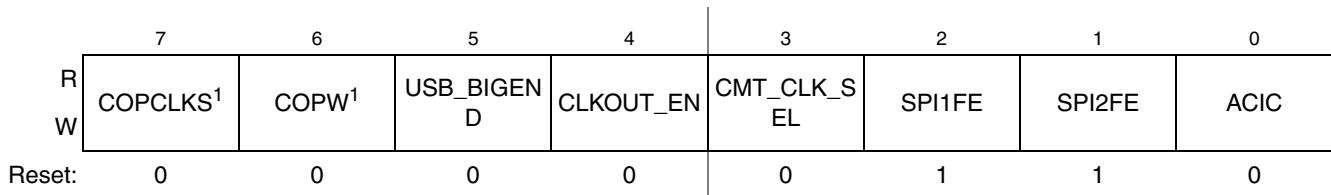
Field	Description
7–6 COPT	<b>COP Watchdog Timeout</b> — These write-once bits select the timeout period of the COP. COPT along with SOPT2[COPCLKS] defines the COP timeout period as described in <a href="#">Table 5-9</a> .
5 STOPE	<b>Stop Mode Enable</b> — This write-once bit is used to enable stop mode. 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].

**Table 5-7. SOPT1 Field Descriptions (continued)**

Field	Description
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].
3–0	Reserved, must be cleared.

## 5.7.4 System Options 2 (SOPT2) Register

This high page register contains bits to configure microcontroller specific features on the MCF51JM128 series microcontrollers.



<sup>1</sup> This bit can be written to only one time after reset. Additional writes are ignored.

**Figure 5-4. System Options 2 (SOPT2) Register****Table 5-8. SOPT2 Register Field Descriptions**

Field	Description
7 COPCLKS	<b>COP Watchdog Clock Select</b> — This write-once bit selects the clock source of the COP watchdog. 0 Internal 1 kHz LPOCLK is source to COP. 1 Bus clock is source to COP.
6 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
5 USB_BIGEND	<b>USB Big Endian</b> — This bit is used to control the endianess (byte order) in USB: 0 Little Endian 1 Big Endian
4 CLKOUT_EN	<b>Clock Output Enable</b> — This bit is used to mux out the BUSCLK clock to PTF4. 0 BUSCLK is not available on the external pin PTF4. 1 BUSCLK is available on the external pin PTF4.
3 CMT_CLK_SEL	<b>CMT Clock Select</b> — This bit selects between BUSCLK and BUSCLK/3 to the CMT module. 0 Clock source to CMT is BUSCLK. 1 Clock source to CMT is BUSCLK/3.
2 SPI1FE	<b>SPI1 Ports Input Filter Enable</b> — This bit enables the input filter on the SPI1 port pins. 0 Input filter disabled (allows for higher maximum baud rate). 1 Input filter enabled (eliminates noise and restricts maximum baud rate).

**Table 5-8. SOPT2 Register Field Descriptions (continued)**

Field	Description
1 SPI2FE	<b>SPI2 Ports Input Filter Enable</b> — This bit enables the input filter on the SPI2 port pins. 0 Input filter disabled (allows for higher maximum baud rate). 1 Input filter enabled (eliminates noise and restricts maximum baud rate).
0 ACIC	<b>Analog Comparator to Input Capture Enable</b> — This bit connects the output of the ACMP to TPM1 input channel 0. See <a href="#">Chapter 20, “Analog Comparator (S08ACMPV2)</a> , and <a href="#">Chapter 22, “Timer/PWM Module,”</a> for more details on this feature. 0 ACMP output not connected to TPM1 input channel 0. 1 ACMP output connected to TPM1 input channel 0.

**Table 5-9. COP Configuration Options**

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

<sup>1</sup> Windowed COP operation requires the user to clear the COP timer in the last 25% of the selected timeout 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.

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

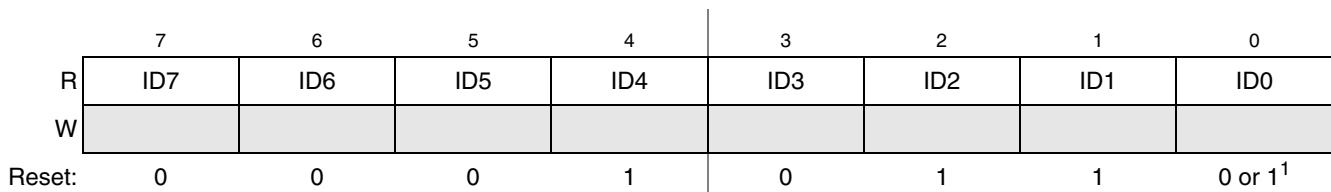
These high page read-only registers are included so host development systems can identify the ColdFire derivative. This allows the development software to recognize where specific memory blocks, registers, and control bits are located in a target microcontroller.

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 [Chapter 6, “ColdFire Core,”](#) for more information.

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

**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 MCF51JM128 series microcontrollers are hard coded to the value 0xC16 (if the CAU is disabled) or 0xC17 (if the CAU is enabled). See also ID bits in <a href="#">Table 5-11</a> .



<sup>1</sup> 0 if the CAU is disabled; 1 if the CAU is enabled.

**Figure 5-6. 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 MCF51JM128 series microcontrollers are hard coded to the value 0xC16 (if the CAU is disabled) or 0xC17 (if the CAU is enabled). See also ID bits in <a href="#">Table 5-10</a> .

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

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



<sup>1</sup> LVWF is set in the case 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 5-7. System Power Management Status and Control 1 Register (SPMSC1)**

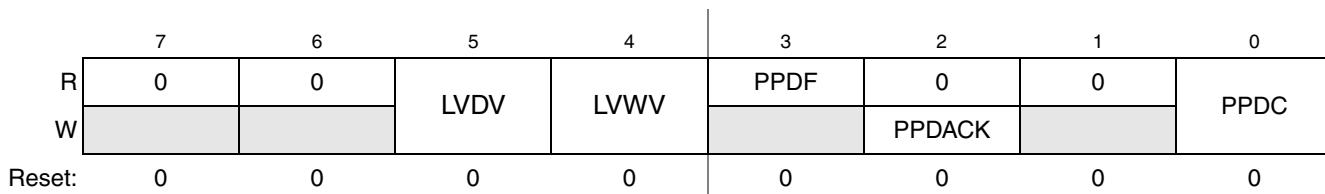
**Table 5-12. SPMSC1 Register Field Descriptions**

Field	Description
7 LWVF	<b>Low-Voltage Warning Flag</b> — The LVWF bit indicates the low-voltage warning status. 0 Low-voltage warning is not present. 1 Low-voltage warning is present or was present.
6 LWACK	<b>Low-Voltage Warning Acknowledge</b> — If LVWF = 1, a low-voltage condition has occurred. To acknowledge this low-voltage warning, write 1 to LWACK, which automatically clears LVWF to 0 if the low-voltage warning is no longer present.
5 LWIE	<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 = 1.
4 LVDRE	<b>Low-Voltage Detect Reset Enable</b> — This write-once bit enables LVDF events to generate a hardware reset (provided LVDE = 1). 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> — Provided LVDE = 1, this read/write bit determines whether 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.

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

This high page register contains status and control bits to configure the low power run and wait modes as well as configure the stop mode behavior of the microcontroller. See [Section 3.8, “Stop Modes,”](#) for more information.

SPMSC2 is not reset when exiting from STOP2.

**Figure 5-8. System Power Management Status and Control 2 Register (SPMSC2)**

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

Field	Description
7-6	Reserved, should be cleared.
5 LVDV	<b>Low-Voltage Detect Voltage Select</b> — The LVDV bit selects the LVD trip point voltage ( $V_{LVD}$ ). 0 Low trip point selected ( $V_{LVD} = V_{LVDL}$ ). 1 High trip point selected ( $V_{LVD} = V_{LVDH}$ ).
4 LVWV	<b>Low-Voltage Warning Voltage Select</b> — The LVWV bit selects the LVW trip point voltage ( $V_{LVW}$ ). 0 Low trip point selected ( $V_{LVW} = V_{LVWL}$ ). 1 High trip point selected ( $V_{LVW} = V_{LVWH}$ ).
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 a 1 to PPDACK clears the PPDF 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 = 1. If PPDC and LPR are set in a single write instruction, only PPDC is set. PPDE must be set for PPDC to be set. PPDC is a write-once-only bit. 0 Stop3 low power mode enabled. 1 Stop2 partial power-down mode enabled.

<sup>1</sup> See Electrical Characteristics appendix for minimum and maximum values.

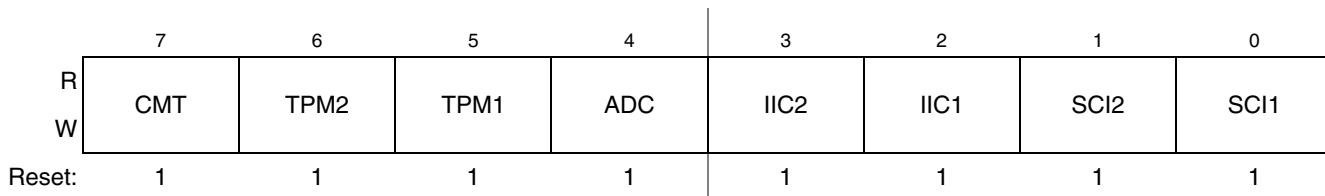
**Table 5-14. LVD and LVW Trip Point Typical Values<sup>1</sup>**

LVDV:LVWV	LVW Trip Point	LVD Trip Point
00	$V_{LVWL} = 2.74$ V	$V_{LVDL} = 2.56$ V
01	$V_{LVWL} = 2.92$ V	
10	$V_{LVWL} = 4.3$ V	$V_{LVDL} = 4.0$ V
11	$V_{LVWL} = 4.6$ V	

<sup>1</sup> See the *MCF51JM128 Data Sheet* for minimum and maximum values.

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

This high page register contains control bits to enable or disable the bus clock to the TPMx, ADC, IICx, and SCIx modules. Gating off the clocks to unused peripherals is used to reduce the microcontroller's run and wait currents. See [Section 5.6, “Peripheral Clock Gating,”](#) for more information.

**Figure 5-9. System Clock Gating Control 1 Register (SCGC1)**

**Table 5-15. SCGC1 Register Field Descriptions**

Field	Description
7 CMT	<b>CMT Clock Gate Control</b> — This bit controls the clock gate to the CMT module. 0 Bus clock to the CMT module is disabled. 1 Bus clock to the CMT module is enabled.
6 TPM2	<b>TPM2 Clock Gate Control</b> — This bit controls the clock gate to the TPM2 module. 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> — This bit controls the clock gate to the TPM1 module. 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> — This bit controls the clock gate to the ADC module. 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> — This bit controls the clock gate to the IIC2 module. 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> — This bit controls the clock gate to the IIC1 module. 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> — This bit controls the clock gate to the SCI2 module. 0 Bus clock to the SCI2 module is disabled. 1 Bus clock to the SCI2 module is enabled.
0 SCI1	<b>SCI1 Clock Gate Control</b> — This bit controls the clock gate to the SCI1 module. 0 Bus clock to the SCI1 module is disabled. 1 Bus clock to the SCI1 module is enabled.

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

This high page register contains control bits to enable or disable the bus clock to the USB IRQ, KBI, ACMP, RTC, and SPIx modules. Gating off the clocks to unused peripherals is used to reduce the microcontroller's run and wait currents. See [Section 5.6, “Peripheral Clock Gating,”](#) for more information.

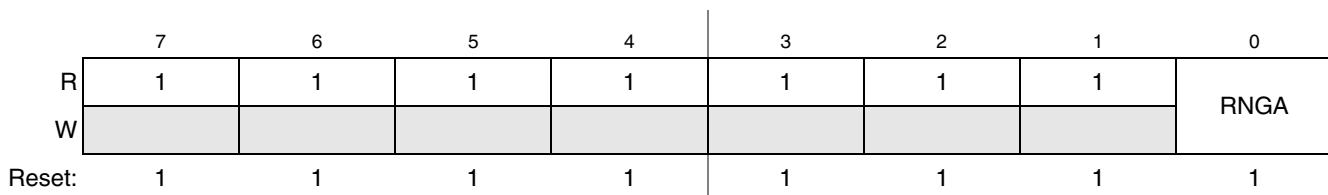
**Figure 5-10. System Clock Gating Control 2 Register (SCGC2)**

**Table 5-16. SCGC2 Register Field Descriptions**

Field	Description
7 USB	<b>USB Clock Gate Control</b> — This bit controls the bus clock gate to the USB module. 0 Bus clock to the USB module is disabled. 1 Bus clock to the USB module is enabled.
6 FLS	<b>FTSR Clock Gate Control</b> — This bit controls the bus clock gate to the flash registers. 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> — This bit controls the bus clock gate to the IRQ module. 0 Bus clock to the IRQ module is disabled. 1 Bus clock to the IRQ module is enabled.
4 KBI	<b>KBI Clock Gate Control</b> — This bit controls the clock gate to both of the KBI modules. 0 Bus clock to the KBI modules is disabled. 1 Bus clock to the KBI modules is enabled.
3 ACMP	<b>ACMP Clock Gate Control</b> — This bit controls the clock gate to both of the ACMP modules. 0 Bus clock to the ACMP modules is disabled. 1 Bus clock to the ACMP modules is enabled.
2 RTC	<b>RTC Clock Gate Control</b> — This bit controls the bus clock gate to the RTC module. Only the bus clock is gated; the clock source of RTC is not controlled by this bit. 0 Bus clock to the RTC module is disabled. 1 Bus clock to the RTC module is enabled.
1 SPI2	<b>SPI2 Clock Gate Control</b> — This bit controls the clock gate to the SPI2 module. 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> — This bit controls the clock gate to the SPI1 module. 0 Bus clock to the SPI1 module is disabled. 1 Bus clock to the SPI1 module is enabled.

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

This high page register contains control bits to enable or disable the bus clock to the USB IRQ, KBI, ACMP, RTC, and SPIx modules. Gating off the clocks to unused peripherals is used to reduce the microcontroller's run and wait currents. See [Section 5.6, “Peripheral Clock Gating,”](#) for more information.

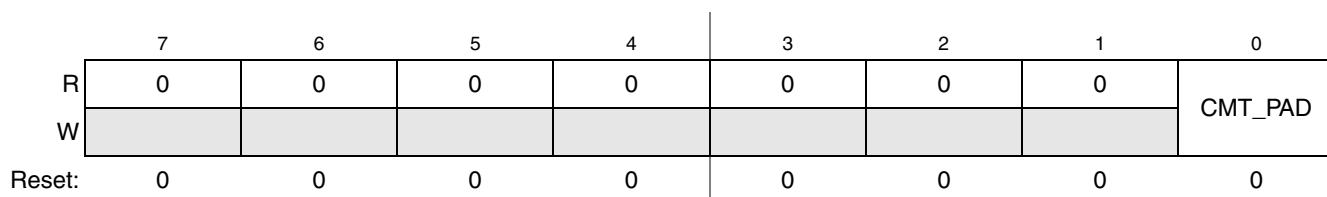
**Figure 5-11. System Clock Gating Control 3 Register (SCGC3)**

**Table 5-17. SCGC3 Register Field Descriptions**

Field	Description
7–1	<b>Reserved, should be set.</b>
0 RNGA	<b>RNGA Clock Gate Control</b> — This bit controls the clock gate to the RNGA module. 0 Bus clock to the RNGA module is disabled. 1 Bus clock to the RNGA module is enabled.

### 5.7.11 System Options 3 Register (SOPT3)

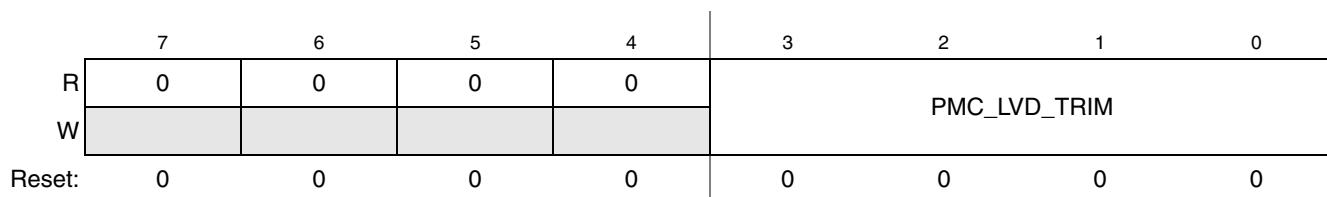
This register contains a bit that controls the pad drive strength for the CMT module.

**Figure 5-12. System Options 3 Register (SOPT3)****Table 5-18. SOPT3 Field Descriptions**

Field	Description
7–1	<b>Reserved, should be cleared.</b>
0 CMT_PAD	<b>CMT pad drive strength</b> — This bit controls the CMT pad drive strength by bounding two pads together. 0 single-pad drive strength 1 dual-pads bounding drive strength

### 5.7.12 System Options 4 Register (SOPT4)

This register contains bits that control the PMC LVD trim.

**Figure 5-13. System Options 4 Register (SOPT4)****Table 5-19. SOPT4 Field Descriptions**

Field	Description
7–4	<b>Reserved, should be cleared.</b>
3–0 PMC_LVD_TRIM	<b>PMC LVD TRIM.</b> When a value is not written to this field, the PMC LVD trim value is taken from the IFR. When a value is written to this field, it supersedes the IFR trim value.



# Chapter 6

## ColdFire Core

### 6.1 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*.

#### 6.1.1 Overview

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

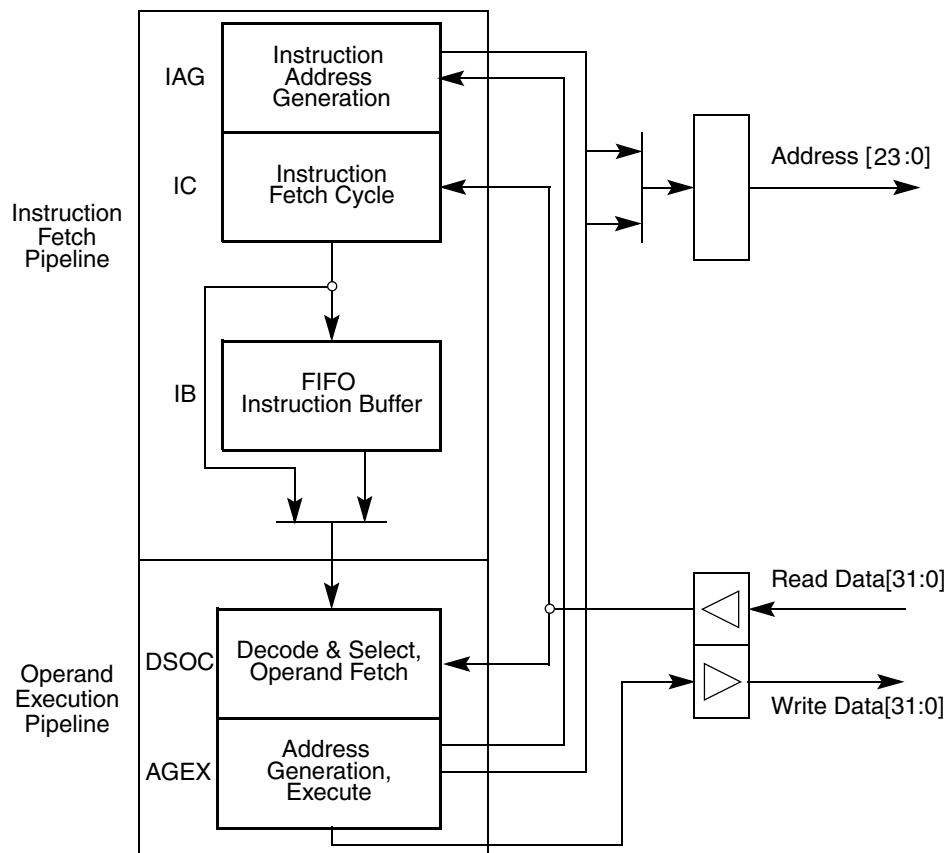


Figure 6-1. V1 ColdFire Core Pipelines

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.

## 6.2 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 6-1](#) 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 6-1. 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 6.3.3.14/6-20	No	6.2.1/6-3
Load: 0x61 Store: 0x41	Data Register 1 (D1)	32	R/W	See 6.3.3.14/6-20	No	6.2.1/6-3
Load: 0x6–7 Store: 0x4–7	Data Register –7 (D–D7)	32	R/W	POR: Undefined Else: Unaffected	No	6.2.1/6-3
Load: 0x68–E Store: 0x48–E	Address Register 0–6 (A0–A6)	32	R/W	POR: Undefined Else: Unaffected	No	6.2.2/6-4
Load: 0x6F Store: 0x4F	User A7 Stack Pointer (A7)	32	R/W	POR: Undefined Else: Unaffected	No	6.2.3/6-4
Load: 0xEE Store: 0xCE	Condition Code Register (CCR)	8	R/W	POR: Undefined Else: Unaffected	No	6.2.4/6-5
Load: 0xEF Store: 0xCF	Program Counter (PC)	32	R/W	Contents of location 0x(00)00_0004	No	6.2.5/6-6
<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	6.2.3/6-4
Load: 0xE1 Store: 0xC1	Vector Base Register (VBR)	32	R/W	See section	Yes; Rc = 0x801	6.2.6/6-6
Load: 0xE2 Store: 0xC2	CPU Configuration Register (CPUCR)	32	W	See section	Yes; Rc = 0x802	6.2.7/6-7
Load: 0xEE Store: 0xCE	Status Register (SR)	16	R/W	0x27--	No	6.2.8/6-8

<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 23, “Version 1 ColdFire Debug \(CF1\\_DEBUG\).”](#) (These BDM commands are not similar to other ColdFire processors.)

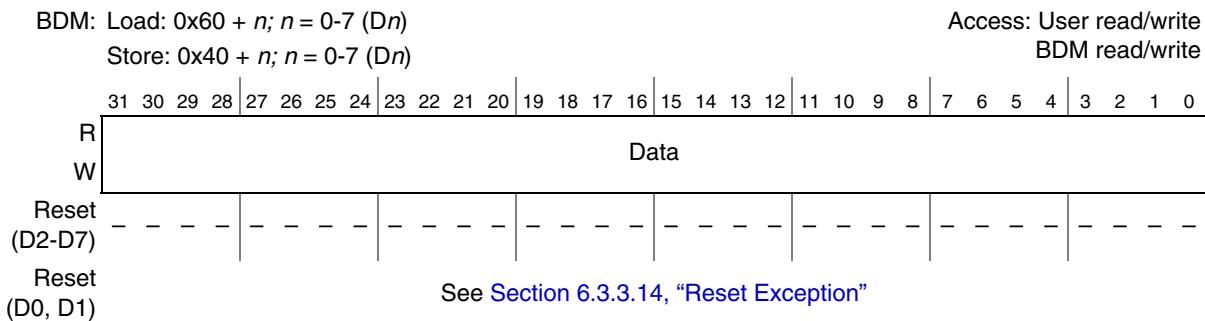
<sup>2</sup> If the given register is written using the MOVEC instruction, the 12-bit control register address (Rc) is also specified.

## 6.2.1 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**

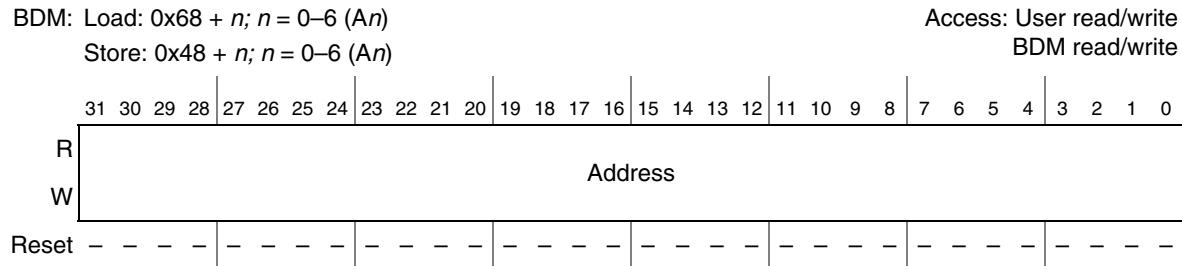
Registers D0 and D1 contain hardware configuration details after reset. See [Section 6.3.3.14, “Reset Exception”](#) for more details.



**Figure 6-2. Data Registers (D0–D7)**

## 6.2.2 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 6-3. Address Registers (A0–A6)**

## 6.2.3 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 USP must be initialized using the `move.l Ay,USP` instruction before any entry into user mode.

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

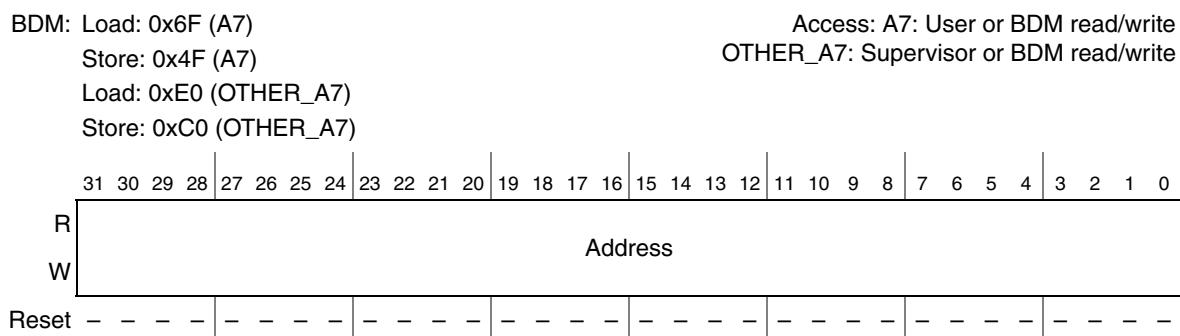


Figure 6-4. Stack Pointer Registers (A7 and OTHER\_A7)

#### 6.2.4 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.

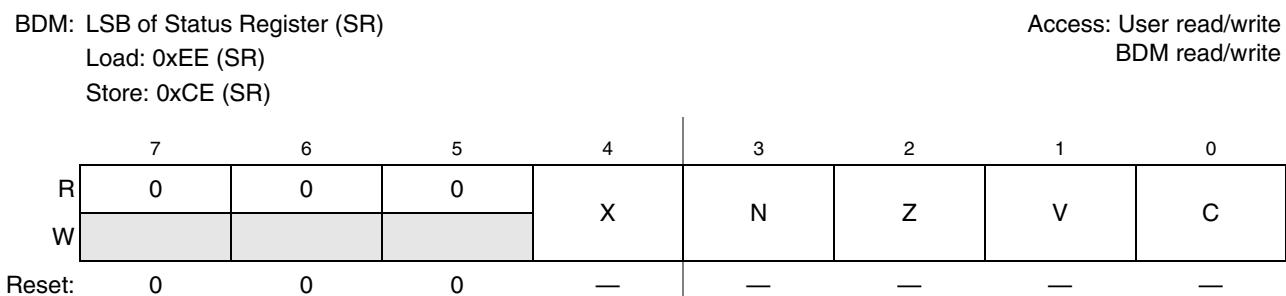


Figure 6-5. Condition Code Register (CCR)

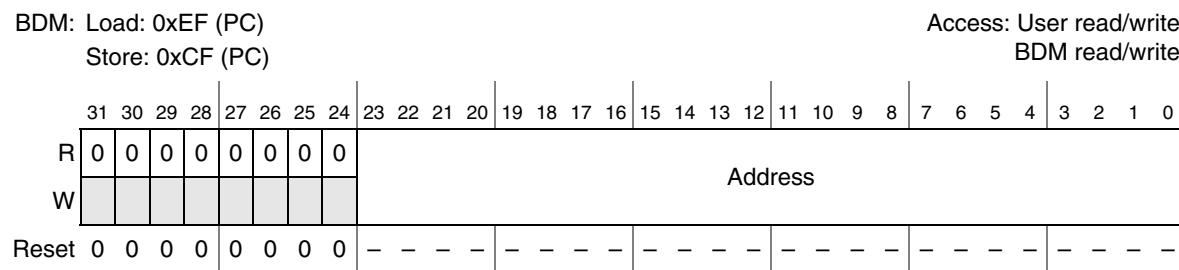
**Table 6-2. CCR Field Descriptions**

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.
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.

## 6.2.5 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 0x(00)00\_0004.

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

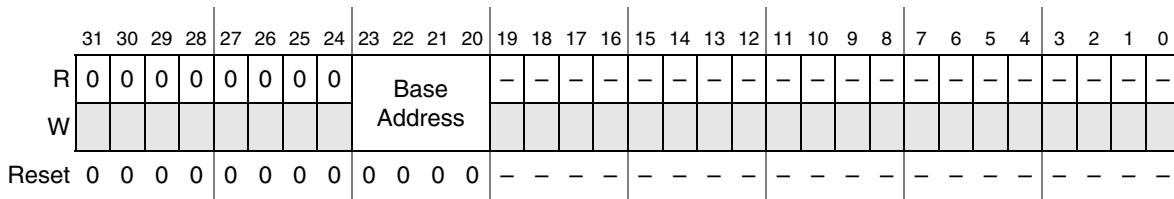
## 6.2.6 Vector Base Register (VBR)

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



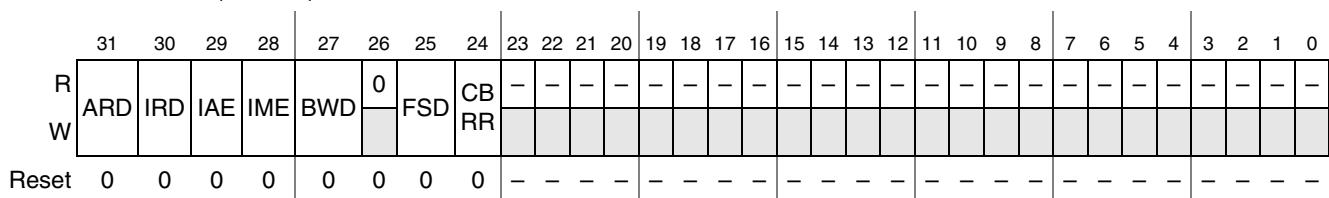
**Figure 6-7. Vector Base Register (VBR)**

### 6.2.7 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 6-8. CPU Configuration Register (CPUCR)**

**Table 6-3. CPUCR Field Descriptions**

Field	Description
31 ARD	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.
	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.
30 IRD	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.
	0 The detection of these types of exception conditions generate a reset event.
	1 No reset is generated in response to these exception conditions.
29 IAE	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.
	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.
28 IME	Interrupt mask enable. Forces the processor to raise the interrupt level mask (SR[I]) to 7 during every interrupt exception.
	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.

**Table 6-3. 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
24–0	Reserved, must be cleared.

### 6.2.8 Status Register (SR)

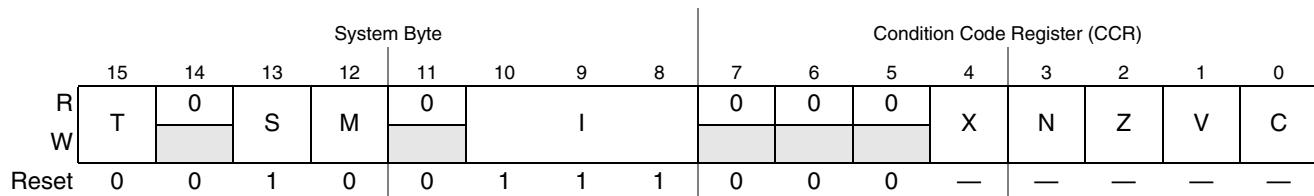
The SR stores the processor status and includes the CCR, the interrupt priority mask, and other control bits. In supervisor mode, software can access the entire SR. In user mode, only the lower 8 bits (CCR) are accessible. The control bits indicate the following states for the processor: trace mode (T 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 6-9. Status Register (SR)****Table 6-4. 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 6-4. 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 6.2.4, “Condition Code Register (CCR)”. </a>

## 6.3 Functional Description

### 6.3.1 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 6-5](#) 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 6-5. 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.

### 6.3.2 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 6-10](#), 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 6-6](#)). 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 6-6. 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 6-6. Exception Vector Assignments (continued)**

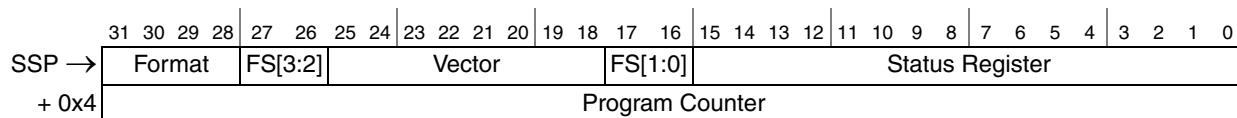
Vector Number(s)	Vector Offset (Hex)	Stacked Program Counter	Assignment
2	0x008	Fault	Access error
3	0x00C	Fault	Address error
4	0x010	Fault	Illegal instruction
5–7	0x014–0x01C	—	Reserved
8	0x020	Fault	Privilege violation
9	0x024	Next	Trace
10	0x028	Fault	Unimplemented line-A opcode
11	0x02C	Fault	Unimplemented 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*.

### 6.3.2.1 Exception Stack Frame Definition

Figure 6-10 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 6-10. 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 6-7](#).

**Table 6-7. 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 6-8](#).

**Table 6-8. 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 6-6](#).

### 6.3.2.2 S08 and ColdFire Exception Processing Comparison

This section presents a brief summary comparing the exception processing differences between the S08 and V1 ColdFire processor families.

**Table 6-9. Exception Processing Comparison**

Attribute	S08	V1 ColdFire
Exception Vector Table	32, 2-byte entries, fixed location at upper end of memory	103, 4-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 + 39 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
Software IACK	No	Yes
Exit Instruction from ISR	RTI	RTE

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 performance noticeably.

Emulation of the S08's 1-level IRQ processing can easily be managed by software convention within the ColdFire interrupt service routines. For this type of operation, only two of the seven interrupt levels are used:

- SR[I] equals 0 indicates interrupts are enabled
- SR[I] equals 7 indicates interrupts are disabled

Recall that ColdFire treats true level 7 interrupts as edge-sensitive, non-maskable requests. Typically, only the IRQ input pin and a low-voltage detect are assigned as level 7 requests. All the remaining interrupt requests (levels 1-6) are masked when SR[I] equals 7. In any case, all ColdFire processors guarantee that the first instruction of any exception handler is executed before interrupt sampling resumes. By making the first instruction of the ISR a store/load status register (`STLDSR #0x2700`) or a move-to-SR (`MOVE.W #2700,SR`) instruction, interrupts can be safely disabled until the service routine is exited with an RTE instruction that lowers the SR[I] back to level 0. The same functionality can also be provided without an explicit instruction by setting CPUCR[IME] because this forces the processor to load SR[I] with 7 on each interrupt exception.

### 6.3.3 Processor Exceptions

#### 6.3.3.1 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.

#### 6.3.3.2 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.

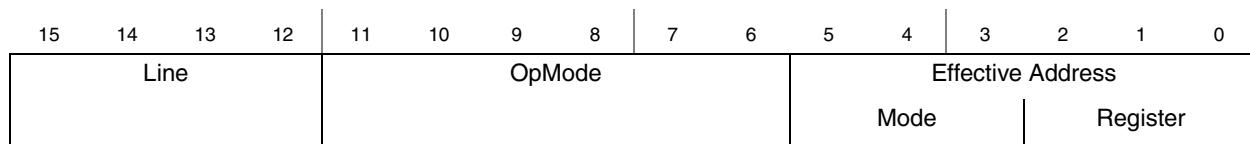
Any attempted use of a word-sized index register ( $Xn.w$ ) or a scale factor of eight on an indexed effective addressing mode generates an address error, as does an attempted execution of a full-format indexed addressing mode, which is defined by bit 8 of extension word 1 being set.

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.

### 6.3.3.3 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 6-11](#). The opword line definition is shown in [Table 6-10](#).



**Figure 6-11. ColdFire Instruction Operation Word (Opword) Format**

**Table 6-10. 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.

### 6.3.3.4 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.

### 6.3.3.5 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.

### **6.3.3.6 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.

### **6.3.3.7 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.

### **6.3.3.8 Debug Interrupt**

See [Chapter 23, “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.

### **6.3.3.9 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.

### 6.3.3.10 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.

### 6.3.3.11 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 6.3.3.14, “Reset Exception,”](#) for details.

For this device, attempted execution of valid integer divide opcodes and all MAC and EMAC instructions result in the unsupported instruction exception.

### 6.3.3.12 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.

### 6.3.3.13 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.

### 6.3.3.14 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 6-12](#).

BDM: Load: 0x60 (D0)								Access: User read-only BDM read-only							
R				PF				VER				REV			
W															
Reset	1	1	0	0	1	1	1	1	0	0	0	1	0	0	0
R				15 14 13 12				11 10 9 8				7 6 5 4			
W				MAC DIV 0 0				0 CAU 0 0				ISA			
Reset	0	0	0	0	0	1	0	0	0	0	1	0	1	0	1

Figure 6-12. D0 Hardware Configuration Info

**Table 6-11. 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.
11	Reserved.
10 CAU	Cryptographic acceleration unit present. This bit signals if the optional cryptographic acceleration unit (CAU) is present in the processor core. 0 CAU coprocessor engine not present in core. 1 CAU coprocessor engine is present in core. (This is the value used for this device.)
9–8	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)								Access: User read-only BDM read-only							
Store: 0x41 (D1)															
R	0	0	0	1	0	0	0	0	FLASHSZ				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	0	0	0	0

Figure 6-13. D1 Hardware Configuration Info

Table 6-12. D1 Hardware Configuration Information Field Description

Field	Description
31–24	Reserved.
23–19 FLASHSZ	Flash bank size. 00000-01110 No flash 10000 64 KB flash 10010 128 KB flash (This is the value used for this device) 10011 96 KB flash 10100 256 KB flash 10110 512 KB flash Else Reserved for future use
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

**Table 6-12. D1 Hardware Configuration Information Field Description (continued)**

Field	Description
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 (This is the value used for this device) 01111 24 KB 01110 32 KB 10000 64 KB 10010 128 KB Else Reserved for future use
2-0	Reserved.

### 6.3.4 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.

#### 6.3.4.1 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 6-13](#).

**Table 6-13. 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

### 6.3.4.2 MOVE Instruction Execution Times

[Table 6-14](#) lists execution times for MOVE.{B,W} instructions; [Table 6-15](#) 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.

$$\begin{array}{ll} \text{ET with } \{\langle\text{ea}\rangle = (\text{d16}, \text{PC})\} & \text{equals ET with } \{\langle\text{ea}\rangle = (\text{d16}, \text{An})\} \\ \text{ET with } \{\langle\text{ea}\rangle = (\text{d8}, \text{PC}, \text{Xi}^*\text{SF})\} & \text{equals ET with } \{\langle\text{ea}\rangle = (\text{d8}, \text{An}, \text{Xi}^*\text{SF})\} \end{array}$$

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

**Table 6-14. 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)	—	—	—

**Table 6-14. MOVE Byte and Word Execution Times (continued)**

Source	Destination						
	Rx	(Ax)	(Ax)+	-(Ax)	(d16,Ax)	(d8,Ax,Xi*SF)	xxx.wl
(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 6-15. 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)	—	—
(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)	—	—	—

### 6.3.4.3 Standard One Operand Instruction Execution Times

**Table 6-16. 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)	—
CLRL.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)	—	—	—	—	—	—	—

**Table 6-16. One Operand Instruction Execution Times (continued)**

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An)	(d8,An,Xn*SF)	xxx.wl	#xxx
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)

**6.3.4.4 Standard Two Operand Instruction Execution Times****Table 6-17. 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)	—	—	—

**Table 6-17. 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
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)
CMP.L	<ea>,Rx	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
CMPI.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)	—	—	—	—	—	—	—
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)	—	—	—	—	—	—	—

### 6.3.4.5 Miscellaneous Instruction Execution Times

**Table 6-18. 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)	—	—	—	—	—	—	—
MOV3Q.L	#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)	—	—	—	—	—	—	—

**Table 6-18. Miscellaneous Instruction Execution Times (continued)**

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An)	(d8,An,Xn*SF)	xxx.wl	#xxx
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)	—	—	—	—	—	—	—
TPF.W		1(0/0)	—	—	—	—	—	—	—
TPFL		1(0/0)	—	—	—	—	—	—	—
UNLK	Ax	2(1/0)	—	—	—	—	—	—	—
WDDATA	<ea>	—	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	—
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).

### 6.3.4.6 Branch Instruction Execution Times

**Table 6-19. General Branch Instruction Execution Times**

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An) (d16,PC)	(d8,An,Xi*SF) (d8,PC,Xi*SF)	xxx.wl	#xxx
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)	—

**Table 6-19. General Branch Instruction Execution Times (continued)**

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An) (d16,PC)	(d8,An,Xi*SF) (d8,PC,Xi*SF)	xxx.wl	#xxx
RTE		—	—	7(2/0)	—	—	—	—	—
RTS		—	—	5(1/0)	—	—	—	—	—

**Table 6-20. 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 7**

## **Multipurpose Clock Generator (S08MCGV3)**

### **7.1 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 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 that 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.

For USB operation on the MCF51JM128 series, the MCG must be configured for PLL engaged external (PEE) mode to achieve a MCGOUT frequency of 48 MHz.

## 7.1.1 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).

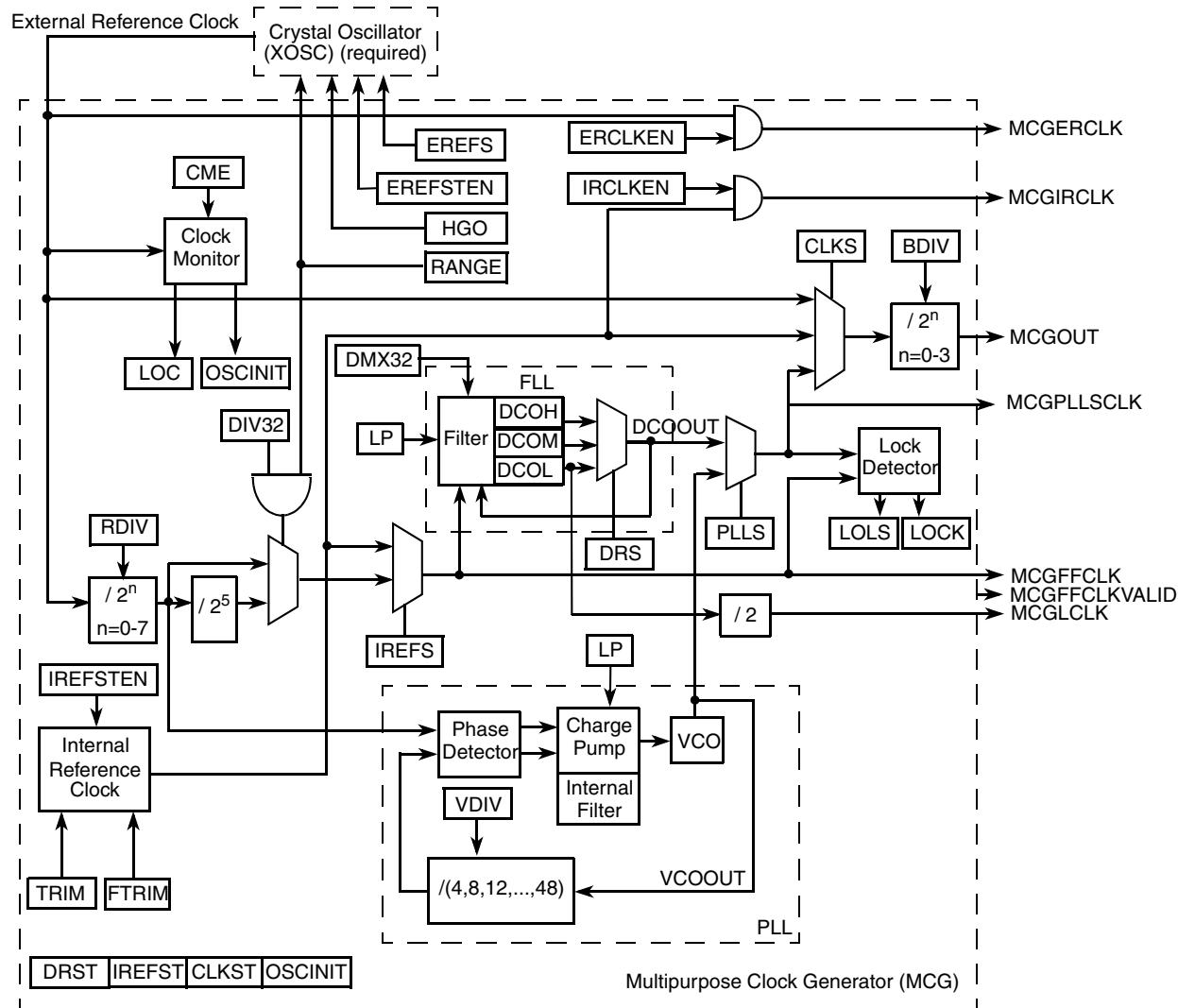


Figure 7-1. Multipurpose Clock Generator (MCG) Block Diagram

**NOTE**

The MCG requires the attachment of a crystal oscillator (XOSC) module, which provides an external reference clock.

## 7.1.2 Modes of Operation

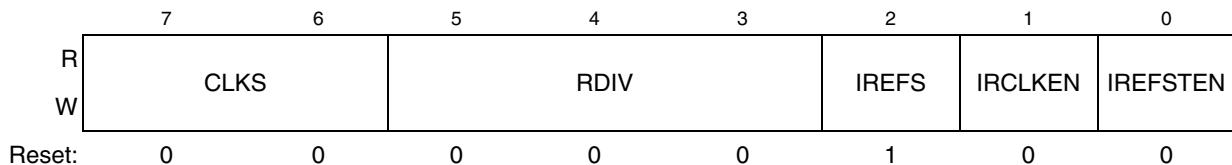
There are several modes of operation for the MCG. For details, see [Section 7.4.1, “MCG Modes of Operation.”](#)

## 7.2 External Signal Description

There are no MCG signals that connect off chip.

## 7.3 Register Definition

### 7.3.1 MCG Control Register 1 (MCGC1)



**Figure 7-2. MCG Control Register 1 (MCGC1)**

**Table 7-1. 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 7-2</a> and <a href="#">Table 7-3</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 7-2. 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	Reserved

**Table 7-3. PLL External Reference Divide Factor**

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

### 7.3.2 MCG Control Register 2 (MCGC2)

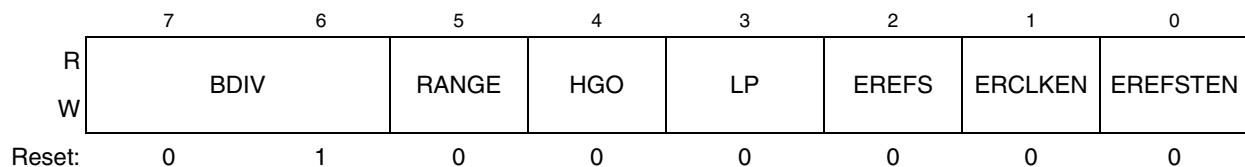
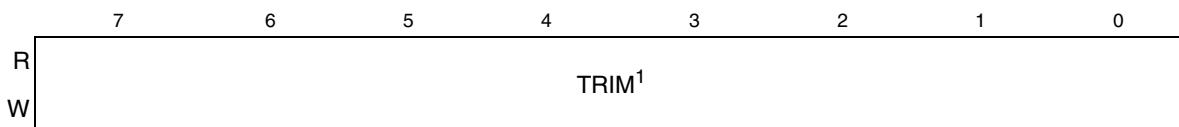


Figure 7-3. MCG Control Register 2 (MCGC2)

Table 7-4. 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 (reset default) 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 16 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 EREFs	<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

### 7.3.3 MCG Trim Register (MCGTRM)



**Figure 7-4. 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 7-5. 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 will adjust twice as much as bit 0). Increasing the binary value in TRIM will increase the period, and decreasing the value will decrease 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>

### 7.3.4 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 7-5. 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 0x0 is loaded.

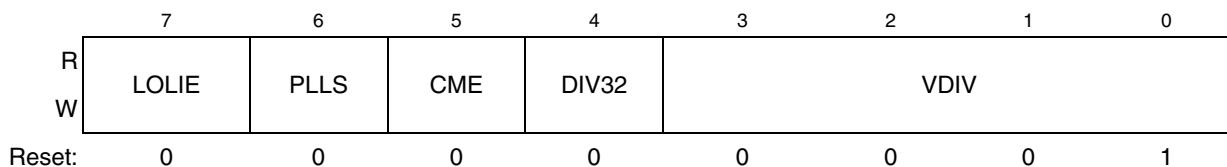
Table 7-6. MCG Status and Control Register Field Description

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 both 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 7-6. MCG Status and Control Register Field Description (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 either EREFS is cleared or when the MCG is in either 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 will increase the period and clearing FTRIM will decrease 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.

### 7.3.5 MCG Control Register 3 (MCGC3)

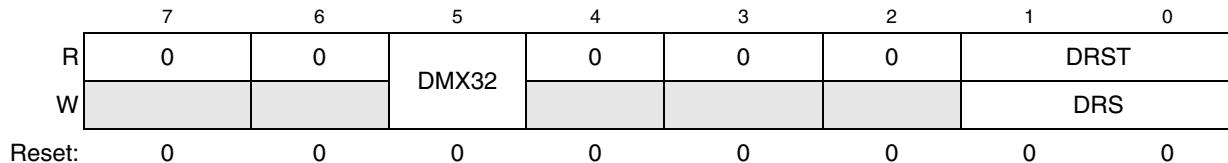
**Figure 7-6. MCG PLL Register (MCGPLL)****Table 7-7. MCG PLL Register Field Descriptions**

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. 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 either 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). Whenever 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 7-7. 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).

### 7.3.6 MCG Control Register 4 (MCGC4)

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

**Table 7-8. 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 7-9</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 7-9</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 either the LP or PLLS bit is set are ignored. 00 Low range. 01 Mid range. 10 High range. 11 Reserved

**Table 7-9. 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
	1	32.768 kHz	1824	59.77 MHz
11		Reserved		

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

### 7.3.7 MCG Test Register (MCGT)

**Table 7-10. MCG Test Register Field Descriptions**

Field	Description
7:6	Reserved for test, user code should not write 1's to these bits.
5	Reserved, user code should not write 1's to these bits
4:1	Reserved for test, user code should not write 1's to these bits.
0	Reserved, user code should not write 1's to these bits

## 7.4 Functional Description

### 7.4.1 MCG Modes of Operation

The MCG operates in one of the modes described in [Table 7-11](#).

#### NOTE

The MCG restricts transitions between modes. For the permitted transitions, see [Section 7.4.2, “MCG Mode State Diagram.”](#)

**Table 7-11. MCG Modes of Operation**

Mode	Related field values	Description
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 a multiplication factor, as selected by the DRS[1:0] and DMX32 bits, 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 a multiplication factor, as selected by the DRS[1:0] and DMX32 bits, 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 a multiplication factor, as selected by the DRS[1:0] and DMX32 bits, 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 a multiplication factor, as selected by the DRS[1:0] and DMX32 bits, 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>

**Table 7-11. MCG Modes of Operation (continued)**

<b>Mode</b>	<b>Related field values</b>	<b>Description</b>
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>	<p>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.</p> <p>In this mode, MCGT[DRST] is read as a 0 regardless of the value of MCGT[DRS].</p>
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> <p>In this mode, MCGT[DRST] is read as a 0 regardless of the value of MCGT[DRS].</p>
Bypassed Low Power Internal (BLPI)	<ul style="list-style-type: none"> <li>MCGC1[IREFS] = 1</li> <li>MCGC1[CLKS] = 01</li> <li>MCGC3[PLLS] = 0</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 FLL bypassed internal (FBI) mode.</p> <p>In this mode, MCGT[DRST] is read as a 0 regardless of the value of MCGT[DRS].</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> <p>In this mode, MCGT[DRST] is read as a 0 regardless of the value of MCGT[DRS].</p>

Table 7-11. MCG Modes of Operation (continued)

Mode	Related field values	Description
Stop	—	<p>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:</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[EREFSSTEN] = 1</li> </ul>

## 7.4.2 MCG Mode State Diagram

Figure 7-9 shows the MCG's mode state diagram. The arrows indicate the permitted mode transitions.

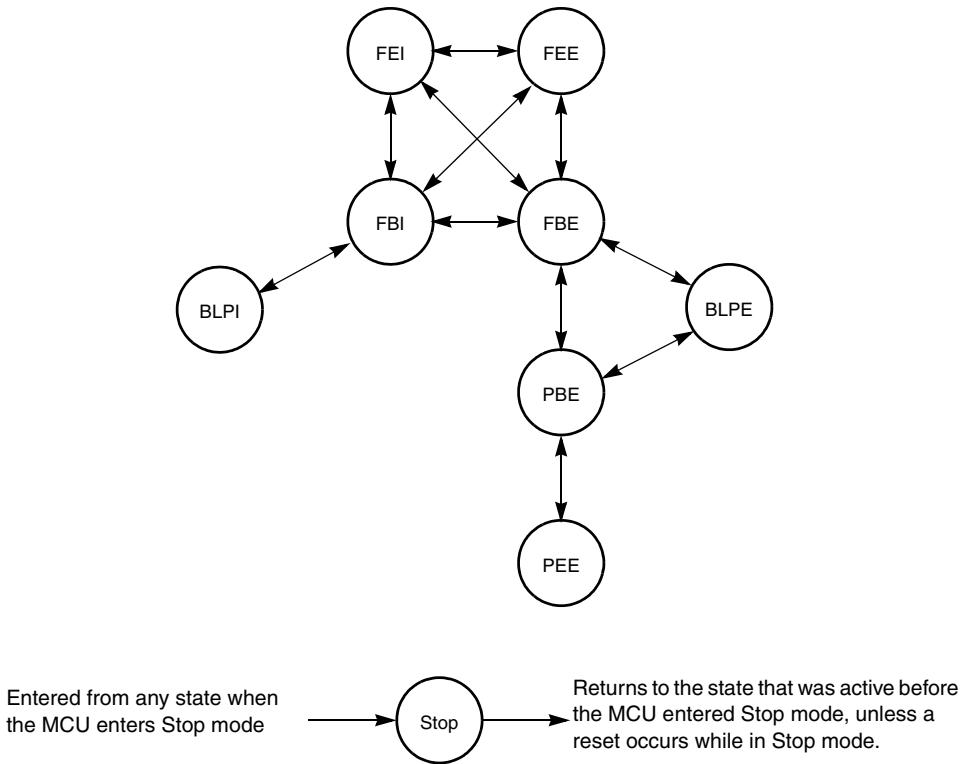


Figure 7-9. MCG Mode State Diagram

## 7.4.3 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 will begin 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 will remain 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. Once 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 7-9](#).

#### 7.4.4 Bus Frequency Divider

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

#### 7.4.5 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.

#### 7.4.6 Internal Reference Clock

When IRCLKEN is set the internal reference clock signal will be 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 will decrease the MCGIRCLK frequency, and writing a smaller value to the MCGTRM register will increase the MCGIRCLK frequency. The TRIM bits will effect 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.

Until MCGIRCLK is trimmed, programming low reference divider (RDIV) factors may result in MCGOUT frequencies that exceed the maximum chip-level frequency and violate the chip-level clock timing specifications (see the [Device Overview](#) chapter).

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

#### 7.4.7 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 will be presented as MGERCLK, which can be used as an additional clock source. When IREFS = 1, the external reference clock will not be used by the FLL or PLL and will only be used as MGERCLK. In these modes, the

frequency can be equal to the maximum frequency the chip-level timing specifications will support (see the [Device Overview](#) chapter).

If EREFSTEN and ERCLKEN bits are both set or the MCG is in FEE, FBE, PEE, PBE or BLPE mode, the external reference clock will keep running during stop mode in order 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 will reset. The LOC bit in the System Reset Status (SRS) register will be set to indicate the error.

## 7.4.8 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. When MCGFFCLK is valid then MCGFFCLKVALID is set to 1. When MCGFFCLK is not valid then MCGFFCLKVALID is set to 0.

This clock is intended for use in systems which 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 either 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.

Note that the FLL can not be used for generation of the system clocks while the PLL is supplying MCGPLLCLK.

## 7.5 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.

### 7.5.1 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 will stabilize in  $t_{irefst}$  microseconds before the FLL can acquire lock. As soon as the internal reference is stable, the FLL will acquire 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.

#### 7.5.1.1 Initializing the MCG

Because the MCG comes out of reset in FEI mode, the only MCG modes which can be directly switched to upon reset are FEE, FBE, and FBI modes (see [Figure 7-9](#)). 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 is still on in FBE mode.
  - The internal reference can optionally be kept running by setting the IRCLKEN bit. This is useful if the application will switch back and forth between internal and external modes. For minimum power consumption, leave the internal reference disabled while in an external clock mode.
4. Once 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. Typical crystal startup times are given in Appendix A, “Electrical Characteristics”.

- 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 is still on and will lock 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 will be 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 will be 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 will be 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.

## 7.5.2 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.

## 7.5.3 MCG Mode Switching

When switching between operational modes of the MCG, certain configuration bits must be changed in order 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) in order 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 will ensure 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 will be 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 7-12. 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 will include 3 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).

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

In this example, the MCG will move 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 will be described. Then a flowchart will be included which 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 in order to select external reference clock as system clock source
    - RDIV (bits 5-3) set to %011, or divide-by-256 because  $8\text{MHz} / 256 = 31.25\text{ kHz}$  which 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 either 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 7-3](#)), 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) still 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 in order to select the output of the PLL as the system clock source

#### Multipurpose Clock Generator (MCGV3)

- b) 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
  - 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

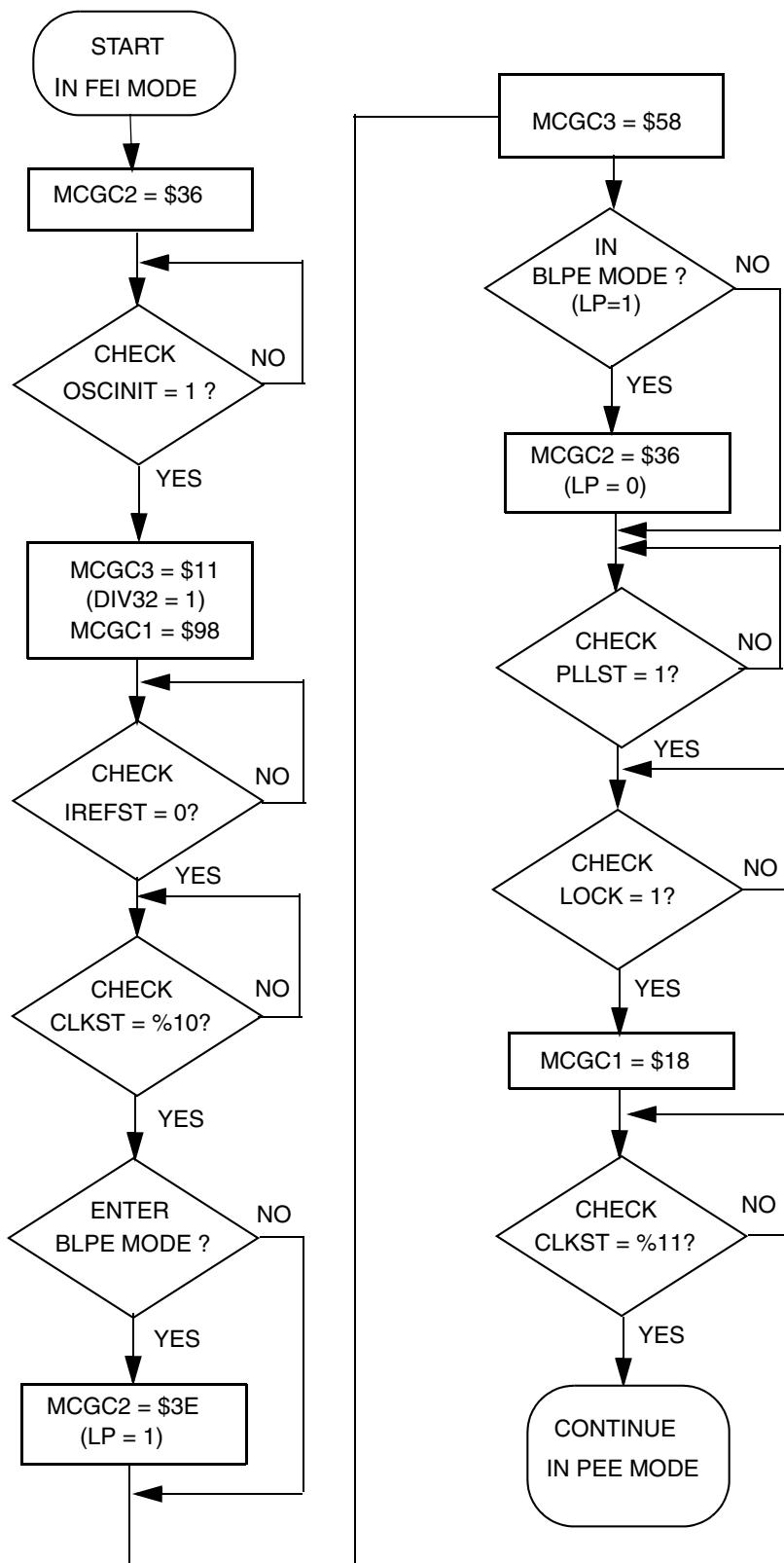


Figure 7-10. Flowchart of FEI to PEE Mode Transition using an 8 MHz crystal

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

In this example, the MCG will move 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 will be described. Then a flowchart will be included which illustrates the sequence.

1. First, PEE must transition to PBE mode:
  - a)  $\text{MCGC1} = 0x98$  (%10011000)
    - CLKS (bits 7 and 6) set to %10 in order 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 either 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 7-2](#)), 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 is still 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 in order 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.

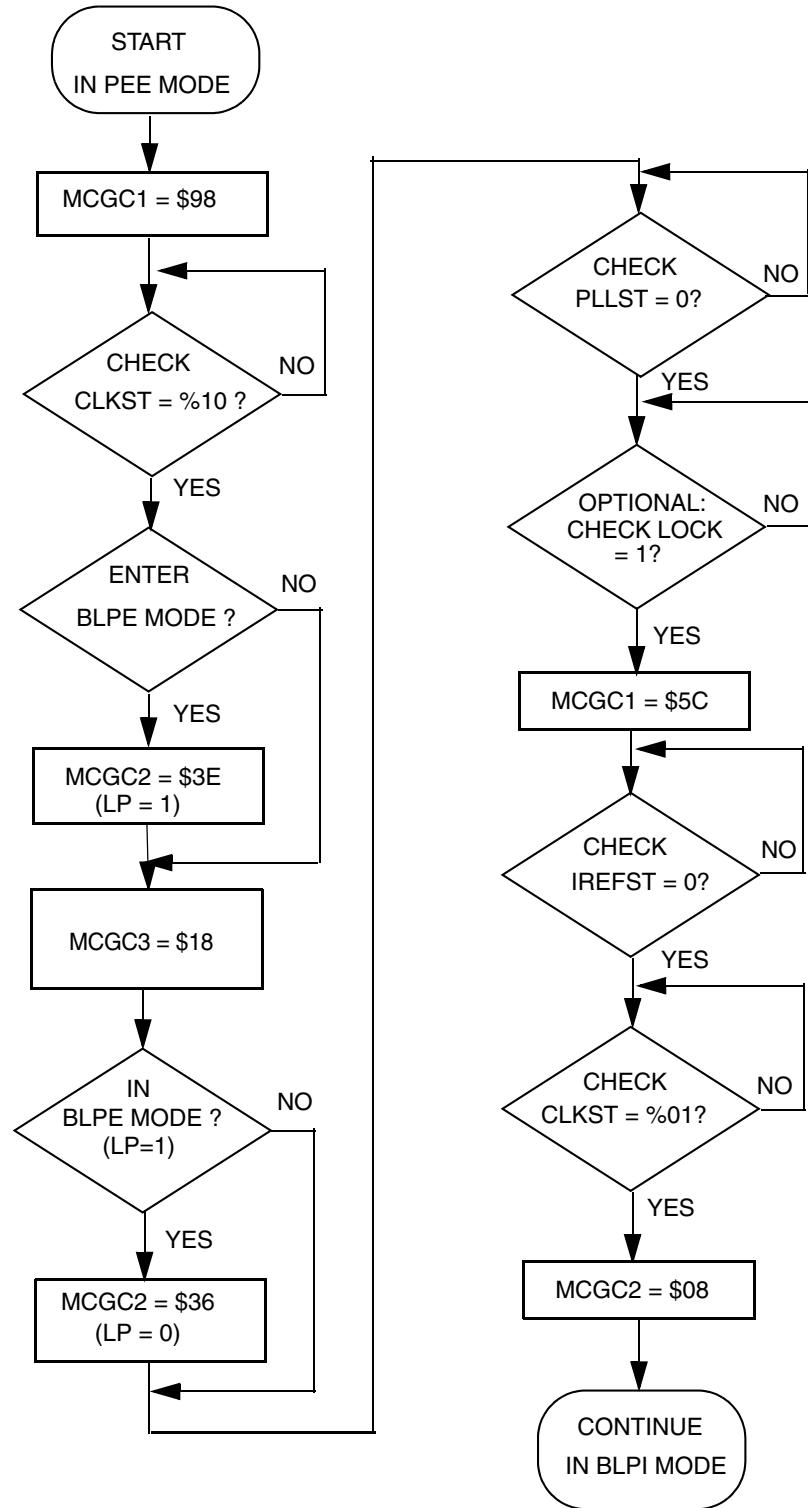


Figure 7-11. Flowchart of PEE to BLPI Mode Transition using an 8 MHz crystal

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

In this example, the MCG will move 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 will be described. Then a flowchart will be included which 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 is still enabled and running.
2. Next, FBI will transition 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 in order 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 \$01 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 will switch back to 512. Then loop again until the LOCK bit is set.

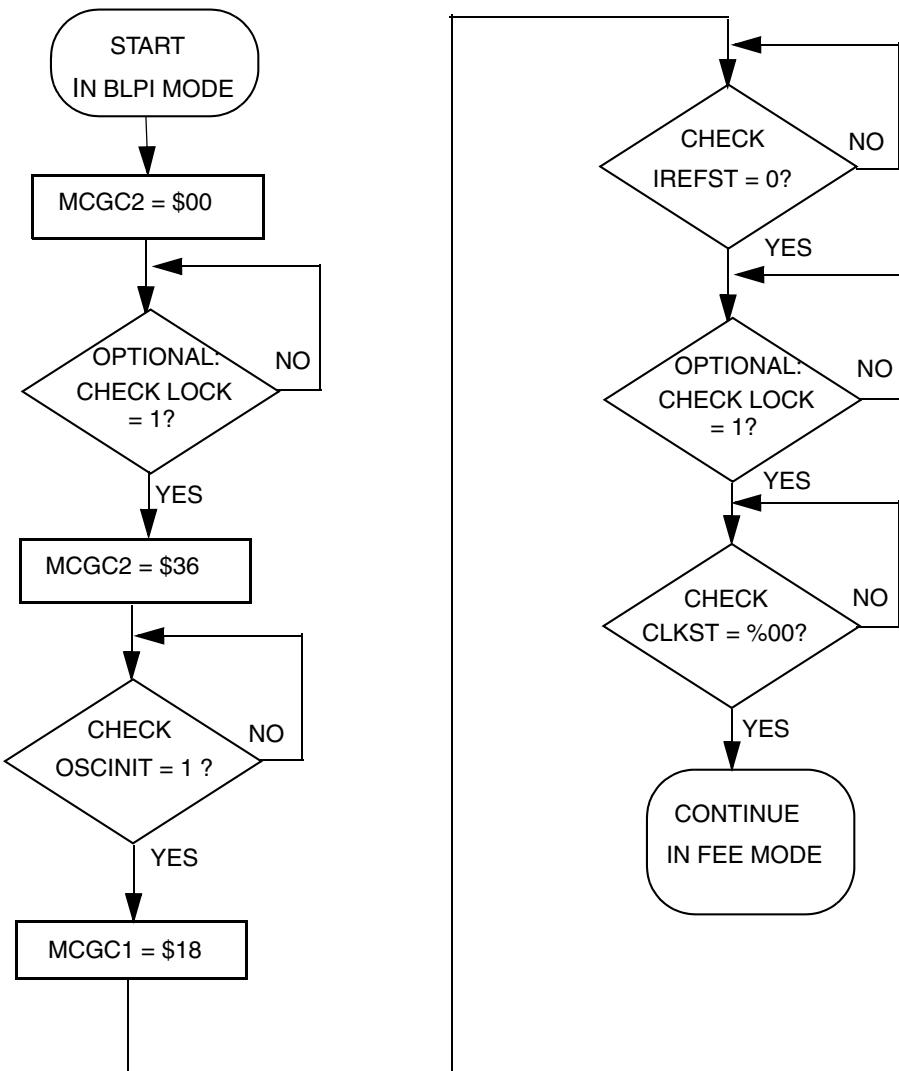


Figure 7-12. Flowchart of BLPI to FEE Mode Transition using an 8 MHz crystal

### 7.5.4 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. We will 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 will decrease the frequency and smaller values will increase 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: Internal Reference Clock Trim later in this section, this approach will be 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.

#### 7.5.4.1   **Example 4: Internal Reference Clock Trim**

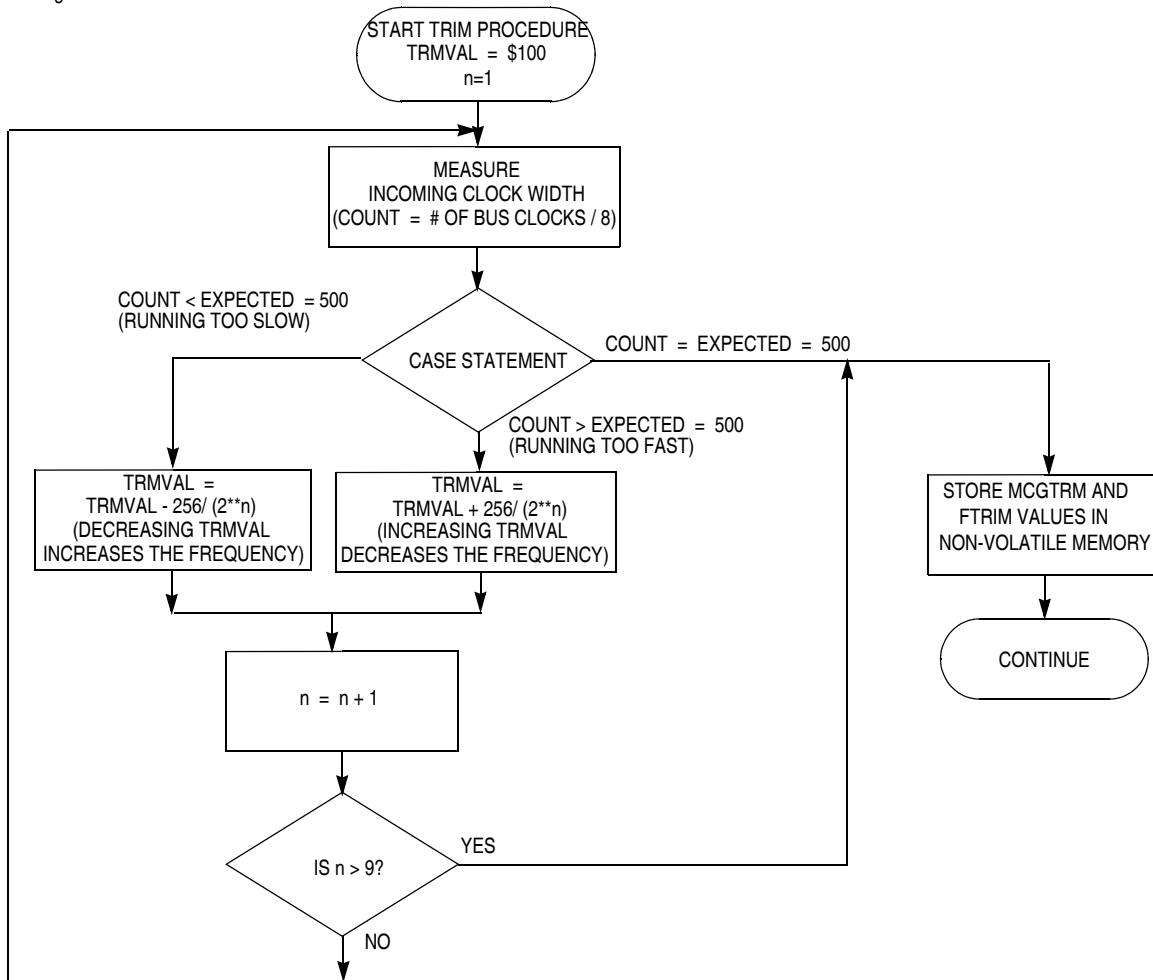
For applications that require a user specified tight frequency tolerance, a trimming procedure is provided that will allow a very accurate internal clock source. This section outlines one example of trimming the internal oscillator. Many other possible trimming procedures are valid and can be used.

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

## Multipurpose Clock Generator (MCGV3)

Initial conditions:

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



**Figure 7-13. 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 7-13](#) 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 will prevent accidental overshoot of the maximum clock frequency.

# 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 + 39 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.

4. 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 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](#).

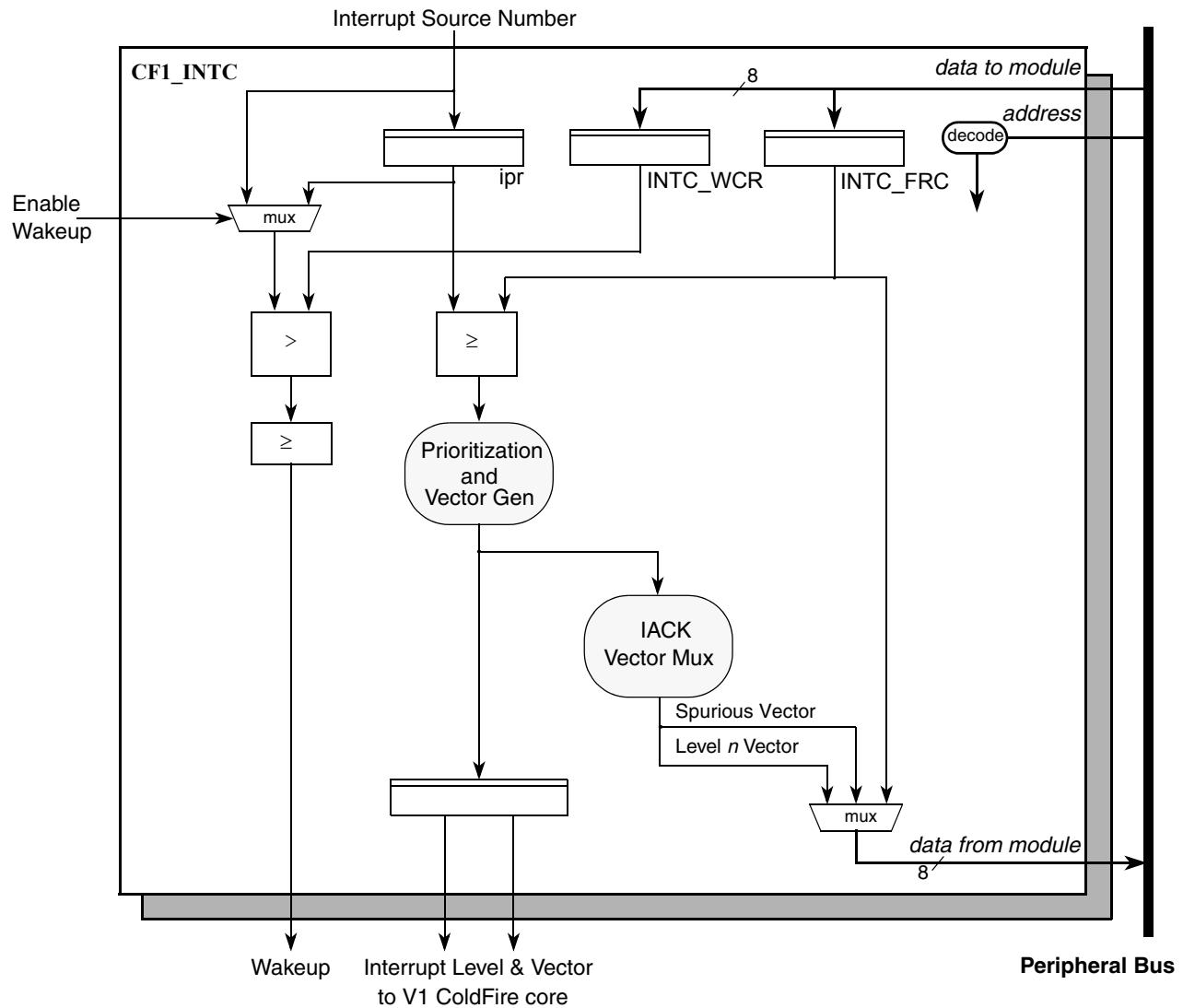


Figure 8-1. CF1\_INTC Block Diagram

### 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 35 peripheral I/O interrupt requests plus seven software (one per level) interrupt requests
- Fixed association between interrupt request source and level plus priority
  - 35 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 and stop modes

### 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 or stop 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 8-2](#).

**Table 8-2. CF1\_INTC Memory Map**

Offset Address	Register Name	Register Description	Width (bits)	Access	Reset Value	Section/ Page
0x10	INTC_FRC	CF1_INTC Force Interrupt Register	8	R/W	0x00	<a href="#">8.3.1/8-6</a>
0x18	INTC_PL6P7	CF1_INTC Programmable Level 6, Priority 7	8	R/W	0x00	<a href="#">8.3.2/8-7</a>
0x19	INTC_PL6P6	CF1_INTC Programmable Level 6, Priority 6	8	R/W	0x00	<a href="#">8.3.2/8-7</a>
0x1B	INTC_WCR	CF1_INTC Wakeup Control Register	8	R/W	0x80	<a href="#">8.3.3/8-8</a>
0x1E	INTC_SFRC	CF1_INTC Set Interrupt Force Register	8	Write	—	<a href="#">8.3.4/8-9</a>
0x1F	INTC_CFRC	CF1_INTC Clear Interrupt Force Register	8	Write	—	<a href="#">8.3.5/8-10</a>
0x20	INTC_SWIACK	CF1_INTC Software Interrupt Acknowledge	8	Read	0x00	<a href="#">8.3.6/8-11</a>
0x24	INTC_LVL1IACK	CF1_INTC Level 1 Interrupt Acknowledge	8	Read	0x18	<a href="#">8.3.6/8-11</a>
0x28	INTC_LVL2IACK	CF1_INTC Level 2 Interrupt Acknowledge	8	Read	0x18	<a href="#">8.3.6/8-11</a>
0x2C	INTC_LVL3IACK	CF1_INTC Level 3 Interrupt Acknowledge	8	Read	0x18	<a href="#">8.3.6/8-11</a>
0x30	INTC_LVL4IACK	CF1_INTC Level 4 Interrupt Acknowledge	8	Read	0x18	<a href="#">8.3.6/8-11</a>
0x34	INTC_LVL5IACK	CF1_INTC Level 5 Interrupt Acknowledge	8	Read	0x18	<a href="#">8.3.6/8-11</a>
0x38	INTC_LVL6IACK	CF1_INTC Level 6 Interrupt Acknowledge	8	Read	0x18	<a href="#">8.3.6/8-11</a>
0x3C	INTC_LVL7IACK	CF1_INTC Level 7 Interrupt Acknowledge	8	Read	0x18	<a href="#">8.3.6/8-11</a>

### 8.3.1 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.

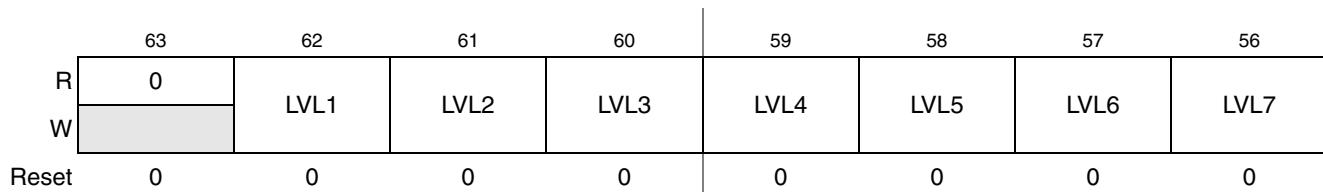
The INTC\_FRC register may be modified directly using a read-modify-write sequence or through a simple write operation using the set/clear force interrupt registers (INTC\_SFRC, INTC\_CFRC).

#### NOTE

Take special notice of the bit numbers within this register, 63–56. This is for compatibility with other ColdFire interrupt controllers.

Offset: CF1\_INTC\_BASE + 0x10 (INTC\_FRC)

Access: Read/Write

**Figure 8-2. Force Interrupt Register (INTC\_FRC)****Table 8-3. 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.

### 8.3.2 INTC Programmable Level 6, Priority {7,6} Registers (INTC\_PL6P{7,6})

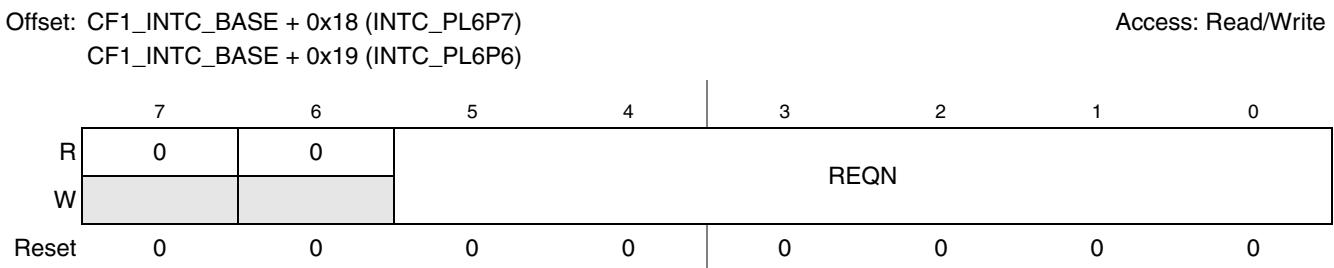
The level seven interrupt requests cannot have their levels reassigned. However, any of the remaining peripheral interrupt requests can be reassigned as the highest priority maskable requests using these two registers (INTC\_PL6P7 and INTC\_PL6P6). The vector number associated with the interrupt requests 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.”](#)



**Figure 8-3. Programmable Level 6, Priority {7,6} Registers (INTC\_PL6P{7,6})**

**Table 8-4. 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). <b>Note:</b> The value must be in a valid interrupt number. Unused or reserved interrupt numbers are ignored.

### 8.3.3 INTC Wakeup Control Register (INTC\_WCR)

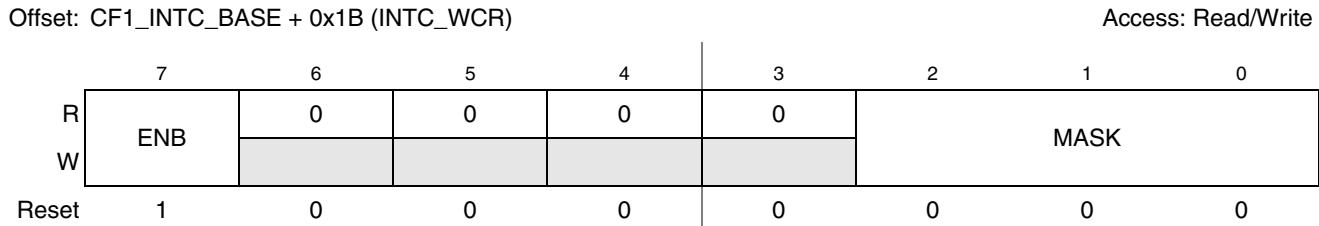
The interrupt controller provides a combinatorial logic path to generate a special wakeup signal to exit from the wait or stop modes. The INTC\_WCR register defines wakeup condition for interrupt recognition during wait and stop modes. This mode of operation works as follows:

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 or stop 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 or stop 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.

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 wakeup signal is defined as:

```
wakeup = INTC_WCR[ENB] & (level of any asserted int request > INTC_WCR[MASK])
```



**Figure 8-4. Wakeup Control Register (INTC\_WCR)**

**Table 8-5. 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 or stop 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 wakeup signal to the clock generation logic.

### 8.3.4 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 8-5. INTC\_SFRC Register**

**Table 8-6. INTC\_SFRC Field Descriptions**

Field	Description
7–6	Reserved, must be cleared.
5–0 SET	<p>For data values within the 56–62 range, the corresponding bit in the INTC_FRC register is set, as defined below.</p> <ul style="list-style-type: none"> <li>0x38 Bit 56, INTC_FRC[LVL7] is set</li> <li>0x39 Bit 57, INTC_FRC[LVL6] is set</li> <li>0x3A Bit 58, INTC_FRC[LVL5] is set</li> <li>0x3B Bit 59, INTC_FRC[LVL4] is set</li> <li>0x3C Bit 60, INTC_FRC[LVL3] is set</li> <li>0x3D Bit 61, INTC_FRC[LVL2] is set</li> <li>0x3E Bit 62, INTC_FRC[LVL1] is set</li> </ul> <p><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.</p>

### 8.3.5 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.

Offset: CF1\_INTC\_BASE + 0x1F (INTC\_CFRC)

Access: Write-only



**Figure 8-6. INTC CFRC Register**

**Table 8-7. 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.

### 8.3.6 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					3	2	1	0
W									
SWIACK Reset	0	0	0	0	0	0	0	0	0
LVL <i>n</i> IACK Reset	0	0	0	1	1	0	0	0	0

**Table 8-8. Software and Level-*n* IACK Registers (INTC\_SWIACK, INTC\_LVL*n*IACK)**

**Table 8-9. 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 8.3.2, “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 18 = vector 82 = level 3, priority 6
- sci1\_tx = interrupt source 19 = vector 83 = level 43, 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 18 (0x12), remaps sci1\_rx as level 6, priority 7.
- Setting INTC\_PL6P6 to 19 (0x13), 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 8-7](#).

```

        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 8-7. 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

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

This section explains software controls related to parallel input/output (I/O) and pin control. The MCF51JM128 series MCUs have up to nine parallel I/O ports that includes a total of 70 I/O pins and one input-only pin. See [Chapter 2, “Pins and Connections,”](#) for more information about pin assignments and external hardware considerations of these pins.

In addition to standard I/O port functionality, Ports C and E have set/clear/toggle functions integrated as part of the ColdFire core itself to improve edge resolution on those pins. See [Section 9.3, “V1 ColdFire Rapid GPIO Functionality,”](#) and [Chapter 10, “Rapid GPIO \(RGPIO\),”](#) for additional details.

Many port pins are shared with on-chip peripherals such as timer systems, communication systems, or keyboard interrupts as shown in [Figure 9-1](#). The peripheral modules have priority over the general-purpose I/O functions so that when a peripheral is enabled, the I/O functions associated with the shared pins may be disabled.

After reset, the shared peripheral functions are disabled and the pins are configured as inputs ( $\text{PTxD}\text{D}_n = 0$ ). The pin control functions for each pin are configured as follows: slew rate control enabled ( $\text{PTxSE}_n = 1$ ), low drive strength selected ( $\text{PTxD}\text{S}_n = 0$ ), and internal pull-ups disabled ( $\text{PTxPE}_n = 0$ ).

### NOTE

Not all general-purpose I/O pins are available on all packages. To avoid extra current drain from floating input pins, the user’s reset initialization routine in the application program must enable on-chip pull-up devices or change the direction of unconnected pins to outputs so the pins do not float.

### 9.1 Port Data and Data Direction

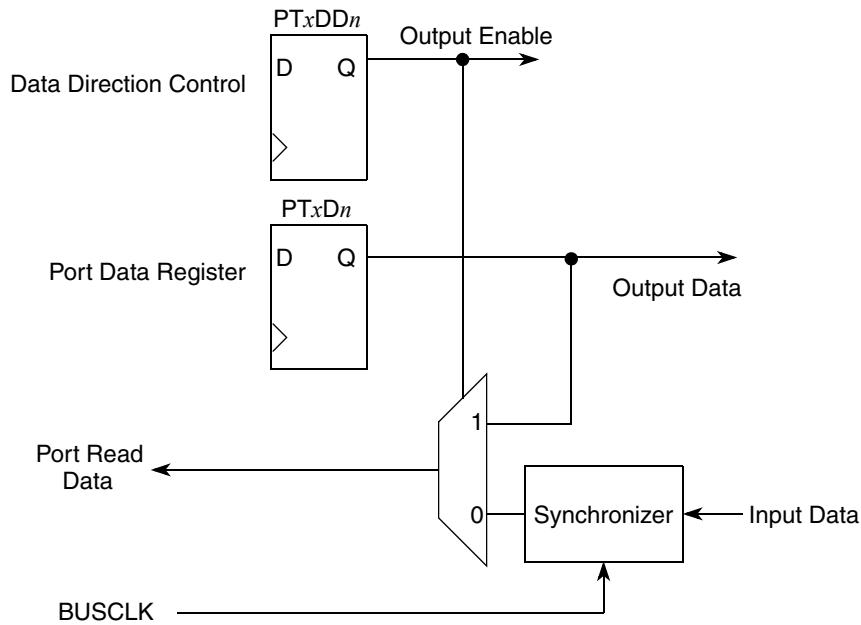
Reading and writing of parallel I/Os are performed through the port data registers. The direction, input or output, is controlled through the port data direction registers. The parallel I/O port function for an individual pin is illustrated in the block diagram shown in [Figure 9-1](#).

The data direction control bit ( $\text{PTxD}\text{D}_n$ ) 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 ( $\text{PTxD}\text{D}_n = 0$ ) and the input buffer is disabled. In general, when a pin is shared with an alternate digital function and an analog function, the analog function has priority such that if the digital and analog functions are enabled, the analog function controls the pin.

It is good programming practice to write to the port data register before changing the direction of a port pin to become an output. This ensures that the pin is not driven momentarily with an old data value that happened to be in the port data register.



**Figure 9-1. Classic Parallel I/O Block Diagram**

## 9.2 Pull-up, Slew Rate, and Drive Strength

A set of high page registers are used to control pull-ups, slew rate, and drive strength 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, but operate independently of the parallel I/O registers.

### 9.2.1 Port Internal Pull-up Enable

An internal pull-up device can be enabled for each port pin by setting the corresponding bit in the pull-up enable register (PTxPEn). The pull-up device is disabled if the pin is configured as an output by the parallel I/O control logic or any shared peripheral function regardless of the state of the corresponding pull-up enable register bit. The pull-up device is also disabled if the pin is controlled by an analog function.

### 9.2.2 Port Slew Rate Enable

Slew rate control can be enabled for each port pin by setting the corresponding bit in the slew rate control register (PTxSEN). When enabled, slew control limits the rate at which an output can transition to reduce EMC emissions. Slew rate control has no effect on pins that are configured as inputs.

### 9.2.3 Port Drive Strength Select

An output pin can be selected to have high output drive strength by setting the corresponding bit in the drive strength select register (PTxDSn). 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, the user must 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.

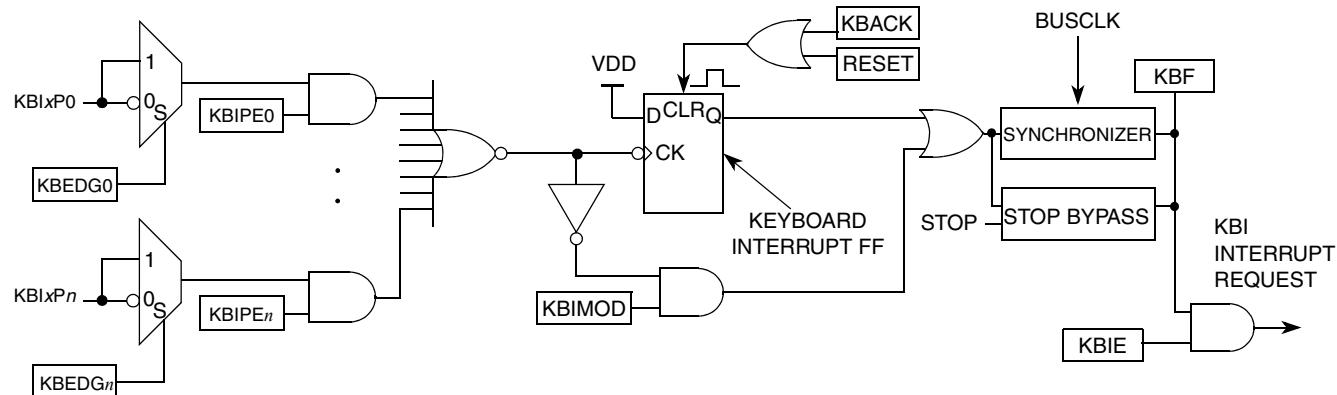
## 9.3 V1 ColdFire Rapid GPIO Functionality

The V1 ColdFire core is capable of performing 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 can be programmed to take priority on ports C and E.

This functionality is further defined in [Chapter 10, “Rapid GPIO \(RGPIO\).](#)

## 9.4 Keyboard Interrupts

Some port B, some port D, and some port G pins can be configured as keyboard interrupt inputs and as an external means of waking the MCU from stop or wait low-power modes. The block diagram for each keyboard interrupt logic is shown [Figure 9-2.](#)



**Figure 9-2. Port Interrupt Block Diagram**

Writing to the KBIPE $n$  bits in the keyboard  $x$  interrupt pin enable register (KBIxPE) independently enables or disables each port pin. Each port can be configured as edge sensitive or edge and level sensitive based on the KBIMOD bit in the keyboard interrupt status and control register (KBIxSC). Edge sensitivity can be software programmed to be falling or rising; the level can be low or high. The polarity of the edge or edge and level sensitivity is selected using the KBEDG $n$  bits in the keyboard interrupt edge select register (KBIxES).

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.

#### 9.4.1 Edge Only Sensitivity

A valid edge on an enabled port pin sets KBF in KBIxSC. If the KBIxSC[KBIE] bit is set, an interrupt request is presented to the CPU. Clearing of KBF is accomplished by writing a 1 to KBIxSC[KBACK].

#### 9.4.2 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 presented to the CPU. Clearing of KBF is accomplished by writing a 1 to KBIxSC[KBACK], 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.

#### 9.4.3 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).

#### 9.4.4 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, the user should do the following:

1. Mask interrupts by clearing KBIxSC[KBIE].
2. Select the pin polarity by setting the appropriate KBIxES[KBEDG $n$ ] bits.
3. If using internal pull-up/pull-down device, configure the associated pull enable bits in KBIxPE.
4. Enable the interrupt pins by setting the appropriate KBIxPE[KBIPE $n$ ] bits.
5. Write to KBIxSC[KBACK] to clear any false interrupts.
6. Set KBIxSC[KBIE] to enable interrupts.

### 9.5 Pin Behavior in Stop Modes

Pin behavior following execution of a STOP instruction depends on the stop mode 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 STOP instruction is executed to place the MCU in Stop2 mode.  
Upon recovery from Stop2 mode, before accessing any I/O, the user should 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. The user must 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. Upon recovery, normal I/O function is available to the user.

## 9.6 Parallel I/O, Keyboard Interrupt, and Pin Control Registers

This section provides information about the registers associated with the parallel I/O ports. The data and data direction registers and the keyboard interrupt registers are located in page zero of the memory map. The pull-up, slew rate, drive strength, and interrupt control registers are located in the high page section of the memory map.

Refer to tables in [Chapter 4, “Memory,”](#) for the absolute address assignments for all parallel I/O and their pin control registers. This section refers to registers and control bits only by their names. A Freescale Semiconductor-provided equate or header file normally is used to translate these names into the appropriate absolute addresses.

### 9.6.1 Port A Registers

Port A is an 8bit Input Output Port and also shares its pins with the Rapid General Purpose Input Output (RGPIO) module. On reset, the pins default to the input state and under the I/O control. Enabling any of the RGPIO bits via the RGPIO\_ENB register takes priority over the normal I/O control.

Port A is controlled by the registers listed below.

#### 9.6.1.1 Port A Data Register (PTAD)

	7	6	5	4		3	2	1	0
R W	PTAD7	PTAD6	PTAD5	PTAD4	PTAD3	PTAD2	PTAD1	PTAD0	
Reset:	0	0	0	0	0	0	0	0	0

Figure 9-3. Port A Data Register (PTAD)

Table 9-1. PTAD Register Field Descriptions

Field	Description
7–0 PTAD $n$	<b>Port A Data Register Bits</b> — For port A pins that are inputs, reads return the logic level on the pin. For port A 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 A pins that are configured as outputs, the logic level is driven out the corresponding MCU pin. Reset forces PTAD to all 0s, but these 0s are not driven out the corresponding pins because reset also configures all port pins as high-impedance inputs with pull-ups/pull-downs disabled.

### 9.6.1.2 Port A Data Direction Register (PTADD)

	7	6	5	4		3	2	1	0
R W	PTADD7	PTADD6	PTADD5	PTADD4		PTADD3	PTADD2	PTADD1	PTADD0
Reset:	0	0	0	0		0	0	0	0

Figure 9-4. Port A Data Direction Register (PTADD)

Table 9-2. PTADD Register Field Descriptions

Field	Description
7–0 PTADD $n$	<b>Data Direction for Port A Bits</b> — These read/write bits control the direction of port A pins and what is read for PTAD reads. 0 Input (output driver disabled) and reads return the pin value. 1 Output driver enabled for port A bit $n$ and PTAD reads return the contents of PTAD $n$ .

### 9.6.1.3 Port A Pull Enable Register (PTAPE)

	7	6	5	4		3	2	1	0
R W	PTAPE7	PTAPE6	PTAPE5	PTAPE4		PTAPE3	PTAPE2	PTAPE1	PTAPE0
Reset:	0	0	0	0		0	0	0	0

Figure 9-5. Internal Pull Enable for Port A Register (PTAPE)

Table 9-3. PTAPE Register Field Descriptions

Field	Description
7–0 PTAPE $n$	<b>Internal Pull Enable for Port A Bits</b> — Each of these control bits determines if the internal pull-up device is enabled for the associated PTA pin. For port A 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 A bit $n$ . 1 Internal pull-up device enabled for port A bit $n$ .

### 9.6.1.4 Port A Slew Rate Enable Register (PTASE)

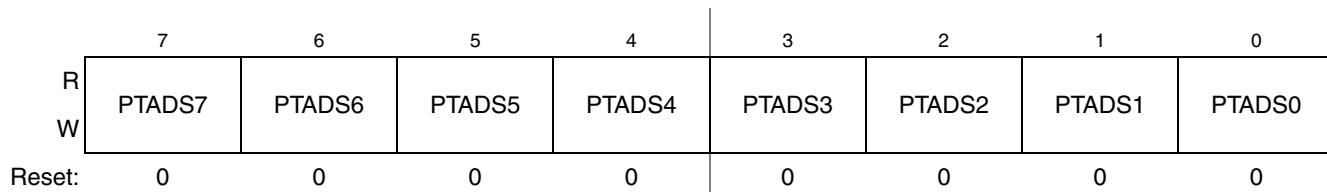
	7	6	5	4		3	2	1	0
R W	PTASE7	PTASE6	PTASE5	PTASE4		PTASE3	PTASE2	PTASE1	PTASE0
Reset:	1	1	1	1		1	1	1	1

Figure 9-6. Slew Rate Enable for Port A Register (PTASE)

**Table 9-4. PTASE Register Field Descriptions**

Field	Description
7–0 PTASE $n$	<b>Output Slew Rate Enable for Port A Bits</b> — Each of these control bits determines if the output slew rate control is enabled for the associated PTA pin. For port A pins that are configured as inputs, these bits have no effect. 0 Output slew rate control disabled for port A bit $n$ . 1 Output slew rate control enabled for port A bit $n$ .

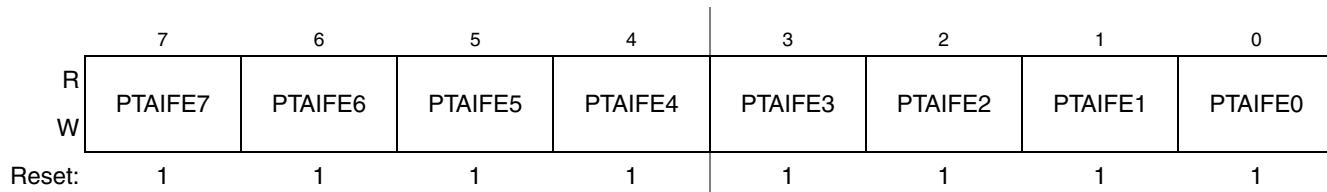
### 9.6.1.5 Port A Drive Strength Selection Register (PTADS)

**Figure 9-7. Drive Strength Selection for Port A Register (PTADS)****Table 9-5. PTADS Register Field Descriptions**

Field	Description
7–0 PTADS $n$	<b>Output Drive Strength Selection for Port A Bits</b> — Each of these control bits selects between low and high output drive for the associated PTA pin. For port A pins that are configured as inputs, these bits have no effect. 0 Low output drive strength selected for port A bit $n$ . 1 High output drive strength selected for port A bit $n$ .

### 9.6.1.6 Port A Input Filter Enable Register (PTAIFE)

The Port A pin incorporates an optional input low pass filter. The filter is enabled during and after reset by setting the associated PTAIFE bit. The filter is disabled through software control by clearing the associated PTAIFE bit.

**Figure 9-8. Port A Input Filter Enable Register (PTAIFE)****Table 9-6. PTAIFE Field Descriptions**

Field	Description
7–0 PTAIFE $n$	<b>Port A Input Filter Enable</b> — Input low-pass filter enable control bits for PTA pins. 0 Input filter disabled 1 Input filter enabled

## 9.6.2 Port B Registers

Port B is an 8bit Input/Output Port and also shares its pins with the SPI2,Keyboard Interrupt and the ADC peripherals. On reset, the pins default to the input state and under the I/O control. Enabling any of the digital peripherals SPI2, Keyboard Interrupt takes priority control over the pins. Enabling the ADC pins via the APCTL1 register takes priority control over the I/O control and digital peripheral control.

Port B is controlled by the registers listed below.

### 9.6.2.1 Port B Data Register (PTBD)

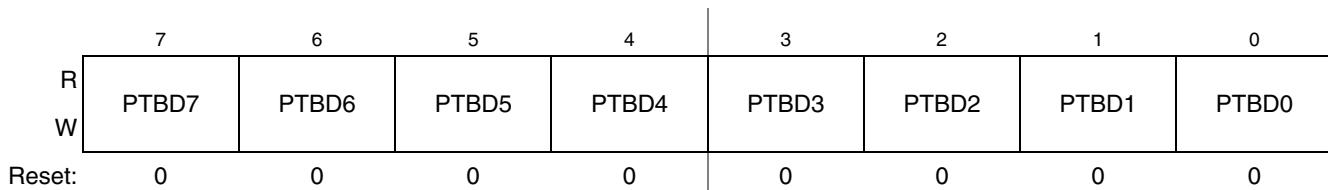


Figure 9-9. Port B Data Register (PTBD)

Table 9-7. PTBD Register Field Descriptions

Field	Description
7–0 PTBD $n$	<b>Port B Data Register Bits</b> — For port B pins that are inputs, reads return the logic level on the pin. For port B 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 B pins that are configured as outputs, the logic level is driven out the corresponding MCU pin. Reset forces PTBD to all 0s, but these 0s are not driven out the corresponding pins because reset also configures all port pins as high-impedance inputs with pull-ups/pull-downs disabled.

### 9.6.2.2 Port B Data Direction Register (PTBDD)

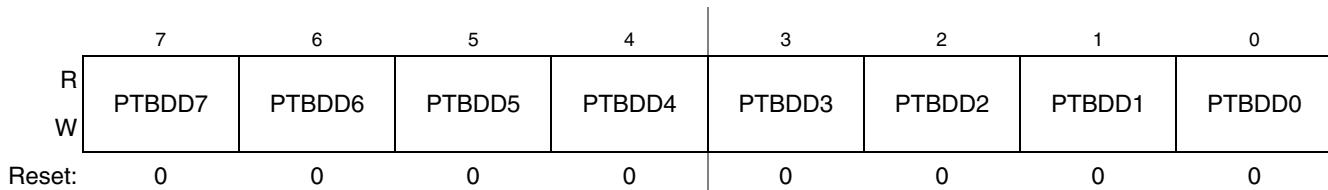


Figure 9-10. Port B Data Direction Register (PTBDD)

Table 9-8. PTBDD Register Field Descriptions

Field	Description
7–0 PTBDD $n$	<b>Data Direction for Port B Bits</b> — These read/write bits control the direction of port B pins and what is read for PTBD reads. 0 Input (output driver disabled) and reads return the pin value. 1 Output driver enabled for port B bit $n$ and PTBD reads return the contents of PTBD $n$ .

### 9.6.2.3 Port B Pull Enable Register (PTBPE)

	7	6	5	4	3	2	1	0
R W	PTBPE7	PTBPE6	PTBPE5	PTBPE4	PTBPE3	PTBPE2	PTBPE1	PTBPE0
Reset:	0	0	0	0	0	0	0	0

Figure 9-11. Internal Pull Enable for Port B Register (PTBPE)

Table 9-9. PTBPE Register Field Descriptions

Field	Description
7–0 PTBPE $n$	<b>Internal Pull Enable for Port B Bits</b> — Each of these control bits determines if the internal pull-up device is enabled for the associated PTB pin. For port B 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 B bit $n$ . 1 Internal pull-up device enabled for port B bit $n$ .

### 9.6.2.4 Port B Slew Rate Enable Register (PTBSE)

	7	6	5	4	3	2	1	0
R W	PTBSE7	PTBSE6	PTBSE5	PTBSE4	PTBSE3	PTBSE2	PTBSE1	PTBSE0
Reset:	1	1	1	1	1	1	1	1

Figure 9-12. Slew Rate Enable for Port B Register (PTBSE)

Table 9-10. PTBSE Register Field Descriptions

Field	Description
7–0 PTBSE $n$	<b>Output Slew Rate Enable for Port B Bits</b> — Each of these control bits determines if the output slew rate control is enabled for the associated PTB pin. For port B pins that are configured as inputs, these bits have no effect. 0 Output slew rate control disabled for port B bit $n$ . 1 Output slew rate control enabled for port B bit $n$ .

### 9.6.2.5 Port B Drive Strength Selection Register (PTBDS)

	7	6	5	4	3	2	1	0
R W	PTBDS7	PTBDS6	PTBDS5	PTBDS4	PTBDS3	PTBDS2	PTBDS1	PTBDS0
Reset:	0	0	0	0	0	0	0	0

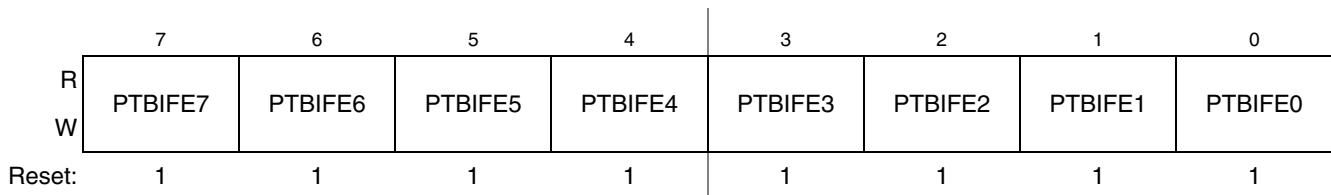
Figure 9-13. Drive Strength Selection for Port B Register (PTBDS)

**Table 9-11. PTBDS Register Field Descriptions**

Field	Description
7–0 PTBDS $n$	<b>Output Drive Strength Selection for Port B Bits</b> — Each of these control bits selects between low and high output drive for the associated PTB pin. For port B pins that are configured as inputs, these bits have no effect. 0 Low output drive strength selected for port B bit $n$ . 1 High output drive strength selected for port B bit $n$ .

### 9.6.2.6 Port B Input Filter Enable Register (PTBIFE)

The Port B pin incorporates an optional input low pass filter. The filter is enabled during and after reset by setting the associated PTBIFE bit. The filter is disabled through software control by clearing the associated PTBIFE bit.

**Figure 9-14. Port B Input Filter Enable Register (PTBIFE)****Table 9-12. PTBIFE Field Descriptions**

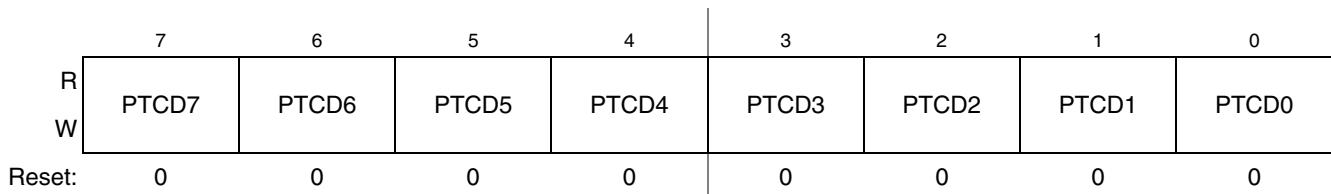
Field	Description
7–0 PTBIFE $n$	<b>Port B Input Filter Enable</b> — Input low-pass filter enable control bits for PTB pins. 0 Input filter disabled 1 Input filter enabled

### 9.6.3 Port C Registers

Port C is an 8bit Input/Output Port and also shares its pins with the CMT, IIC1,SCI2 and the msCAN peripherals. On reset, the pins default to the input state and under the I/O control. Enabling any of the digital peripherals CMT, IIC1,SCI2 and the msCAN takes priority control over the pins.

Port C is controlled by the registers listed below.

#### 9.6.3.1 Port C Data Register (PTCD)

**Figure 9-15. Port C Data Register (PTCD)**

**Table 9-13. PTCD Register Field Descriptions**

Field	Description
7–0 PTCD $n$	<b>Port C Data Register Bits</b> — For port C pins that are inputs, reads return the logic level on the pin. For port C 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 C pins that are configured as outputs, the logic level is driven out the corresponding MCU pin. Reset forces PTCD to all 0s, but these 0s are not driven out the corresponding pins because reset also configures all port pins as high-impedance inputs with pull-ups disabled.

### 9.6.3.2 Port C Data Direction Register (PTCDD)

	7	6	5	4		3	2	1	0
R	PTCDD7	PTCDD6	PTCDD5	PTCDD4	PTCDD3	PTCDD2	PTCDD1	PTCDD0	
W	0	0	0	0	0	0	0	0	0

Reset: 0 0 0 0 0 0 0 0 0 0

**Figure 9-16. Port C Data Direction Register (PTCDD)****Table 9-14. PTCDD Register Field Descriptions**

Field	Description
7–0 PTCDD $n$	<b>Data Direction for Port C Bits</b> — These read/write bits control the direction of port C pins and what is read for PTCD reads. 0 Input (output driver disabled) and reads return the pin value. 1 Output driver enabled for port C bit $n$ and PTCD reads return the contents of PTCD $n$ .

### 9.6.3.3 Port C Pull Enable Register (PTCPE)

	7	6	5	4		3	2	1	0
R	PTCPE7	PTCPE6	PTCPE5	PTCPE4	PTCPE3	PTCPE2	PTCPE1	PTCPE0	
W	0	0	0	0	0	0	0	0	0

Reset: 0 0 0 0 0 0 0 0 0 0

**Figure 9-17. Internal Pull Enable for Port C Register (PTCPE)****Table 9-15. PTCPE Register Field Descriptions**

Field	Description
7–0 PTCPE $n$	<b>Internal Pull Enable for Port C Bits</b> — Each of these control bits determines if the internal pull-up device is enabled for the associated PTC pin. For port C 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 C bit $n$ . 1 Internal pull-up device enabled for port C bit $n$ .

### 9.6.3.4 Port C Slew Rate Enable Register (PTCSE)

	7	6	5	4		3	2	1	0
R W	PTCSE7	PTCSE6	PTCSE5	PTCSE4		PTCSE3	PTCSE2	PTCSE1	PTCSE0
Reset:	1	1	1	1		1	1	1	1

Figure 9-18. Slew Rate Enable for Port C Register (PTCSE)

Table 9-16. PTCSE Register Field Descriptions

Field	Description
7–0 PTCSE $n$	<b>Output Slew Rate Enable for Port C Bits</b> — Each of these control bits determines if the output slew rate control is enabled for the associated PTC pin. For port C pins that are configured as inputs, these bits have no effect. 0 Output slew rate control disabled for port C bit $n$ . 1 Output slew rate control enabled for port C bit $n$ .

### 9.6.3.5 Port C Drive Strength Selection Register (PTCDS)

	7	6	5	4		3	2	1	0
R W	PTCDS7	PTCDS6	PTCDS5	PTCDS4		PTCDS3	PTCDS2	PTCDS1	PTCDS0
Reset:	0	0	0	0		0	0	0	0

Figure 9-19. Drive Strength Selection for Port C Register (PTCDS)

Table 9-17. PTCDS Register Field Descriptions

Field	Description
7–0 PTCDS $n$	<b>Output Drive Strength Selection for Port C Bits</b> — Each of these control bits selects between low and high output drive for the associated PTC pin. For port C pins that are configured as inputs, these bits have no effect. 0 Low output drive strength selected for port C bit $n$ . 1 High output drive strength selected for port C bit $n$ .

### 9.6.3.6 Port C Input Filter Enable Register (PTCIFE)

The Port C pin incorporates an optional input low pass filter. The filter is enabled during and after reset by setting the associated PTCIFE bit. The filter is disabled through software control by clearing the associated PTCIFE bit.

	7	6	5	4		3	2	1	0
R W	PTCIFE7	PTCIFE6	PTCIFE5	PTCIFE4		PTCIFE3	PTCIFE2	PTCIFE1	PTCIFE0
Reset:	1	1	1	1		1	1	1	1

Figure 9-20. Port C Input Filter Enable Register (PTCIFE)

**Table 9-18. PTCIFE Field Descriptions**

Field	Description
7–0 PTCIFE <sub>n</sub>	<b>Port C Input Filter Enable</b> — Input low-pass filter enable control bits for PTC pins. 0 Input filter disabled 1 Input filter enabled

## 9.6.4 Port D Registers

Port D is an 8-bit input/output port that shares its pins with the Keyboard Interrupt, ACMP and the ADC peripherals. After reset, the pins default to the input state under the I/O control. Enabling any of the digital peripherals, Keyboard Interrupt, and the ADC takes priority control over the pins. Enabling the ACMP takes priority control over the I/O control, ADC, and digital peripheral control.

Port D is controlled by the registers listed below.

### 9.6.4.1 Port D Data Register (PTDD)

	7	6	5	4		3	2	1	0
R W	PTDD7	PTDD6	PTDD5	PTDD4	PTDD3	PTDD2	PTDD1	PTDD0	
Reset:	0	0	0	0	0	0	0	0	0

**Figure 9-21. Port D Data Register (PTDD)****Table 9-19. PTDD Register Field Descriptions**

Field	Description
7–0 PTDD <sub>n</sub>	<b>Port D Data Register Bits</b> — For port D pins that are inputs, reads return the logic level on the pin. For port D 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 D pins that are configured as outputs, the logic level is driven out the corresponding MCU pin. Reset forces PTDD to all 0s, but these 0s are not driven out the corresponding pins because reset also configures all port pins as high-impedance inputs with pull-ups/pull-downs disabled.

### 9.6.4.2 Port D Data Direction Register (PTDDD)

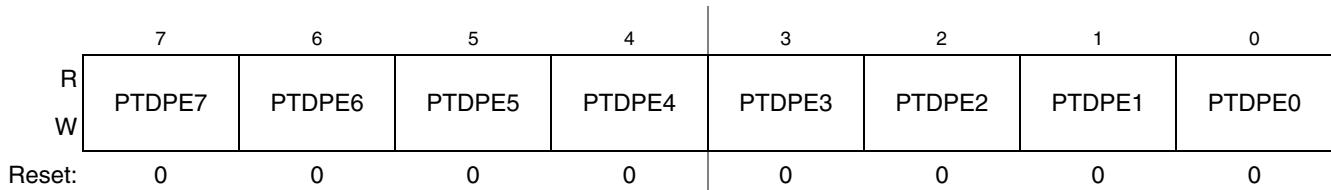
	7	6	5	4		3	2	1	0
R W	PTDDD7	PTDDD6	PTDDD5	PTDDD4	PTDDD3	PTDDD2	PTDDD1	PTDDD0	
Reset:	0	0	0	0	0	0	0	0	0

**Figure 9-22. Port D Data Direction Register (PTDDD)**

**Table 9-20. PTDDD Register Field Descriptions**

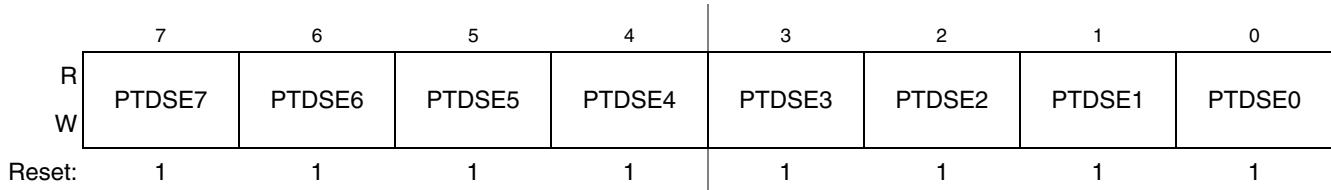
Field	Description
7–0 PTDDD $n$	<b>Data Direction for Port D Bits</b> — These read/write bits control the direction of port D pins and what is read for PTDD reads. 0 Input (output driver disabled) and reads return the pin value. 1 Output driver enabled for port D bit $n$ and PTDD reads return the contents of PTDD $n$ .

### 9.6.4.3 Port D Pull Enable Register (PTDPE)

**Figure 9-23. Internal Pull Enable for Port D Register (PTDPE)****Table 9-21. PTDPE Register Field Descriptions**

Field	Description
7–0 PTDPE $n$	<b>Internal Pull Enable for Port D Bits</b> — Each of these control bits determines if the internal pull-up device is enabled for the associated PTD pin. For port D 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 D bit $n$ . 1 Internal pull-up device enabled for port D bit $n$ .

### 9.6.4.4 Port D Slew Rate Enable Register (PTDSE)

**Figure 9-24. Slew Rate Enable for Port D Register (PTDSE)****Table 9-22. PTDSE Register Field Descriptions**

Field	Description
7–0 PTDSE $n$	<b>Output Slew Rate Enable for Port D Bits</b> — Each of these control bits determines if the output slew rate control is enabled for the associated PTD pin. For port D pins that are configured as inputs, these bits have no effect. 0 Output slew rate control disabled for port D bit $n$ . 1 Output slew rate control enabled for port D bit $n$ .

### 9.6.4.5 Port D Drive Strength Selection Register (PTDDS)

	7	6	5	4		3	2	1	0
R W	PTDDS7	PTDDS6	PTDDS5	PTDDS4		PTDDS3	PTDDS2	PTDDS1	PTDDS0
Reset:	0	0	0	0		0	0	0	0

Figure 9-25. Drive Strength Selection for Port D Register (PTDDS)

Table 9-23. PTDDS Register Field Descriptions

Field	Description
7–0 PTDDSn	<b>Output Drive Strength Selection for Port D Bits</b> — Each of these control bits selects between low and high output drive for the associated PTD pin. For port D pins that are configured as inputs, these bits have no effect. 0 Low output drive strength selected for port D bit <i>n</i> . 1 High output drive strength selected for port D bit <i>n</i> .

### 9.6.4.6 Port D Input Filter Enable Register (PTDIFE)

The Port D pin incorporates an optional input low pass filter. The filter is enabled during and after reset by setting the associated PTDIFE bit. The filter is disabled through software control by clearing the associated PTDIFE bit.

	7	6	5	4		3	2	1	0
R W	PTDIFE7	PTDIFE6	PTDIFE5	PTDIFE4		PTDIFE3	PTDIFE2	PTDIFE1	PTDIFE0
Reset:	1	1	1	1		1	1	1	1

Figure 9-26. Port D Input Filter Enable Register (PTDIFE)

Table 9-24. PTDIFE Field Descriptions

Field	Description
7–0 PTDIFEn	<b>Port D Input Filter Enable</b> — Input low-pass filter enable control bits for PTD pins. 0 Input filter disabled 1 Input filter enabled

### 9.6.5 Port E Registers

Port E is an 8bit Input/Output Port and also shares its pins with the SPI1,SCI1 and the TP1 peripherals. On reset, the pins default to the input state and under the I/O control. Enabling any of the digital peripherals SPI1,SCI1 and the TP1 takes priority control over the pins.

Port E is controlled by the registers listed below.

### 9.6.5.1 Port E Data Register (PTED)

	7	6	5	4		3	2	1	0
R W	PTED7	PTED6	PTED5	PTED4		PTED3	PTED2	PTED1	PTED0
Reset:	0	0	0	0		0	0	0	0

Figure 9-27. Port E Data Register (PTED)

Table 9-25. PTED Register Field Descriptions

Field	Description
7–0 PTED $n$	<b>Port E Data Register Bits</b> — For port E pins that are inputs, reads return the logic level on the pin. For port E 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 E pins that are configured as outputs, the logic level is driven out the corresponding MCU pin. Reset forces PTED to all 0s, but these 0s are not driven out the corresponding pins because reset also configures all port pins as high-impedance inputs with pull-ups disabled.

### 9.6.5.2 Port E Data Direction Register (PTEDD)

	7	6	5	4		3	2	1	0
R W	PTEDD7	PTEDD6	PTEDD5	PTEDD4		PTEDD3	PTEDD2	PTEDD1	PTEDD0
Reset:	0	0	0	0		0	0	0	0

Figure 9-28. Port E Data Direction Register (PTEDD)

Table 9-26. PTEDD Register Field Descriptions

Field	Description
7–0 PTEDD $n$	<b>Data Direction for Port E Bits</b> — These read/write bits control the direction of port E pins and what is read for PTED reads. 0 Input (output driver disabled) and reads return the pin value. 1 Output driver enabled for port E bit $n$ and PTED reads return the contents of PTED $n$ .

### 9.6.5.3 Port E Pull Enable Register (PTEPE)

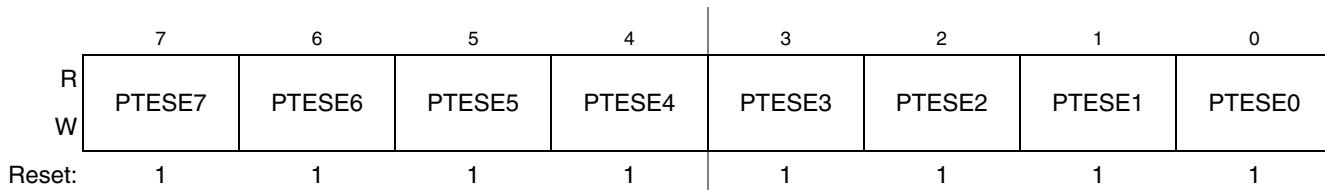
	7	6	5	4		3	2	1	0
R W	PTEPE7	PTEPE6	PTEPE5	PTEPE4		PTEPE3	PTEPE2	PTEPE1	PTEPE0
Reset:	0	0	0	0		0	0	0	0

Figure 9-29. Internal Pull Enable for Port E Register (PTEPE)

**Table 9-27. PTEPE Register Field Descriptions**

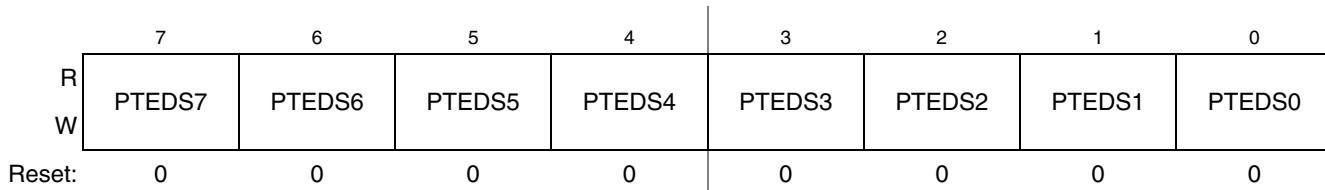
Field	Description
7–0 PTEPE $n$	<b>Internal Pull Enable for Port E Bits</b> — Each of these control bits determines if the internal Input filter device is enabled for the associated PTE pin. For port E pins that are configured as outputs, these bits have no effect and the internal pull devices are disabled. 0 Internal pull-up device disabled for port E bit $n$ . 1 Internal pull-up device enabled for port E bit $n$ .

### 9.6.5.4 Port E Slew Rate Enable Register (PTESE)

**Figure 9-30. Slew Rate Enable for Port E Register (PTESE)****Table 9-28. PTESE Register Field Descriptions**

Field	Description
7–0 PTESE $n$	<b>Output Slew Rate Enable for Port E Bits</b> — Each of these control bits determines if the output slew rate control is enabled for the associated PTE pin. For port E pins that are configured as inputs, these bits have no effect. 0 Output slew rate control disabled for port E bit $n$ . 1 Output slew rate control enabled for port E bit $n$ .

### 9.6.5.5 Port E Drive Strength Selection Register (PTEDS)

**Figure 9-31. Drive Strength Selection for Port E Register (PTEDS)****Table 9-29. PTEDS Register Field Descriptions**

Field	Description
7–0 PTEDS $n$	<b>Output Drive Strength Selection for Port E Bits</b> — Each of these control bits selects between low and high output drive for the associated PTE pin. For port E pins that are configured as inputs, these bits have no effect. 0 Low output drive strength selected for port E bit $n$ . 1 High output drive strength selected for port E bit $n$ .

### 9.6.5.6 Port E Input Filter Enable Register (PTEIFE)

The Port E pin incorporates an optional input low pass filter. The filter is enabled during and after reset by setting the associated PTEIFE bit. The filter is disabled through software control by clearing the associated PTEIFE bit.

	7	6	5	4	3	2	1	0
R W	PTEIFE7	PTEIFE6	PTEIFE5	PTEIFE4	PTEIFE3	PTEIFE2	PTEIFE1	PTEIFE0
Reset:	1	1	1	1	1	1	1	1

Figure 9-32. Port E Input Filter Enable Register (PTEIFE)

Table 9-30. PTEIFE Field Descriptions

Field	Description
7–0 PTEIFEn	<b>Port E Input Filter Enable</b> — Input low-pass filter enable control bits for PTE pins. 0 Pull-up disabled 1 Pull-up enabled

## 9.6.6 Port F Registers

Port F is an 8bit Input/Output Port and also shares its pins with the msCAN, TPM1 and the TPM2 peripherals. On reset, the pins default to the input state and under the I/O control. Enabling any of the digital peripherals msCAN, TPM1 and the TPM2 takes priority control over the pins.

Port F is controlled by the registers listed below.

### 9.6.6.1 Port F Data Register (PTFD)

	7	6	5	4	3	2	1	0
R W	PTFD7	PTFD6	PTFD5	PTFD4	PTFD3	PTFD2	PTFD1	PTFD0
Reset:	0	0	0	0	0	0	0	0

Figure 9-33. Port F Data Register (PTFD)

Table 9-31. PTFD Register Field Descriptions

Field	Description
7–0 PTFDn	<b>Port F Data Register Bits</b> — For port F pins that are inputs, reads return the logic level on the pin. For port F 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 F pins that are configured as outputs, the logic level is driven out the corresponding MCU pin. Reset forces PTFD to all 0s, but these 0s are not driven out the corresponding pins because reset also configures all port pins as high-impedance inputs with pull-ups disabled.

### 9.6.6.2 Port F Data Direction Register (PTFDD)

	7	6	5	4		3	2	1	0
R W	PTFDD7	PTFDD6	PTFDD5	PTFDD4		PTFDD3	PTFDD2	PTFDD1	PTFDD0
Reset:	0	0	0	0		0	0	0	0

Figure 9-34. Port F Data Direction Register (PTFDD)

Table 9-32. PTFDD Register Field Descriptions

Field	Description
7–0 PTFDD $n$	<b>Data Direction for Port F Bits</b> — These read/write bits control the direction of port F pins and what is read for PTFD reads. 0 Input (output driver disabled) and reads return the pin value. 1 Output driver enabled for port F bit $n$ and PTFD reads return the contents of PTFD $n$ .

### 9.6.6.3 Port F Pull Enable Register (PTFPE)

	7	6	5	4		3	2	1	0
R W	PTFPE7	PTFPE6	PTFPE5	PTFPE4		PTFPE3	PTFPE2	PTFPE1	PTFPE0
Reset:	0	0	0	0		0	0	0	0

Figure 9-35. Internal Pull Enable for Port F Register (PTFPE)

Table 9-33. PTFPE Register Field Descriptions

Field	Description
7–0 PTFPE $n$	<b>Internal Pull Enable for Port F Bits</b> — Each of these control bits determines if the internal pull-up device is enabled for the associated PTF pin. For port F 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 F bit $n$ . 1 Internal pull-up device enabled for port F bit $n$ .

### 9.6.6.4 Port F Slew Rate Enable Register (PTFSE)

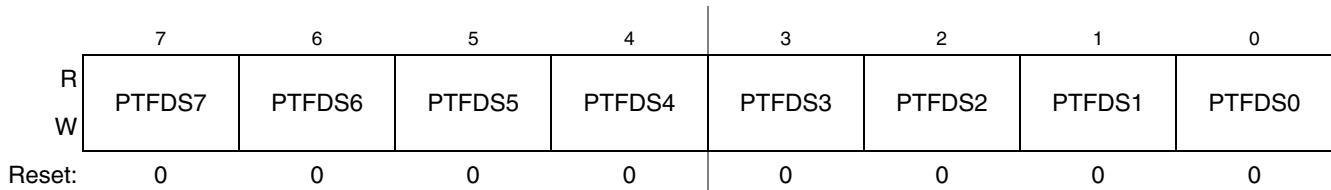
	7	6	5	4		3	2	1	0
R W	PTFSE7	PTFSE6	PTFSE5	PTFSE4		PTFSE3	PTFSE2	PTFSE1	PTFSE0
Reset:	1	1	1	1		1	1	1	1

Figure 9-36. Slew Rate Enable for Port F Register (PTFSE)

**Table 9-34. PTFSE Register Field Descriptions**

Field	Description
7–0 PTFSE $n$	<b>Output Slew Rate Enable for Port F Bits</b> — Each of these control bits determines if the output slew rate control is enabled for the associated PTF pin. For port F pins that are configured as inputs, these bits have no effect. 0 Output slew rate control disabled for port F bit $n$ . 1 Output slew rate control enabled for port F bit $n$ .

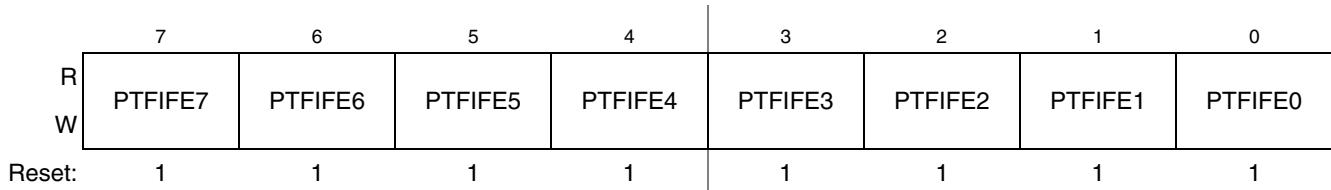
### 9.6.6.5 Port F Drive Strength Selection Register (PTFDS)

**Figure 9-37. Drive Strength Selection for Port F Register (PTFDS)****Table 9-35. PTFDS Register Field Descriptions**

Field	Description
7–0 PTFDS $n$	<b>Output Drive Strength Selection for Port F Bits</b> — Each of these control bits selects between low and high output drive for the associated PTF pin. For port F pins that are configured as inputs, these bits have no effect. 0 Low output drive strength selected for port F bit $n$ . 1 High output drive strength selected for port F bit $n$ .

### 9.6.6.6 Port F Input Filter Enable Register (PTFIFE)

The Port F pin incorporates an optional input low pass filter. The filter is enabled during and after reset by setting the associated PTFIFE bit. The filter is disabled through software control by clearing the associated PTFIFE bit.

**Figure 9-38. Port F Input Filter Enable Register (PTFIFE)****Table 9-36. PTFIFE Field Descriptions**

Field	Description
7–0 PTFIFE $n$	<b>Port F Input Filter Enable</b> — Input low-pass filter enable control bits for PTF pins. 0 Input filter disabled 1 Input filter enabled

## 9.6.7 Port G Registers

Port G is an 8bit Input/Output Port and also shares its pins with the EXTAL/XTAL and Keyboard Interrupt peripherals. On reset, the pins default to the input state and under the I/O control. Enabling any of the keyboard interupts or XTAL/EXTAL takes priority control over the pins.

Port G is controlled by the registers listed below.

### 9.6.7.1 Port G Data Register (PTGD)

	7	6	5	4	3	2	1	0
R W	PTGD7	PTGD6	PTGD5	PTGD4	PTGD3	PTGD2	PTGD1	PTGD0
Reset:	0	0	0	0	0	0	0	0

Figure 9-39. Port G Data Register (PTGD)

Table 9-37. PTGD Register Field Descriptions

Field	Description
7–0 PTGD $n$	<b>Port G Data Register Bits</b> — For port G pins that are inputs, reads return the logic level on the pin. For port G 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 G pins that are configured as outputs, the logic level is driven out the corresponding MCU pin. Reset forces PTGD to all 0s, but these 0s are not driven out the corresponding pins because reset also configures all port pins as high-impedance inputs with pull-ups disabled.

### 9.6.7.2 Port G Data Direction Register (PTGDD)

	7	6	5	4	3	2	1	0
R W	PTGDD7	PTGDD6	PTGDD5	PTGDD4	PTGDD3	PTGDD2	PTGDD1	PTGDD0
Reset:	0	0	0	0	0	0	0	0

Figure 9-40. Port G Data Direction Register (PTGDD)

Table 9-38. PTGDD Register Field Descriptions

Field	Description
7–0 PTGDD $n$	<b>Data Direction for Port G Bits</b> — These read/write bits control the direction of port G pins and what is read for PTGD reads. 0 Input (output driver disabled) and reads return the pin value. 1 Output driver enabled for port G bit $n$ and PTGD reads return the contents of PTGD $n$ .

### 9.6.7.3 Port G Pull Enable Register (PTGPE)

	7	6	5	4		3	2	1	0
R W	PTGPE7	PTGPE6	PTGPE5	PTGPE4		PTGPE3	PTGPE2	PTGPE1	PTGPE0
Reset:	0	0	0	0		0	0	0	0

Figure 9-41. Internal Pull Enable for Port G Register (PTGPE)

Table 9-39. PTGPE Register Field Descriptions

Field	Description
7–0 PTGPE $n$	<b>Internal Pull Enable for Port G Bits</b> — Each of these control bits determines if the internal pull-up device is enabled for the associated PTG pin. For port G 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 G bit $n$ . 1 Internal pull-up device enabled for port G bit $n$ .

### 9.6.7.4 Port G Slew Rate Enable Register (PTGSE)

	7	6	5	4		3	2	1	0
R W	PTGSE7	PTGSE6	PTGSE5	PTGSE4		PTGSE3	PTGSE2	PTGSE1	PTGSE0
Reset:	1	1	1	1		1	1	1	1

Figure 9-42. Slew Rate Enable for Port G Register (PTGSE)

Table 9-40. PTGSE Register Field Descriptions

Field	Description
7–0 PTGSE $n$	<b>Output Slew Rate Enable for Port G Bits</b> — Each of these control bits determines if the output slew rate control is enabled for the associated PTG pin. For port G pins that are configured as inputs, these bits have no effect. 0 Output slew rate control disabled for port G bit $n$ . 1 Output slew rate control enabled for port G bit $n$ .

### 9.6.7.5 Port G Drive Strength Selection Register (PTGDS)

	7	6	5	4		3	2	1	0
R W	PTGDS7	PTGDS6	PTGDS5	PTGDS4		PTGDS3	PTGDS2	PTGDS1	PTGDS0
Reset:	0	0	0	0		0	0	0	0

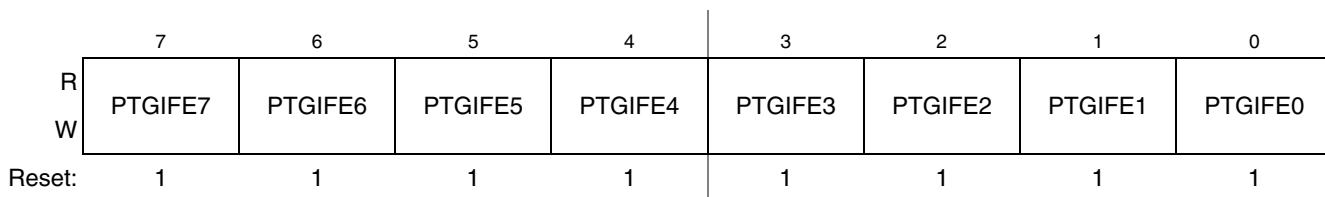
Figure 9-43. Drive Strength Selection for Port G Register (PTGDS)

**Table 9-41. PTGDS Register Field Descriptions**

Field	Description
7–0 PTGDS <sub>n</sub>	<b>Output Drive Strength Selection for Port G Bits</b> — Each of these control bits selects between low and high output drive for the associated PTG pin. For port G pins that are configured as inputs, these bits have no effect. 0 Low output drive strength selected for port G bit <i>n</i> . 1 High output drive strength selected for port G bit <i>n</i> .

### 9.6.7.6 Port G Input Filter Enable Register (PTGIFE)

The Port G pin incorporates an optional input low pass filter. The filter is enabled during and after reset by setting the associated PTGIFE bit. The filter is disabled through software control by clearing the associated PTGIFE bit.

**Figure 9-44. Port G Input Filter Enable Register (PTGIFE)****Table 9-42. PTGIFE Field Descriptions**

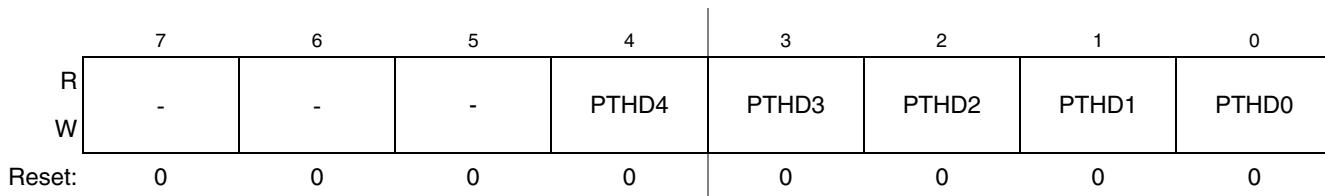
Field	Description
7–0 PTGIFE <sub>n</sub>	<b>Port G Input Filter Enable</b> — Input low-pass filter enable control bits for PTG pins. 0 Input filter disabled 1 Input filter enabled

### 9.6.8 Port H Registers

Port G is an 8-bit Input/Output Port and also shares its pins with the IIC2 and Rapid General Purpose Input output (GPIO) peripherals. On reset, the pins default to the input state and under the I/O control. Enabling the IIC2 or GPIO takes priority control over the pins.

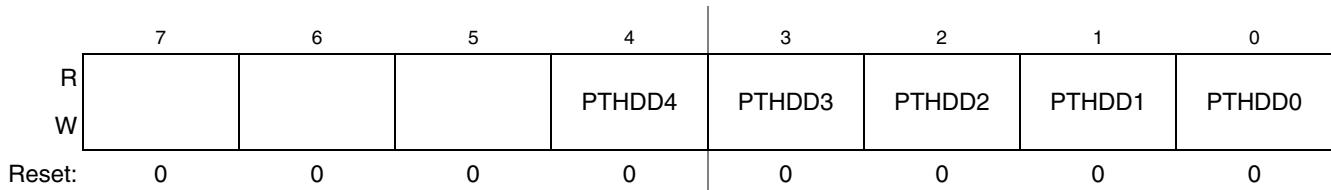
Port H is controlled by the registers listed below.

#### 9.6.8.1 Port H Data Register (PTHD)

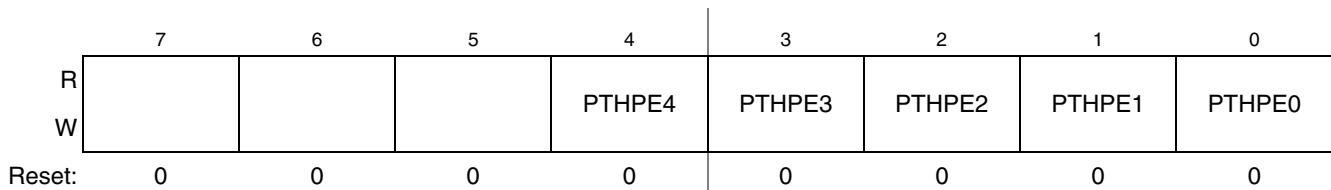
**Figure 9-45. Port H Data Register (PTHD)**

**Table 9-43. PTHD Register Field Descriptions**

Field	Description
4–0 PTHD $n$	<b>Port H Data Register Bits</b> — For port H pins that are inputs, reads return the logic level on the pin. For port H 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 H pins that are configured as outputs, the logic level is driven out the corresponding MCU pin. Reset forces PTHD to all 0s, but these 0s are not driven out the corresponding pins because reset also configures all port pins as high-impedance inputs with pull-ups disabled.

**9.6.8.2 Port H Data Direction Register (PTHDD)****Figure 9-46. Port H Data Direction Register (PTHDD)****Table 9-44. PTHDD Register Field Descriptions**

Field	Description
4–0 PTHDD $n$	<b>Data Direction for Port H Bits</b> — These read/write bits control the direction of port H pins and what is read for PTHD reads. 0 Input (output driver disabled) and reads return the pin value. 1 Output driver enabled for port H bit $n$ and PTHD reads return the contents of PTHD $n$ .

**9.6.8.3 Port H Pull Enable Register (PTHPE)****Figure 9-47. Internal Pull Enable for Port H Register (PTHPE)****Table 9-45. PTHPE Register Field Descriptions**

Field	Description
4–0 PTHPE $n$	<b>Internal Pull Enable for Port H Bits</b> — Each of these control bits determines if the internal pull-up device is enabled for the associated PTH pin. For port H 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 H bit $n$ . 1 Internal pull-up device enabled for port H bit $n$ .

### 9.6.8.4 Port H Slew Rate Enable Register (PTHSE)

	7	6	5	4		3	2	1	0
R W				PTHSE4	PTHSE3	PTHSE2	PTHSE1		PTHSE0
Reset:	1	1	1	1	1	1	1	1	1

Figure 9-48. Slew Rate Enable for Port H Register (PTHSE)

Table 9-46. PTHSE Register Field Descriptions

Field	Description
4–0 PTHSE $n$	<b>Output Slew Rate Enable for Port H Bits</b> — Each of these control bits determines if the output slew rate control is enabled for the associated PTH pin. For port H pins that are configured as inputs, these bits have no effect. 0 Output slew rate control disabled for port H bit $n$ . 1 Output slew rate control enabled for port H bit $n$ .

### 9.6.8.5 Port H Drive Strength Selection Register (PTHDS)

	7	6	5	4		3	2	1	0
R W				PTHDS4	PTHDS3	PTHDS2	PTHDS1		PTHDS0
Reset:	0	0	0	0	0	0	0	0	0

Figure 9-49. Drive Strength Selection for Port H Register (PTHDS)

Table 9-47. PTHDS Register Field Descriptions

Field	Description
4–0 PTHDS $n$	<b>Output Drive Strength Selection for Port H Bits</b> — Each of these control bits selects between low and high output drive for the associated PTH pin. For port H pins that are configured as inputs, these bits have no effect. 0 Low output drive strength selected for port H bit $n$ . 1 High output drive strength selected for port H bit $n$ .

### 9.6.8.6 Port H Input Filter Enable Register (PTHIFE)

The Port H pin incorporates an optional input low pass filter. The filter is enabled during and after reset by setting the associated PTHIFE bit. The filter is disabled through software control by clearing the associated PTHIFE bit.

	7	6	5	4		3	2	1	0
R W	PTHIFE7	PTHIFE6	PTHIFE5	PTHIFE4	PTHIFE3	PTHIFE2	PTHIFE1		PTHIFE0
Reset:	1	1	1	1	1	1	1	1	1

Figure 9-50. Port H Input Filter Enable Register (PTHIFE)

**Table 9-48. PTHIFE Field Descriptions**

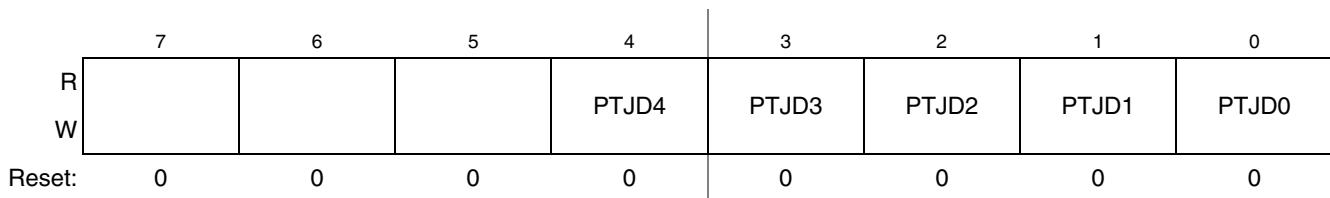
Field	Description
7–0 PTHIFE <sub>n</sub>	<b>Port H Input Filter Enable</b> — Input low-pass filter enable control bits for PTH pins. 0 Input filter disabled 1 Input filter enabled

## 9.6.9 Port J Registers

Port J is an 8-bit Input Output Port and shares its pins with the Rapid General Purpose Input output module (RGPIO). On reset, the pins default to the input state and under the I/O control. Enabling any of the RGPIO bits via the RGPIO\_ENB register takes priority over the normal I/O control.

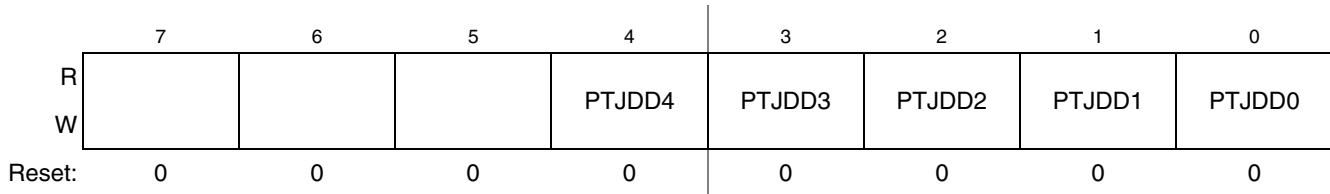
Port J is controlled by the registers listed below.

### 9.6.9.1 Port J Data Register (PTJD)

**Figure 9-51. Port J Data Register (PTJD)****Table 9-49. PTJD Register Field Descriptions**

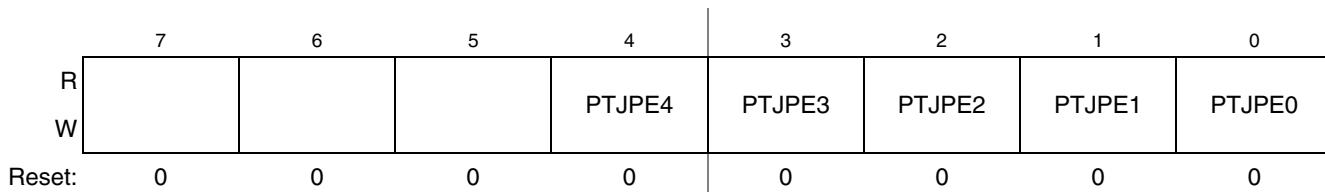
Field	Description
4–0 PTJD <sub>n</sub>	<b>Port J Data Register Bits</b> — For port J pins that are inputs, reads return the logic level on the pin. For port J pins that are configured as outputs, reads return the last value written to this register. Writes are latched into all bits of this register. For port J pins that are configured as outputs, the logic level is driven out the corresponding MCU pin. Reset forces PTJD to all 0s, but these 0s are not driven out the corresponding pins because reset also configures all port pins as high-impedance inputs with pull-ups disabled.

### 9.6.9.2 Port J Data Direction Register (PTJDD)

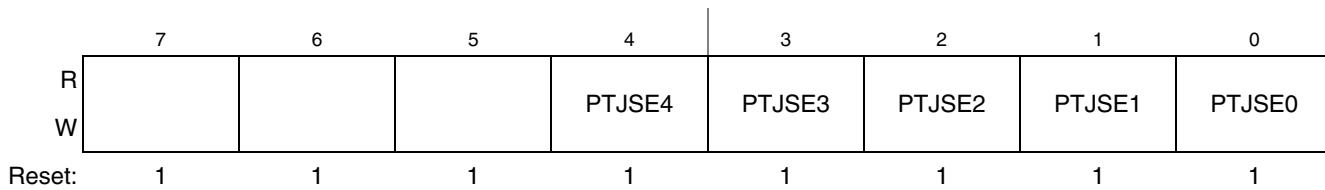
**Figure 9-52. Port J Data Direction Register (PTJDD)**

**Table 9-50. PTJDD Register Field Descriptions**

Field	Description
4–0 PTJDD $n$	<b>Data Direction for Port J Bits</b> — These read/write bits control the direction of port J pins and what is read for PTJD reads. 0 Input (output driver disabled) and reads return the pin value. 1 Output driver enabled for port J bit $n$ and PTJD reads return the contents of PTJD $n$ .

**9.6.9.3 Port J Pull Enable Register (PTJPE)****Figure 9-53. Internal Pull Enable for Port J Register (PTJPE)****Table 9-51. PTJPE Register Field Descriptions**

Field	Description
4–0 PTJPE $n$	<b>Internal Pull Enable for Port J Bits</b> — Each of these control bits determines if the internal pull-up device is enabled for the associated PTJ pin. For port J 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 J bit $n$ . 1 Internal pull-up device enabled for port J bit $n$ .

**9.6.9.4 Port J Slew Rate Enable Register (PTJSE)****Figure 9-54. Slew Rate Enable for Port J Register (PTJSE)****Table 9-52. PTJSE Register Field Descriptions**

Field	Description
4–0 PTJSE $n$	<b>Output Slew Rate Enable for Port J Bits</b> — Each of these control bits determines if the output slew rate control is enabled for the associated PTJ pin. For port J pins that are configured as inputs, these bits have no effect. 0 Output slew rate control disabled for port J bit $n$ . 1 Output slew rate control enabled for port J bit $n$ .

### 9.6.9.5 Port J Drive Strength Selection Register (PTJDS)

	7	6	5	4		3	2	1	0
R W				PTJDS4		PTJDS3	PTJDS2	PTJDS1	PTJDS0
Reset:	0	0	0	0		0	0	0	0

Figure 9-55. Drive Strength Selection for Port J Register (PTJDS)

Table 9-53. PTJDS Register Field Descriptions

Field	Description
4–0 PTJDS <sub>n</sub>	<b>Output Drive Strength Selection for Port J Bits</b> — Each of these control bits selects between low and high output drive for the associated PTJ pin. For port J pins that are configured as inputs, these bits have no effect. 0 Low output drive strength selected for port J bit <i>n</i> . 1 High output drive strength selected for port J bit <i>n</i> .

### 9.6.9.6 Port J Input Filter Enable Register (PTJIFE)

The Port J pin incorporates an optional input low pass filter. The filter is enabled during and after reset by setting the associated PTJIFE bit. The filter is disabled through software control by clearing the associated PTJIFE bit.

	7	6	5	4		3	2	1	0
R W	PTJIFE7	PTJIFE6	PTJIFE5	PTJIFE4		PTJIFE3	PTJIFE2	PTJIFE1	PTJIFE0
Reset:	1	1	1	1		1	1	1	1

Figure 9-56. Port J Input Filter Enable Register (PTJIFE)

Table 9-54. PTJIFE Field Descriptions

Field	Description
7–0 PTJIFE <sub>n</sub>	<b>Port J Input Filter Enable</b> — Input low-pass filter enable control bits for PTJ pins. 0 Input filter disabled 1 Input filter enabled

### 9.6.10 Keyboard Interrupt 1 (KBI1) Registers

KBI1 is controlled by the registers listed below. Table 9-55 shows KBI1 pin mapping to the port I/O pins.

Table 9-55. KBI1 Pin Mapping

Port pin	PTG3	PTG2	PTB5	PTB4	PTD3	PTD2	PTG1	PTG0
<b>KBI1 pin</b>	KBI1P7	KBI1P6	KBI1P5	KBI1P4	KBI1P3	KBI1P2	KBI1P1	KBI1P0

### 9.6.10.1 KBI1 Interrupt Status and Control Register (KBI1SC)

	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	0

Figure 9-57. KBI1 Interrupt Status and Control Register (KBI1SC)

Table 9-56. KBI1SC Register Field Descriptions

Field	Description
7–4	Reserved, should be cleared.
3 KBF	<b>KBI1 Interrupt Flag</b> — KBF indicates when a KBI1 interrupt is detected. Writes have no effect on KBF. 0 No KBI1 interrupt detected. 1 KBI1 interrupt detected.
2 KBACK	<b>KBI1 Interrupt Acknowledge</b> — Writing a 1 to KBACK is part of the flag clearing mechanism. KBACK always reads as 0.
1 KBIE	<b>KBI1 Interrupt Enable</b> — KBIE determines whether a KBI1 interrupt is requested. 0 KBI1 interrupt request not enabled. 1 KBI1 interrupt request enabled.
0 KBIMOD	<b>KBI1 Detection Mode</b> — KBIMOD (along with the KBI1ES bits) controls the detection mode of the KBI1 interrupt pins. 0 KBI1 pins detect edges only. 1 KBI1 pins detect edges and levels.

### 9.6.10.2 KBI1 Interrupt Pin Select Register (KBI1PE)

	7	6	5	4		3	2	1	0
R	KBIPE7	KBIPE6	KBIPE5	KBIPE4	KBIPE3	KBIPE2	KBIPE1	KBIPE0	
W	0	0	0	0	0	0	0	0	0

Figure 9-58. KBI1 Interrupt Pin Select Register (KBI1PE)

Table 9-57. KBI1PE Register Field Descriptions

Field	Description
7–0 KBIPE $n$	<b>KBI1 Interrupt Pin Selects</b> — Each of the KBIPE $n$ bits enable the corresponding KBI1 interrupt pin. 0 Pin not enabled as interrupt. 1 Pin enabled as interrupt.

### 9.6.10.3 KBI1 Interrupt Edge Select Register (KBI1ES)

	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 9-59. KBI1 Edge Select Register (KBI1ES)

Table 9-58. KBI1ES Register Field Descriptions

Field	Description
7–0 KBEDG $n$	<b>KBI1 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.

# 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.

#### 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

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.

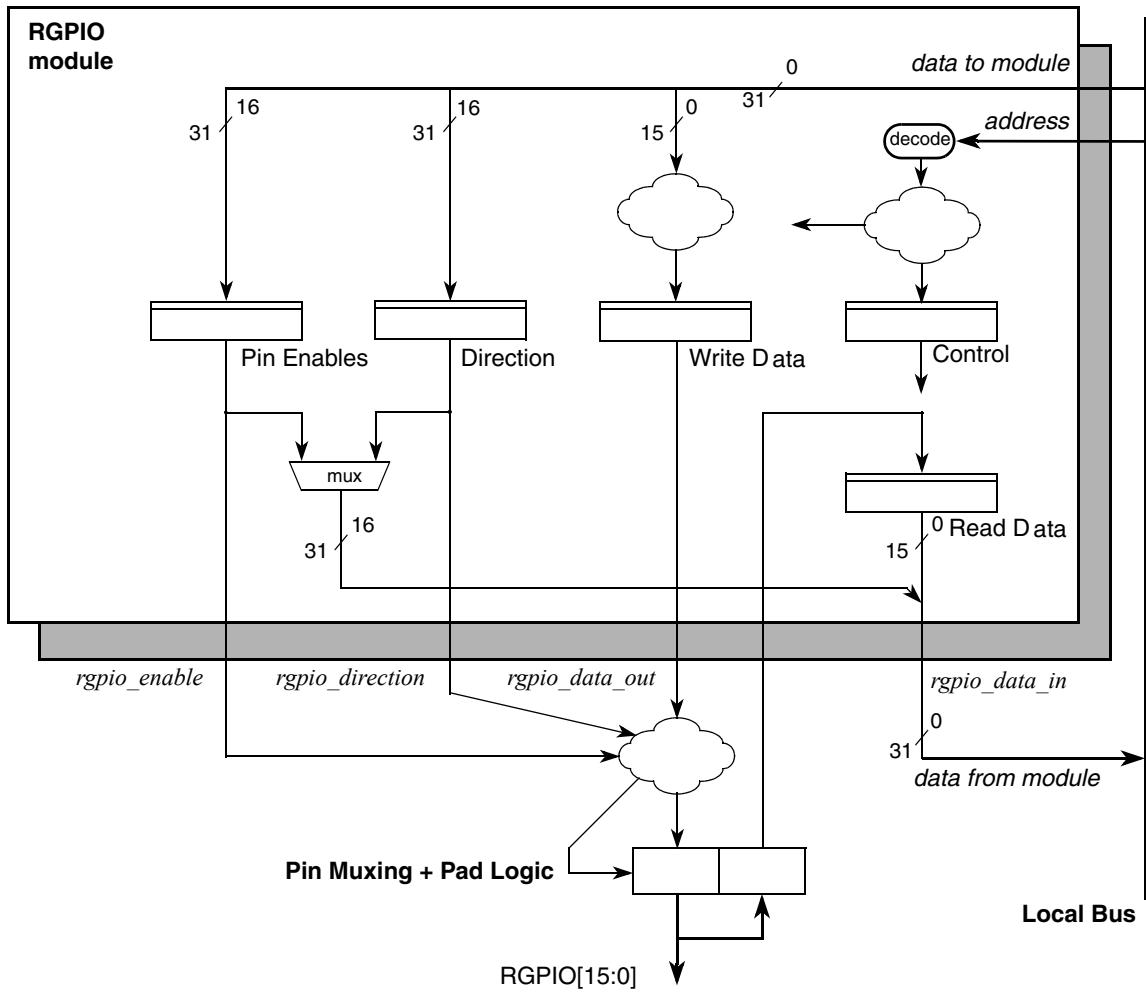


Figure 10-1. RGPIO Block Diagram

### 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
RGPIO[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	
RGPIO[15:0]	I/O	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.	
		State Meaning	Asserted— Input: Indicates the RGPIO pin was sampled as a logic high at the time of the read. Output: Indicates a properly-enabled RGPIO output pin is to be driven high. Negated— Input: Indicates the RGPIO pin was sampled as a logic low at the time of the read. Output: Indicates a properly-enabled RGPIO output pin is to be driven low.
		Timing	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 GPIO module provides a compact 16-byte programming model based at a system memory address of 0x(00)C0\_0000 (noted as GPIO\_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 GPIO pins. Additionally, the programming model can be referenced using any operand size access (byte, word, longword). Performance is typically maximized using 32-bit accesses.

### NOTE

Writes to the two-byte fields at GPIO\_BASE + 0x8 and GPIO\_BASE + 0xC are allowed, but do not affect any program-visible register within the GPIO module.

**Table 10-3. GPIO Write Memory Map**

Offset Address	Register	Width (bits)	Access	Reset Value	Section/Page
0x00	GPIO Data Direction Register (GPIO_DIR)	16	W	0x0000	<a href="#">10.3.1/10-4</a>
0x02	GPIO Write Data Register (GPIO_DATA)	16	W	0x0000	<a href="#">10.3.2/10-5</a>
0x04	GPIO Pin Enable Register (GPIO_ENB)	16	W	0x0000	<a href="#">10.3.3/10-6</a>
0x06	GPIO Write Data Clear Register (GPIO_CLR)	16	W	N/A	<a href="#">10.3.4/10-6</a>
0x0A	GPIO Write Data Set Register (GPIO_SET)	16	W	N/A	<a href="#">10.3.5/10-7</a>
0x0E	GPIO Write Data Toggle Register (GPIO_TOG)	16	W	N/A	<a href="#">10.3.6/10-7</a>

**Table 10-4. GPIO Read Memory Map**

Offset Address	Register	Width (bits)	Access	Reset Value	Section/Page
0x00	GPIO data direction register (GPIO_DIR)	16	R	0x0000	<a href="#">10.3.1/10-4</a>
0x02	GPIO write data register (GPIO_DATA)	16	R	0x0000	<a href="#">10.3.2/10-5</a>
0x04	GPIO pin enable register (GPIO_ENB)	16	R	0x0000	<a href="#">10.3.3/10-6</a>
0x06	GPIO write data register (GPIO_DATA)	16	R	0x0000	<a href="#">10.3.2/10-5</a>
0x08	GPIO data direction register (GPIO_DIR)	16	R	0x0000	<a href="#">10.3.1/10-4</a>
0x0A	GPIO write data register (GPIO_DATA)	16	R	0x0000	<a href="#">10.3.2/10-5</a>
0x0C	GPIO data direction register (GPIO_DIR)	16	R	0x0000	<a href="#">10.3.1/10-4</a>
0x0E	GPIO write data register (GPIO_DATA)	16	R	0x0000	<a href="#">10.3.2/10-5</a>

### 10.3.1 GPIO Data Direction (GPIO\_DIR)

The read/write GPIO\_DIR register defines whether a properly-enabled GPIO pin is configured as an input or output:

- Setting any bit in GPIO\_DIR configures a properly-enabled GPIO port pin as an output
- Clearing any bit in GPIO\_DIR configures a properly-enabled GPIO port pin as an input

At reset, all bits in the RGPI0\_DIR are cleared.

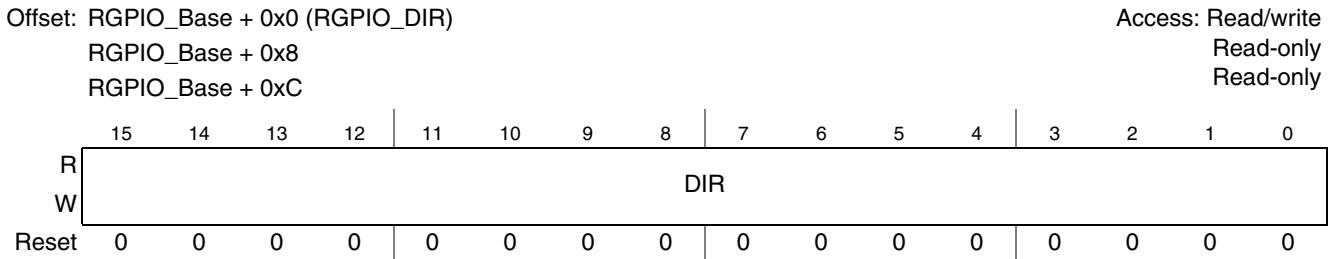


Figure 10-2. RGPI0 Data Direction Register (RGPI0\_DIR)

Table 10-5. RGPI0\_DIR Field Descriptions

Field	Description
15–0 DIR	RGPI0 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.

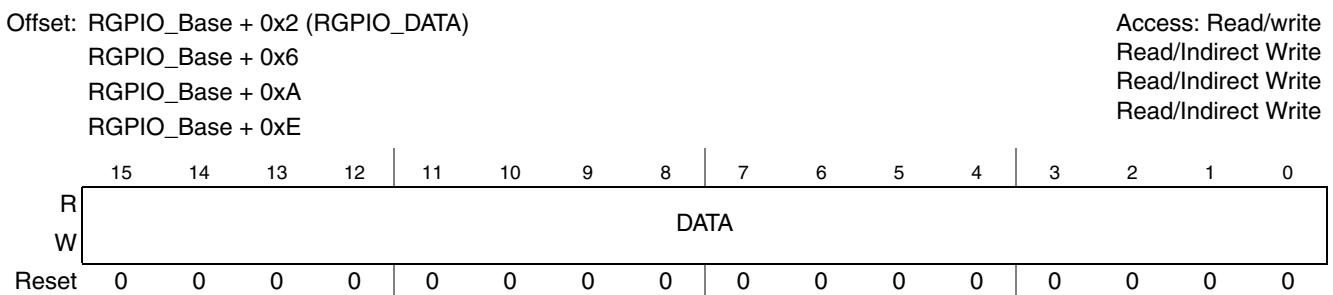


Figure 10-3. RGPI0 Data Register (RGPI0\_DATA)

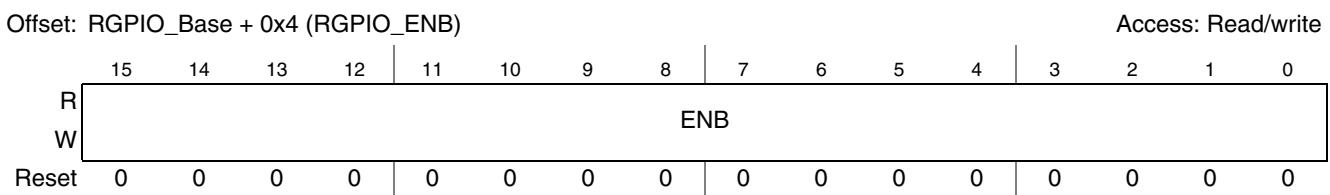
**Table 10-6. GPIO\_DATA Field Descriptions**

Field	Description
15–0 DATA	GPIO data. 0 A properly-enabled GPIO output pin is driven with a logic 0, or a properly-enabled GPIO input pin was read as a logic 0 1 A properly-enabled GPIO output pin is driven with a logic 1, or a properly-enabled GPIO input pin was read as a logic 1

### 10.3.3 GPIO Pin Enable (GPIO\_ENB)

The GPIO\_ENB register configures the corresponding package pin as a GPIO pin instead of the normal GPIO pin mapped onto the peripheral bus.

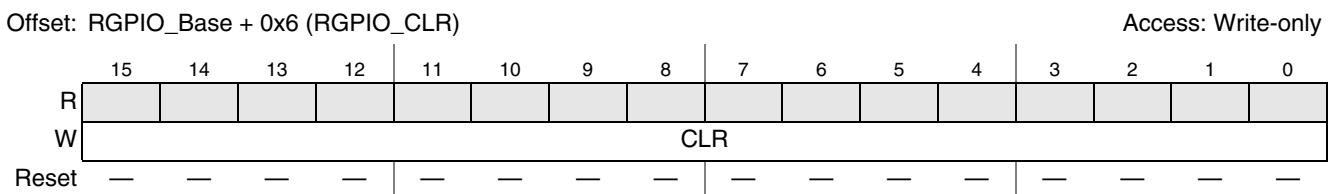
The GPIO\_ENB register is read/write. At reset, all bits in the GPIO\_ENB are cleared, disabling the GPIO functionality.

**Figure 10-4. GPIO Enable Register (GPIO\_ENB)****Table 10-7. GPIO\_ENB Field Descriptions**

Field	Description
15–0 ENB	GPIO enable. 0 The corresponding package pin is configured for use as a normal GPIO pin, not a GPIO 1 The corresponding package pin is configured for use as a GPIO pin

### 10.3.4 GPIO Clear Data (GPIO\_CLR)

The GPIO\_CLR register provides a mechanism to clear specific bits in the GPIO\_DATA by performing a simple write. Clearing a bit in GPIO\_CLR clears the corresponding bit in the GPIO\_DATA register. Setting it has no effect. The GPIO\_CLR register is write-only; reads of this address return the GPIO\_DATA register.

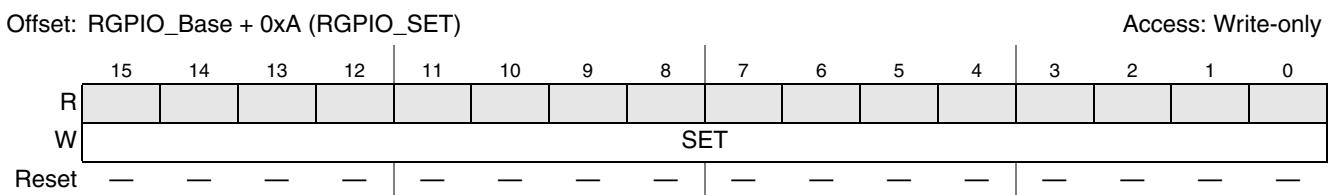
**Figure 10-5. GPIO Clear Data Register (GPIO\_CLR)**

**Table 10-8. GPIO\_CLR Field Descriptions**

Field	Description
15–0 CLR	GPIO clear data. 0 Clears the corresponding bit in the GPIO_DATA register 1 No effect

### 10.3.5 GPIO Set Data (GPIO\_SET)

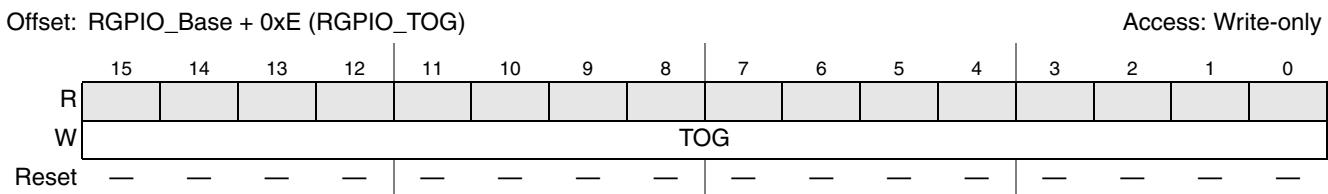
The GPIO\_SET register provides a mechanism to set specific bits in the GPIO\_DATA register by performing a simple write. Setting a bit in GPIO\_SET asserts the corresponding bit in the GPIO\_DATA register. Clearing it has no effect. The GPIO\_SET register is write-only; reads of this address return 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	GPIO set data. 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. GPIO Toggle Data Register (GPIO\_TOG)**

**Table 10-10. RGPIO\_TOG Field Descriptions**

Field	Description
15–0 TOG	RGPIO toggle data. 0 No effect 1 Inverts the corresponding bit in RGPIO_DATA

## 10.4 Functional Description

The RGPIO module is a relatively-simple design with its behavior controlled by the program-visible registers defined within its programming model.

The RGPIO 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 RGPIO 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 RGPIO module disables the entire 16-bit data port. Prior to using the RGPIO port, software typically:

- Enables the appropriate pins in RGPIO\_ENB
- Configures the pin direction in RGPIO\_DIR
- Defines the contents of the data register (RGPIO\_DATA)

## 10.6 Application Information

This section examines the relative performance of the RGPIO 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 (RGPIO 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**

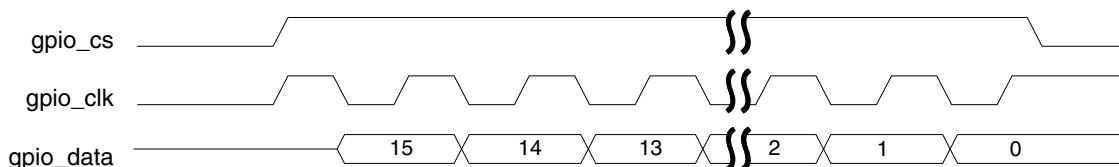
Loop	Peripheral Bus-mapped GPIO			RGPIO		
	Sq-Wave Frequency	Frequency @ CPU $f = 50$ MHz	Relative Speed	Sq-Wave Frequency	Frequency @ CPU $f = 50$ MHz	Relative Speed
bchg	$(1/24) \times f$ MHz	2.083 MHz	1.00x	$(1/14) \times f$ MHz	3.571 MHz	1.71x
set+clr (+toggle)	$(1/12) \times f$ MHz	4.167 MHz	2.00x	$(1/8) \times f$ 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
```

## Rapid GPIO (RGPIO)

```

send_16b_spi_message_rgpio:
00510: 4fef ffff4          lea    -12(%sp),%sp      # allocate stack space
00514: 48d7 008c           movm.l &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 fff5     and.l   &0xffff,%d0    # negate chip-select
00568: 1080                 mov.b   %d0,(%a0)    # update gpio

0056a: 4cd7 008c           movm.l (%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

## Freescale's Controller Area Network (MSCAN)

### 11.1 Introduction

#### 11.1.1 Features

Freescale's controller area network (MSCAN) definition is based on the MSCAN12 definition, which is the specific implementation of the MSCAN concept targeted for the M68HC12 microcontroller Family.

The module is a communication controller implementing the CAN 2.0A/B protocol as defined in the Bosch specification dated September 1991. For users to fully understand the MSCAN specification, it is recommended that the Bosch specification be read first to familiarize the reader with the terms and concepts contained within this document.

Though not exclusively intended for automotive applications, CAN protocol is designed to meet the specific requirements of a vehicle serial data bus: real-time processing, reliable operation in the EMI environment of a vehicle, cost-effectiveness, and required bandwidth.

MSCAN uses an advanced buffer arrangement resulting in predictable real-time behavior and simplified application software.

The basic features of the MSCAN are as follows:

- Implementation of the CAN protocol — Version 2.0A/B
  - Standard and extended data frames
  - Zero to eight bytes data length
  - Programmable bit rate up to 1 Mbps<sup>1</sup>
  - Support for remote frames
- Five receive buffers with FIFO storage scheme
- Three transmit buffers with internal prioritization using a local priority concept
- Flexible maskable identifier filter supports two full-size (32-bit) extended identifier filters, or four 16-bit filters, or eight 8-bit filters
- Programmable wakeup functionality with integrated low-pass filter
- Programmable loopback mode supports self-test operation
- Programmable listen-only mode for monitoring of CAN bus
- Programmable bus-off recovery functionality
- Separate signalling and interrupt capabilities for all CAN receiver and transmitter error states (warning, error passive, bus-off)

1. Depending on the actual bit timing and the clock jitter of the PLL.

- Programmable MSCAN clock source bus clock or oscillator clock
- Internal timer for time-stamping of received and transmitted messages
- Three low-power modes: sleep, power down, and MSCAN enable
- Global initialization of configuration registers

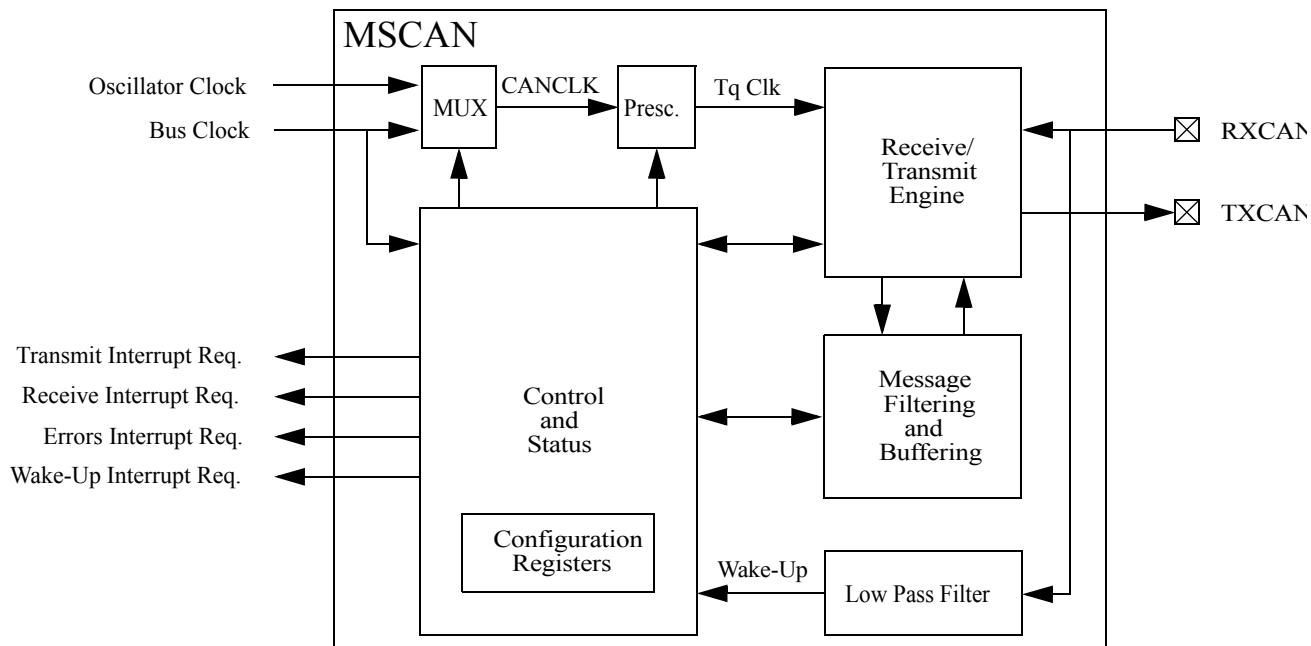
### 11.1.2 Modes of Operation

For details on low-power mode operation, refer to [Table 3-2](#) in [Chapter 3, “Modes of Operation”](#).

The following modes of operation are specific to the MSCAN. See [Section 11.5, “Functional Description,”](#) for details.

- Listen-Only Mode
- MSCAN Sleep Mode
- MSCAN Initialization Mode
- MSCAN Power Down Mode

### 11.1.3 Block Diagram



**Figure 11-1. MSCAN Block Diagram**

## 11.2 External Signal Description

The MSCAN uses two external pins.

### 11.2.1 RXCAN — CAN Receiver Input Pin

RXCAN is the MSCAN receiver input pin.

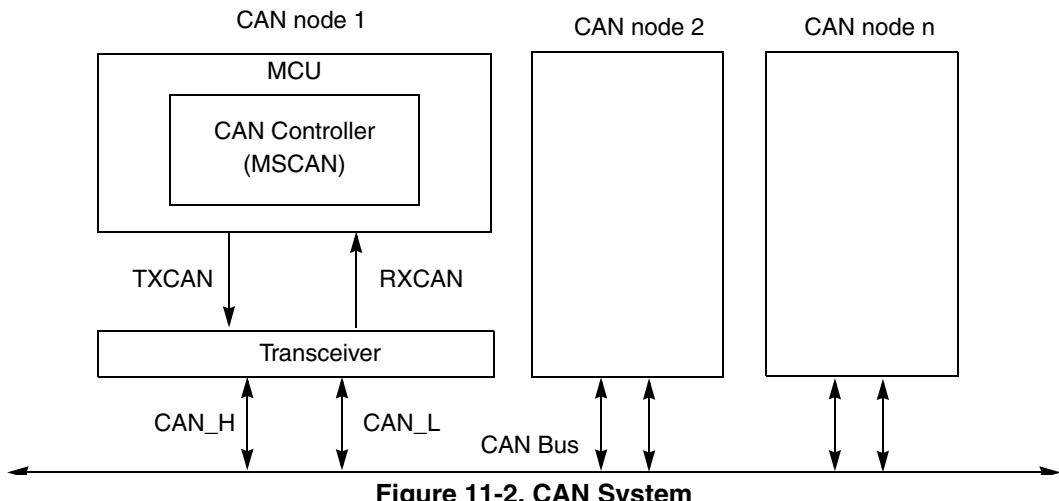
## 11.2.2 TXCAN — CAN Transmitter Output Pin

TXCAN is the MSCAN transmitter output pin. The TXCAN output pin represents the logic level on the CAN bus:

- 0 = Dominant state
- 1 = Recessive state

## 11.2.3 CAN System

A typical CAN system with MSCAN is shown in [Figure 11-2](#). Each CAN node is connected physically to the CAN bus lines through a transceiver device. The transceiver is capable of driving the large current needed for the CAN bus and has current protection against defective CAN or defective nodes.



**Figure 11-2. CAN System**

## 11.3 Register Definition

This section describes in detail all the registers and register bits in the MSCAN module. Each description includes a standard register diagram with an associated figure number. Details of register bit and field function follow the register diagrams, in bit order. All bits of all registers in this module are completely synchronous to internal clocks during a register read.

### 11.3.1 MSCAN Control Register 0 (CANCTL0)

The CANCTL0 register provides various control bits of the MSCAN module as described below.

R	7	6	5	4	3	2	1	0
	RXFRM	RXACT	CSWAI	SYNCH		TIME	WUPE	SLPRQ
W					0	0	0	1
Reset:	0	0	0	0	0	0	0	1
	= Unimplemented							

**Figure 11-3. MSCAN Control Register 0 (CANCTL0)**

## NOTE

The CANCTL0 register, except WUPE, INITRQ, and SLPRQ, is held in the reset state when the initialization mode is active (INITRQ = 1 and INITAK = 1). This register is writable again as soon as the initialization mode is exited (INITRQ = 0 and INITAK = 0).

Read: Anytime

Write: Anytime when out of initialization mode; exceptions are read-only RXACT and SYNCH, RXFRM (set by the module only), and INITRQ (also writable in initialization mode).

**Table 11-1. CANCTL0 Register Field Descriptions**

Field	Description
7 RXFRM <sup>1</sup>	<b>Received Frame Flag</b> — This bit is read and clear only. It is set when a receiver has received a valid message correctly, independently of the filter configuration. After it is set, it remains set until cleared by software or reset. Clearing is done by writing a 1. Writing a 0 is ignored. This bit is not valid in loopback mode. 0 No valid message was received since last clearing this flag 1 A valid message was received since last clearing of this flag
6 RXACT	<b>Receiver Active Status</b> — This read-only flag indicates the MSCAN is receiving a message. The flag is controlled by the receiver front end. This bit is not valid in loopback mode. 0 MSCAN is transmitting or idle <sup>2</sup> 1 MSCAN is receiving a message (including when arbitration is lost) <sup>2</sup>
5 CSWAI <sup>3</sup>	<b>CAN Stops in Wait Mode</b> — Enabling this bit allows for lower power consumption in wait mode by disabling all the clocks at the CPU bus interface to the MSCAN module. 0 The module is not affected during wait mode 0 The module ceases to be clocked during wait mode
4 SYNCH	<b>Synchronized Status</b> — This read-only flag indicates whether the MSCAN is synchronized to the CAN bus and able to participate in the communication process. It is set and cleared by the MSCAN. 0 MSCAN is not synchronized to the CAN bus 0 MSCAN is synchronized to the CAN bus
3 TIME	<b>Timer Enable</b> — This bit activates an internal 16-bit wide free running timer clocked by the bit clock rate. If the timer is enabled, a 16-bit time stamp is assigned to each transmitted/received message within the active TX/RX buffer. As soon as a message is acknowledged on the CAN bus, the time stamp is written to the highest bytes (0x000E, 0x000F) in the appropriate buffer (see <a href="#">Section 11.4, “Programmer’s Model of Message Storage”</a> ). The internal timer is reset (all bits set to 0) when disabled. This bit is held low in initialization mode. 0 Disable internal MSCAN timer 0 Enable internal MSCAN timer
2 WUPE <sup>4</sup>	<b>Wake-Up Enable</b> — This configuration bit allows the MSCAN to restart from sleep mode when traffic on CAN is detected (see <a href="#">Section 11.5.5.4, “MSCAN Sleep Mode”</a> ). 0 Wake-up disabled — The MSCAN ignores traffic on CAN 0 Wake-up enabled — The MSCAN is able to restart

**Table 11-1. CANCTL0 Register Field Descriptions (continued)**

Field	Description
1 SLPRQ <sup>5</sup>	<p><b>Sleep Mode Request</b> — This bit requests the MSCAN to enter sleep mode, which is an internal power saving mode (see <a href="#">Section 11.5.5.4, “MSCAN Sleep Mode”</a>). The sleep mode request is serviced when the CAN bus is idle, i.e., the module is not receiving a message and all transmit buffers are empty. The module indicates entry to sleep mode by setting SLPAK = 1 (see <a href="#">Section 11.3.2, “MSCAN Control Register 1 (CANCTL1)”</a>). SLPRQ cannot be set while the WUPIF flag is set (see <a href="#">Section 11.3.5, “MSCAN Receiver Flag Register (CANRFLG)”</a>). Sleep mode is active until SLPRQ is cleared by the CPU. Depending on the setting of WUPE, the MSCAN detects activity on the CAN bus and clears SLPRQ itself.</p> <p>0 Running — The MSCAN functions normally</p> <p>1 Sleep mode request — The MSCAN enters sleep mode when CAN bus idle</p>
0 INITRQ <sup>6,7</sup>	<p><b>Initialization Mode Request</b> — When this bit is set by the CPU, the MSCAN skips to initialization mode (see <a href="#">Section 11.5.5.5, “MSCAN Initialization Mode”</a>). Any ongoing transmission or reception is aborted. Synchronization to the CAN bus is lost. The module indicates entry to initialization mode by setting INITAK = 1 (<a href="#">Section 11.3.2, “MSCAN Control Register 1 (CANCTL1)”</a>).</p> <p>The following registers enter their hard reset state and restore their default values: CANCTL0<sup>8</sup>, CANRFLG<sup>9</sup>, CANRIER<sup>10</sup>, CANTFLG, CANTIER, CANTARQ, CANTAACK, and CANTBSEL.</p> <p>The registers CANCTL1, CANBTR0, CANBTR1, CANIDAC, CANIDAR0-7, and CANIDMR0-7 can only be written by the CPU when the MSCAN is in initialization mode (INITRQ = 1 and INITAK = 1). The values of the error counters are not affected by initialization mode.</p> <p>When this bit is cleared by the CPU, the MSCAN restarts and then tries to synchronize to the CAN bus. If the MSCAN is not in bus-off state, it synchronizes after 11 consecutive recessive bits on the CAN bus; if the MSCAN is in bus-off state, it continues to wait for 128 occurrences of 11 consecutive recessive bits.</p> <p>Writing to other bits in CANCTL0, CANRFLG, CANRIER, CANTFLG, or CANTIER must be done only after initialization mode is exited, which is INITRQ = 0 and INITAK = 0.</p> <p>0 Normal operation</p> <p>1 MSCAN in initialization mode</p>

<sup>1</sup> The MSCAN must be in normal mode for this bit to become set.

<sup>2</sup> See the Bosch CAN 2.0A/B specification for a detailed definition of transmitter and receiver states.

<sup>3</sup> To protect from accidentally violating the CAN protocol, the TXCAN pin is immediately forced to a recessive state when the CPU enters wait (CSWAI = 1) or stop mode (see [Section 11.5.5.2, “Operation in Wait Mode”](#) and [Section 11.5.5.3, “Operation in Stop Mode”](#)).

<sup>4</sup> The CPU has to make sure that the WUPE bit and the WUPIE wake-up interrupt enable bit (see [Section 11.3.6, “MSCAN Receiver Interrupt Enable Register \(CANRIER\)”](#)) is enabled, if the recovery mechanism from stop or wait is required.

<sup>5</sup> The CPU cannot clear SLPRQ before the MSCAN has entered sleep mode (SLPRQ = 1 and SLPAK = 1).

<sup>6</sup> The CPU cannot clear INITRQ before the MSCAN has entered initialization mode (INITRQ = 1 and INITAK = 1).

<sup>7</sup> To protect from accidentally violating the CAN protocol, the TXCAN pin is immediately forced to a recessive state when the initialization mode is requested by the CPU. Thus, the recommended procedure is to bring the MSCAN into sleep mode (SLPRQ = 1 and SLPAK = 1) before requesting initialization mode.

<sup>8</sup> Not including WUPE, INITRQ, and SLPRQ.

<sup>9</sup> TSTAT1 and TSTAT0 are not affected by initialization mode.

<sup>10</sup> RSTAT1 and RSTAT0 are not affected by initialization mode.

### 11.3.2 MSCAN Control Register 1 (CANCTL1)

The CANCTL1 register provides various control bits and handshake status information of the MSCAN module as described below.

	7	6	5	4		3	2	1	0
R	CANE	CLKSRC	LOOPB	LISTEN	BORM	WUPM	SLPAK	INITAK	
W									
Reset:	0	0	0	1	0	0	0	1	

= Unimplemented

Figure 11-4. MSCAN Control Register 1(CANCTL1)

Read: Anytime

Write: Anytime when INITRQ = 1 and INITAK = 1, except CANE that is write once in normal and anytime in special system operation modes when the MSCAN is in initialization mode (INITRQ = 1 and INITAK = 1).

Table 11-2. CANCTL1 Register Field Descriptions

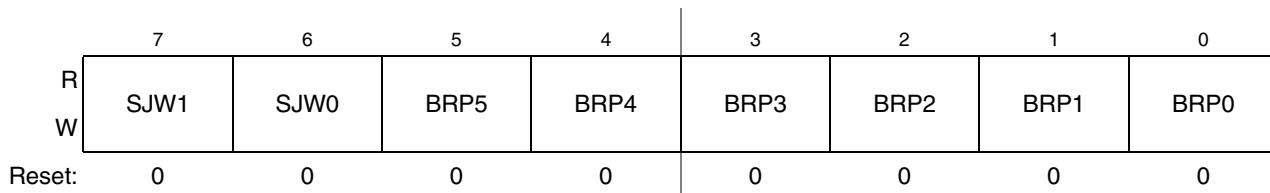
Field	Description
7 CANE	MSCAN Enable 0 MSCAN module is disabled 1 MSCAN module is enabled
6 CLKSRC	<b>MSCAN Clock Source</b> — This bit defines the clock source for the MSCAN module (only for systems with a clock generation module; <a href="#">Section 11.5.2.7, “Clock System,”</a> and <a href="#">Section Figure 11-41., “MSCAN Clocking Scheme,”</a> ). 0 MSCAN clock source is the oscillator clock 1 MSCAN clock source is the bus clock
5 LOOPB	<b>Loopback Self Test Mode</b> — When this bit is set, the MSCAN performs an internal loopback that can be used for self test operation. The bit stream output of the transmitter is fed back to the receiver internally. 0 Loopback self test disabled 1 Loopback self test enabled
4 LISTEN	<b>Listen Only Mode</b> — This bit configures the MSCAN as a CAN bus monitor. When LISTEN is set, all valid CAN messages with matching ID are received, but no acknowledgement or error frames are sent out (see <a href="#">Section 11.5.4.4, “Listen-Only Mode”</a> ). In addition, the error counters are frozen. Listen only mode supports applications that require hot plugging or throughput analysis. The MSCAN is unable to transmit any messages when listen only mode is active. 0 Normal operation 1 Listen only mode activated
3 BORM	<b>Bus-Off Recovery Mode</b> — This bits configures the bus-off state recovery mode of the MSCAN. Refer to <a href="#">Section 11.6.2, “Bus-Off Recovery,”</a> for details. 0 Automatic bus-off recovery (see Bosch CAN 2.0A/B protocol specification) 1 Bus-off recovery upon user request
2 WUPM	<b>Wake-Up Mode</b> — If WUPE in CANCTL0 is enabled, this bit defines whether the integrated low-pass filter is applied to protect the MSCAN from spurious wake-up (see <a href="#">Section 11.5.5.4, “MSCAN Sleep Mode”</a> ). 0 MSCAN wakes up the CPU after any recessive to dominant edge on the CAN bus 1 MSCAN wakes up the CPU only in case of a dominant pulse on the CAN bus that has a length of $T_{wup}$

**Table 11-2. CANCTL1 Register Field Descriptions (continued)**

Field	Description
1 SLPAK	<b>Sleep Mode Acknowledge</b> — This flag indicates whether the MSCAN module has entered sleep mode (see <a href="#">Section 11.5.5.4, “MSCAN Sleep Mode”</a> ). It is used as a handshake flag for the SLPRQ sleep mode request. Sleep mode is active when SLPRQ = 1 and SLPAK = 1. Depending on the setting of WUPE, the MSCAN clears the flag if it detects activity on the CAN bus while in sleep mode. CPU clearing the SLPRQ bit also resets the SLPAK bit. 0 Running — The MSCAN operates normally 1 Sleep mode active — The MSCAN has entered sleep mode
0 INITAK	<b>Initialization Mode Acknowledge</b> — This flag indicates whether the MSCAN module is in initialization mode (see <a href="#">Section 11.5.5.5, “MSCAN Initialization Mode”</a> ). It is used as a handshake flag for the INITRQ initialization mode request. Initialization mode is active when INITRQ = 1 and INITAK = 1. The registers CANCTL1, CANBTR0, CANBTR1, CANIDAC, CANIDAR0–CANIDAR7, and CANIDMR0–CANIDMR7 can be written only by the CPU when the MSCAN is in initialization mode. 0 Running — The MSCAN operates normally 1 Initialization mode active — The MSCAN is in initialization mode

### 11.3.3 MSCAN Bus Timing Register 0 (CANBTR0)

The CANBTR0 register configures various CAN bus timing parameters of the MSCAN module.

**Figure 11-5. MSCAN Bus Timing Register 0 (CANBTR0)**

Read: Anytime

Write: Anytime in initialization mode (INITRQ = 1 and INITAK = 1)

**Table 11-3. CANBTR0 Register Field Descriptions**

Field	Description
7:6 SJW[1:0]	<b>Synchronization Jump Width</b> — The synchronization jump width defines the maximum number of time quanta ( $T_q$ ) clock cycles a bit can be shortened or lengthened to achieve resynchronization to data transitions on the CAN bus (see <a href="#">Table 11-4</a> ).
5:0 BRP[5:0]	<b>Baud Rate Prescaler</b> — These bits determine the time quanta ( $T_q$ ) clock used to build up the bit timing (see <a href="#">Table 11-5</a> ).

**Table 11-4. Synchronization Jump Width**

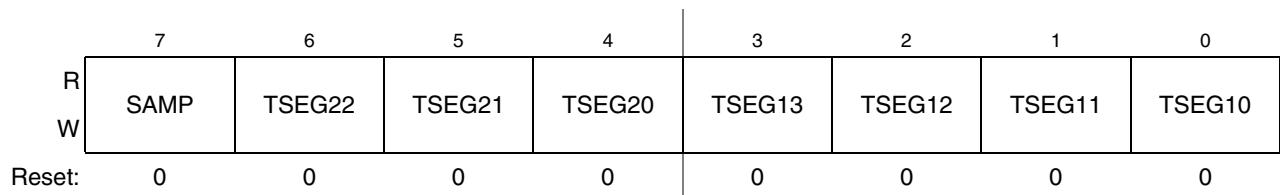
SJW1	SJW0	Synchronization Jump Width
0	0	1 $T_q$ clock cycle
0	1	2 $T_q$ clock cycles
1	0	3 $T_q$ clock cycles
1	1	4 $T_q$ clock cycles

**Table 11-5. Baud Rate Prescaler**

<b>BRP5</b>	<b>BRP4</b>	<b>BRP3</b>	<b>BRP2</b>	<b>BRP1</b>	<b>BRP0</b>	<b>Prescaler value (P)</b>
0	0	0	0	0	0	1
0	0	0	0	0	1	2
0	0	0	0	1	0	3
0	0	0	0	1	1	4
:	:	:	:	:	:	:
1	1	1	1	1	1	64

### 11.3.4 MSCAN Bus Timing Register 1 (CANBTR1)

The CANBTR1 register configures various CAN bus timing parameters of the MSCAN module.

**Figure 11-6. MSCAN Bus Timing Register 1 (CANBTR1)**

Read: Anytime

Write: Anytime in initialization mode (INITRQ = 1 and INITAK = 1)

**Table 11-6. CANBTR1 Register Field Descriptions**

Field	Description
7 SAMP	<b>Sampling</b> — This bit determines the number of CAN bus samples taken per bit time. 0 One sample per bit. 1 Three samples per bit <sup>1</sup> . If SAMP = 0, the resulting bit value is equal to the value of the single bit positioned at the sample point. If SAMP = 1, the resulting bit value is determined by using majority rule on the three total samples. For higher bit rates, it is recommended that only one sample is taken per bit time (SAMP = 0).
6:4 TSEG2[2:0]	<b>Time Segment 2</b> — Time segments within the bit time fix the number of clock cycles per bit time and the location of the sample point ( <a href="#">Figure 11-42</a> ). Time segment 2 (TSEG2) values are programmable as shown in <a href="#">Table 11-7</a> .
3:0 TSEG1[3:0]	<b>Time Segment 1</b> — Time segments within the bit time fix the number of clock cycles per bit time and the location of the sample point ( <a href="#">Figure 11-42</a> ). Time segment 1 (TSEG1) values are programmable as shown in <a href="#">Table 11-8</a> .

<sup>1</sup> In this case, PHASE\_SEG1 must be at least 2 time quanta (Tq).

**Table 11-7. Time Segment 2 Values**

TSEG22	TSEG21	TSEG20	Time Segment 2
0	0	0	1 Tq clock cycle <sup>1</sup>
0	0	1	2 Tq clock cycles
:	:	:	:
1	1	0	7 Tq clock cycles
1	1	1	8 Tq clock cycles

<sup>1</sup> This setting is not valid. Please refer to [Table 11-35](#) for valid settings.

**Table 11-8. Time Segment 1 Values**

TSEG13	TSEG12	TSEG11	TSEG10	Time segment 1
0	0	0	0	1 Tq clock cycle <sup>1</sup>
0	0	0	1	2 Tq clock cycles <sup>1</sup>
0	0	1	0	3 Tq clock cycles <sup>1</sup>
0	0	1	1	4 Tq clock cycles
:	:	:	:	:
1	1	1	0	15 Tq clock cycles
1	1	1	1	16 Tq clock cycles

<sup>1</sup> This setting is not valid. Please refer to [Table 11-35](#) for valid settings.

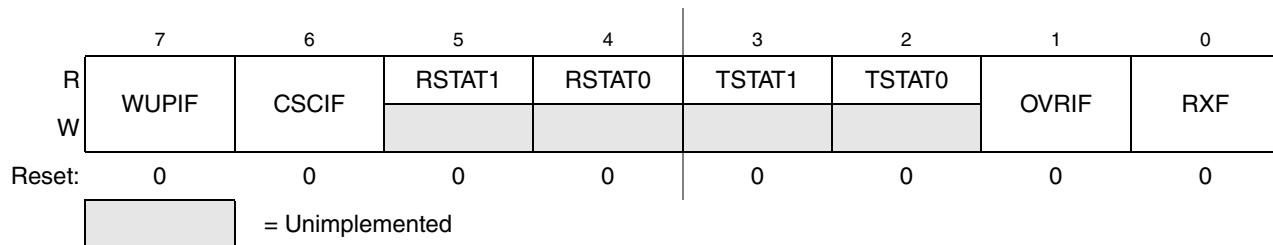
The bit time is determined by the oscillator frequency, the baud rate prescaler, and the number of time quanta (Tq) clock cycles per bit (as shown in [Table 11-7](#) and [Table 11-8](#)).

**Eqn. 11-1**

$$\text{bit.time} = \frac{(\text{Prescaler} \cdot \text{value})}{f_{\text{CANCLK}}} \cdot (1 + \text{TimeSegment1} + \text{TimeSegment2})$$

### 11.3.5 MSCAN Receiver Flag Register (CANRFLG)

A flag can be cleared only by software (writing a 1 to the corresponding bit position) when the condition that caused the setting is no longer valid. Every flag has an associated interrupt enable bit in the CANRIER register.

**Figure 11-7. MSCAN Receiver Flag Register (CANRFLG)**

**NOTE**

The CANRFLG register is held in the reset state<sup>1</sup> when the initialization mode is active (INITRQ = 1 and INITAK = 1). This register is writable again as soon as the initialization mode is exited (INITRQ = 0 and INITAK = 0).

Read: Anytime

Write: Anytime when out of initialization mode, except RSTAT[1:0] and TSTAT[1:0] flags that are read-only; write of 1 clears flag; write of 0 is ignored.

**Table 11-9. CANRFLG Register Field Descriptions**

Field	Description
7 WUPIF	<b>Wake-Up Interrupt Flag</b> — If the MSCAN detects CAN bus activity while in sleep mode (see <a href="#">Section 11.5.5.4, "MSCAN Sleep Mode,"</a> ) and WUPE = 1 in CANTCTL0 (see <a href="#">Section 11.3.1, "MSCAN Control Register 0 (CANCTL0)"</a> ), the module sets WUPIF. If not masked, a wake-up interrupt is pending while this flag is set. 0 No wake-up activity observed while in sleep mode 1 MSCAN detected activity on the CAN bus and requested wake-up
6 CSCIF	<b>CAN Status Change Interrupt Flag</b> — This flag is set when the MSCAN changes its current CAN bus status due to the actual value of the transmit error counter (TEC) and the receive error counter (REC). An additional 4-bit (RSTAT[1:0], TSTAT[1:0]) status register, which is split into separate sections for TEC/REC, informs the system on the actual CAN bus status (see <a href="#">Section 11.3.6, "MSCAN Receiver Interrupt Enable Register (CANRIER)"</a> ). If not masked, an error interrupt is pending while this flag is set. CSCIF provides a blocking interrupt. That guarantees that the receiver/transmitter status bits (RSTAT/TSTAT) are only updated when no CAN status change interrupt is pending. If the TECs/RECs change their current value after the CSCIF is asserted, which would cause an additional state change in the RSTAT/TSTAT bits, these bits keep their status until the current CSCIF interrupt is cleared again. 0 No change in CAN bus status occurred since last interrupt 1 MSCAN changed current CAN bus status
5:4 RSTAT[1:0]	<b>Receiver Status Bits</b> — The values of the error counters control the actual CAN bus status of the MSCAN. As soon as the status change interrupt flag (CSCIF) is set, these bits indicate the appropriate receiver related CAN bus status of the MSCAN. The coding for the bits RSTAT1, RSTAT0 is: 00 RxOK: $0 \leq$ receive error counter $\leq 96$ 01 RxWRN: $96 <$ receive error counter $\leq 127$ 10 RxERR: $127 <$ receive error counter 11 Bus-off <sup>1</sup> : transmit error counter $> 255$
3:2 TSTAT[1:0]	<b>Transmitter Status Bits</b> — The values of the error counters control the actual CAN bus status of the MSCAN. As soon as the status change interrupt flag (CSCIF) is set, these bits indicate the appropriate transmitter related CAN bus status of the MSCAN. The coding for the bits TSTAT1, TSTAT0 is: 00 TxOK: $0 \leq$ transmit error counter $\leq 96$ 01 TxWRN: $96 <$ transmit error counter $\leq 127$ 10 TxERR: $127 <$ transmit error counter $\leq 255$ 11 Bus-Off: transmit error counter $> 255$

1. The RSTAT[1:0], TSTAT[1:0] bits are not affected by initialization mode.

**Table 11-9. CANRFLG Register Field Descriptions (continued)**

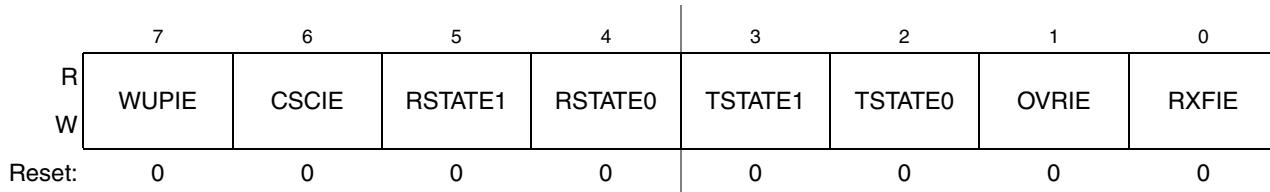
Field	Description
1 OVRIF	<b>Overrun Interrupt Flag</b> — This flag is set when a data overrun condition occurs. If not masked, an error interrupt is pending while this flag is set. 0 No data overrun condition 1 A data overrun detected
0 RXF <sup>2</sup>	<b>Receive Buffer Full Flag</b> — RXF is set by the MSCAN when a new message is shifted in the receiver FIFO. This flag indicates whether the shifted buffer is loaded with a correctly received message (matching identifier, matching cyclic redundancy code (CRC) and no other errors detected). After the CPU has read that message from the RxFG buffer in the receiver FIFO, the RXF flag must be cleared to release the buffer. A set RXF flag prohibits the shifting of the next FIFO entry into the foreground buffer (RxFG). If not masked, a receive interrupt is pending while this flag is set. 0 No new message available within the RxFG 1 The receiver FIFO is not empty. A new message is available in the RxFG.

<sup>1</sup> Redundant Information for the most critical CAN bus status that is bus-off. This only occurs if the Tx error counter exceeds a number of 255 errors. Bus-off affects the receiver state. As soon as the transmitter leaves its bus-off state the receiver state skips to RxOK too. Refer also to TSTAT[1:0] coding in this register.

<sup>2</sup> To ensure data integrity, do not read the receive buffer registers while the RXF flag is cleared. For MCUs with dual CPUs, reading the receive buffer registers while the RXF flag is cleared may result in a CPU fault condition.

### 11.3.6 MSCAN Receiver Interrupt Enable Register (CANRIER)

This register contains the interrupt enable bits for the interrupt flags described in the CANRFLG register.

**Figure 11-8. MSCAN Receiver Interrupt Enable Register (CANRIER)**

#### NOTE

The CANRIER register is held in the reset state when the initialization mode is active (INITRQ=1 and INITAK=1). This register is writable when not in initialization mode (INITRQ=0 and INITAK=0).

The RSTATE[1:0], TSTATE[1:0] bits are not affected by initialization mode.

Read: Anytime

Write: Anytime when not in initialization mode

**Table 11-10. CANIER Register Field Descriptions**

Field	Description
7 WUPIE <sup>1</sup>	Wake-Up Interrupt Enable 0 No interrupt request is generated from this event. 1 A wake-up event causes a Wake-Up interrupt request.
6 CSCIE	CAN Status Change Interrupt Enable 0 No interrupt request is generated from this event. 1 A CAN Status Change event causes an error interrupt request.
5:4 RSTATE[1:0]	<b>Receiver Status Change Enable</b> — These RSTAT enable bits control the sensitivity level in which receiver state changes are causing CSCIF interrupts. Independent of the chosen sensitivity level the RSTAT flags continue to indicate the actual receiver state and are only updated if no CSCIF interrupt is pending. 00 Do not generate any CSCIF interrupt caused by receiver state changes. 01 Generate CSCIF interrupt only if the receiver enters or leaves bus-off state. Discard other receiver state changes for generating CSCIF interrupt. 10 Generate CSCIF interrupt only if the receiver enters or leaves RxErr or bus-off <sup>2</sup> state. Discard other receiver state changes for generating CSCIF interrupt. 11 Generate CSCIF interrupt on all state changes.
3:2 TSTATE[1:0]	<b>Transmitter Status Change Enable</b> — These TSTAT enable bits control the sensitivity level in which transmitter state changes are causing CSCIF interrupts. Independent of the chosen sensitivity level, the TSTAT flags continue to indicate the actual transmitter state and are only updated if no CSCIF interrupt is pending. 00 Do not generate any CSCIF interrupt caused by transmitter state changes. 01 Generate CSCIF interrupt only if the transmitter enters or leaves bus-off state. Discard other transmitter state changes for generating CSCIF interrupt. 10 Generate CSCIF interrupt only if the transmitter enters or leaves TxErr or bus-off state. Discard other transmitter state changes for generating CSCIF interrupt. 11 Generate CSCIF interrupt on all state changes.
1 OVRIE	Overrun Interrupt Enable 0 No interrupt request is generated from this event. 1 An overrun event causes an error interrupt request.
0 RXFIE	Receiver Full Interrupt Enable 0 No interrupt request is generated from this event. 1 A receive buffer full (successful message reception) event causes a receiver interrupt request.

<sup>1</sup> WUPIE and WUPE (see [Section 11.3.1, “MSCAN Control Register 0 \(CANCTL0\)”](#)) must be enabled if the recovery mechanism from stop or wait is required.

<sup>2</sup> Bus-off state is defined by the CAN standard (see Bosch CAN 2.0A/B protocol specification: for only transmitters. Because the only possible state change for the transmitter from bus-off to TxOK also forces the receiver to skip its current state to RxOK, the coding of the RXSTAT[1:0] flags define an additional bus-off state for the receiver (see [Section 11.3.5, “MSCAN Receiver Flag Register \(CANRFLG\)”](#)).

### 11.3.7 MSCAN Transmitter Flag Register (CANTFLG)

The transmit buffer empty flags each have an associated interrupt enable bit in the CANTIER register.

R	7	6	5	4		3	2	1	0
W	0	0	0	0		0	TXE2	TXE1	TXE0
Reset:	0	0	0	0		0	1	1	1
	[  ]	= Unimplemented							

**Figure 11-9. MSCAN Transmitter Flag Register (CANTFLG)****NOTE**

The CANTFLG register is held in the reset state when the initialization mode is active ( $\text{INITRQ} = 1$  and  $\text{INITAK} = 1$ ). This register is writable when not in initialization mode ( $\text{INITRQ} = 0$  and  $\text{INITAK} = 0$ ).

Read: Anytime

Write: Anytime for TXEx flags when not in initialization mode; write of 1 clears flag, write of 0 is ignored.

**Table 11-11. CANTFLG Register Field Descriptions**

Field	Description
2:0 TXE[2:0]	<b>Transmitter Buffer Empty</b> — This flag indicates that the associated transmit message buffer is empty, and thus not scheduled for transmission. The CPU must clear the flag after a message is set up in the transmit buffer and is due for transmission. The MSCAN sets the flag after the message is sent successfully. The flag is also set by the MSCAN when the transmission request is successfully aborted due to a pending abort request (see <a href="#">Section 11.3.9, “MSCAN Transmitter Message Abort Request Register (CANTARQ)”</a> ). If not masked, a transmit interrupt is pending while this flag is set. Clearing a TXEx flag also clears the corresponding ABTAKx (see <a href="#">Section 11.3.10, “MSCAN Transmitter Message Abort Acknowledge Register (CANTAAK)”</a> ). When a TXEx flag is set, the corresponding ABTRQx bit is cleared (see <a href="#">Section 11.3.9, “MSCAN Transmitter Message Abort Request Register (CANTARQ)”</a> ). When listen-mode is active (see <a href="#">Section 11.3.2, “MSCAN Control Register 1 (CANCTL1)”</a> ) the TXEx flags cannot be cleared and no transmission is started. Read and write accesses to the transmit buffer are blocked, if the corresponding TXEx bit is cleared (TXEx = 0) and the buffer is scheduled for transmission. 0 The associated message buffer is full (loaded with a message pending or in transmission) 1 The associated message buffer is empty (not scheduled)

### 11.3.8 MSCAN Transmitter Interrupt Enable Register (CANTIER)

This register contains the interrupt enable bits for the transmit buffer empty interrupt flags.

R	7	6	5	4		3	2	1	0
W	0	0	0	0		0	TXEIE2	TXEIE1	TXEIE0
Reset:	0	0	0	0		0	0	0	0
	[  ]	= Unimplemented							

**Figure 11-10. MSCAN Transmitter Interrupt Enable Register (CANTIER)**

**NOTE**

The CANTIER register is held in the reset state when the initialization mode is active (INITRQ = 1 and INITAK = 1). This register is writable when not in initialization mode (INITRQ = 0 and INITAK = 0).

Read: Anytime

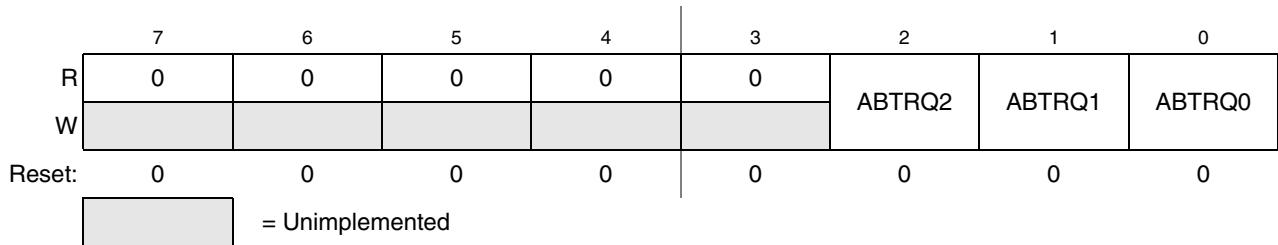
Write: Anytime when not in initialization mode

**Table 11-12. CANTIER Register Field Descriptions**

Field	Description
2:0 TXEIE[2:0]	Transmitter Empty Interrupt Enable 0 No interrupt request is generated from this event. 1 A transmitter empty (transmit buffer available for transmission) event causes a transmitter empty interrupt request. See <a href="#">Section 11.5.2.2, "Transmit Structures"</a> for details.

**11.3.9 MSCAN Transmitter Message Abort Request Register (CANTARQ)**

The CANTARQ register allows abort request of messages queued for transmission.

**Figure 11-11. MSCAN Transmitter Message Abort Request Register (CANTARQ)****NOTE**

The CANTARQ register is held in the reset state when the initialization mode is active (INITRQ = 1 and INITAK = 1). This register is writable when not in initialization mode (INITRQ = 0 and INITAK = 0).

Read: Anytime

Write: Anytime when not in initialization mode

**Table 11-13. CANTARQ Register Field Descriptions**

Field	Description
2:0 ABTRQ[2:0]	<b>Abort Request</b> — The CPU sets the ABTRQx bit to request that a scheduled message buffer (TXEx = 0) be aborted. The MSCAN grants the request if the message has not already started transmission, or if the transmission is not successful (lost arbitration or error). When a message is aborted, the associated TXE (see <a href="#">Section 11.3.7, "MSCAN Transmitter Flag Register (CANTFLG)"</a> ) and abort acknowledge flags (ABTAK, see <a href="#">Section 11.3.10, "MSCAN Transmitter Message Abort Acknowledge Register (CANTAAK)"</a> ) are set and a transmit interrupt occurs if enabled. The CPU cannot reset ABTRQx. ABTRQx is reset when the associated TXE flag is set. 0 No abort request 1 Abort request pending

### 11.3.10 MSCAN Transmitter Message Abort Acknowledge Register (CANTAAK)

The CANTAAK register indicates the successful abort of messages queued for transmission, if requested by the appropriate bits in the CANTARQ register.

	7	6	5	4		3	2	1	0
R	0	0	0	0		0	ABTAK2	ABTAK1	ABTAK0
W									
Reset:	0	0	0	0		0	0	0	0
	= Unimplemented								

Figure 11-12. MSCAN Transmitter Message Abort Acknowledge Register (CANTAAK)

#### NOTE

The CANTAAK register is held in the reset state when the initialization mode is active (INITRQ = 1 and INITAK = 1).

Read: Anytime

Write: Unimplemented for ABTAKx flags

Table 11-14. CANTAAK Register Field Descriptions

Field	Description
2:0 ABTAK[2:0]	<b>Abort Acknowledge</b> — This flag acknowledges that a message was aborted due to a pending transmission abort request from the CPU. After a particular message buffer is flagged empty, this flag can be used by the application software to identify whether the message was aborted successfully or was sent anyway. The ABTAKx flag is cleared when the corresponding TXE flag is cleared. 0 The message was not aborted. 1 The message was aborted.

### 11.3.11 MSCAN Transmit Buffer Selection Register (CANTBSEL)

The CANTBSEL selections of the actual transmit message buffer, which is accessible in the CANTXFG register space.

	7	6	5	4		3	2	1	0
R	0	0	0	0		0	TX2	TX1	TX0
W									
Reset:	0	0	0	0		0	0	0	0
	= Unimplemented								

Figure 11-13. MSCAN Transmit Buffer Selection Register (CANTBSEL)

#### NOTE

The CANTBSEL register is held in the reset state when the initialization mode is active (INITRQ = 1 and INITAK = 1). This register is writable when not in initialization mode (INITRQ = 0 and INITAK = 0).

Read: Find the lowest ordered bit set to 1, all other bits is read as 0

Write: Anytime when not in initialization mode

**Table 11-15. CANTBSEL Register Field Descriptions**

Field	Description
2:0 TX[2:0]	<p><b>Transmit Buffer Select</b> — The lowest numbered bit places the respective transmit buffer in the CANTXFG register space (e.g., TX1 = 1 and TX0 = 1 selects transmit buffer TX0; TX1 = 1 and TX0 = 0 selects transmit buffer TX1). Read and write accesses to the selected transmit buffer is blocked, if the corresponding TXEx bit is cleared and the buffer is scheduled for transmission (see <a href="#">Section 11.3.7, “MSCAN Transmitter Flag Register (CANTFLG)“</a>).</p> <p>0 The associated message buffer is deselected 1 The associated message buffer is selected, if lowest numbered bit</p>

The following gives a short programming example of the usage of the CANTBSEL register:

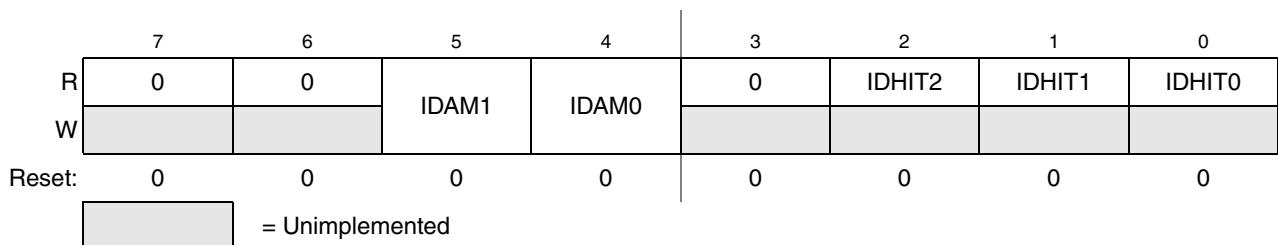
To get the next available transmit buffer, application software must read the CANTFLG register and write this value back into the CANTBSEL register. In this example Tx buffers TX1 and TX2 are available. The value read from CANTFLG is therefore 0b0000\_0110. When writing this value back to CANTBSEL, the Tx buffer TX1 is selected in the CANTXFG because the lowest numbered bit set to 1 is at bit position 1. Reading back this value out of CANTBSEL results in 0b0000\_0010, because only the lowest numbered bit position set to 1 is presented. This mechanism eases the application software the selection of the next available Tx buffer.

- LDD CANTFLG; value read is 0b0000\_0110
- STD CANTBSEL; value written is 0b0000\_0110
- LDD CANTBSEL; value read is 0b0000\_0010

If all transmit message buffers are deselected, no accesses are allowed to the CANTXFG buffer register.

### 11.3.12 MSCAN Identifier Acceptance Control Register (CANIDAC)

The CANIDAC register is used for identifier filter acceptance control as described below.



**Figure 11-14. MSCAN Identifier Acceptance Control Register (CANIDAC)**

Read: Anytime

Write: Anytime in initialization mode (INITRQ = 1 and INITAK = 1), except bits IDHITx, which are read-only

**Table 11-16. CANIDAC Register Field Descriptions**

Field	Description
5:4 IDAM[1:0]	<b>Identifier Acceptance Mode</b> — The CPU sets these flags to define the identifier acceptance filter organization (see <a href="#">Section 11.5.2.4, “Identifier Acceptance Filter”</a> ). <a href="#">Table 11-17</a> summarizes the different settings. In filter closed mode, no message is accepted such that the foreground buffer is never reloaded.
2:0 IDHIT[2:0]	Identifier Acceptance Hit Indicator — The MSCAN sets these flags to indicate an identifier acceptance hit (see <a href="#">Section 11.5.2.4, “Identifier Acceptance Filter”</a> ). <a href="#">Table 11-18</a> summarizes the different settings.

**Table 11-17. Identifier Acceptance Mode Settings**

IDAM1	IDAM0	Identifier Acceptance Mode
0	0	Two 32-bit acceptance filters
0	1	Four 16-bit acceptance filters
1	0	Eight 8-bit acceptance filters
1	1	Filter closed

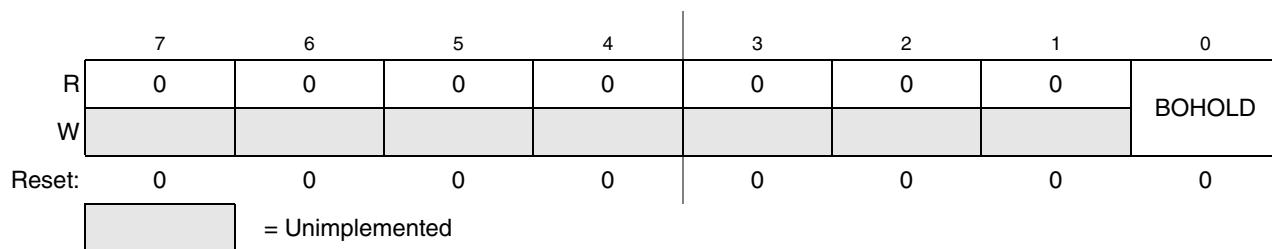
**Table 11-18. Identifier Acceptance Hit Indication**

IDHIT2	IDHIT1	IDHITO	Identifier Acceptance Hit
0	0	0	Filter 0 hit
0	0	1	Filter 1 hit
0	1	0	Filter 2 hit
0	1	1	Filter 3 hit
1	0	0	Filter 4 hit
1	0	1	Filter 5 hit
1	1	0	Filter 6 hit
1	1	1	Filter 7 hit

The IDHITx indicators are always related to the message in the foreground buffer (RxFG). When a message gets shifted into the foreground buffer of the receiver FIFO the indicators are updated as well.

### 11.3.13 MSCAN Miscellaneous Register (CANMISC)

This register provides additional features.



**Figure 11-15. MSCAN Miscellaneous Register (CANMISC)**

Read: Anytime

Write: Anytime; write of 1 clears flag; write of 0 ignored

**Table 11-19. CANMISC Register Field Descriptions**

Field	Description
0 BOHOLD	<b>Bus-off State Hold Until User Request</b> — If BORM is set in <a href="#">Section 11.3.2, “MSCAN Control Register 1 (CANCTL1)</a> , this bit indicates whether the module has entered the bus-off state. Clearing this bit requests the recovery from bus-off. Refer to <a href="#">Section 11.6.2, “Bus-Off Recovery,”</a> for details. 0 Module is not bus-off or recovery has been requested by user in bus-off state 1 Module is bus-off and holds this state until user request

### 11.3.14 MSCAN Receive Error Counter (CANRXERR)

This register reflects the status of the MSCAN receive error counter.

	7	6	5	4		3	2	1	0
R	RXERR7	RXERR6	RXERR5	RXERR4		RXERR3	RXERR2	RXERR1	RXERR0
W									
Reset:	0	0	0	0		0	0	0	0
	[ ]	= Unimplemented							

**Figure 11-16. MSCAN Receive Error Counter (CANRXERR)**

Read: Only when in sleep mode (SLPRQ = 1 and SLPAK = 1) or initialization mode (INITRQ = 1 and INITAK = 1)

Write: Unimplemented

#### NOTE

Reading this register when in any other mode other than sleep or initialization mode may return an incorrect value. For MCUs with dual CPUs, this may result in a CPU fault condition.

Writing to this register when in special test modes can alter the MSCAN functionality.

### 11.3.15 MSCAN Transmit Error Counter (CANTXERR)

This register reflects the status of the MSCAN transmit error counter.

	7	6	5	4		3	2	1	0
R	TXERR7	TXERR6	TXERR5	TXERR4		TXERR3	TXERR2	TXERR1	TXERR0
W									
Reset:	0	0	0	0		0	0	0	0
	[ ]	= Unimplemented							

**Figure 11-17. MSCAN Transmit Error Counter (CANTXERR)**

Read: Only when in sleep mode ( $\text{SLPRQ} = 1$  and  $\text{SLPAK} = 1$ ) or initialization mode ( $\text{INITRQ} = 1$  and  $\text{INITAK} = 1$ )

Write: Unimplemented

#### NOTE

Reading this register when in any other mode other than sleep or initialization mode, may return an incorrect value. For MCUs with dual CPUs, this may result in a CPU fault condition.

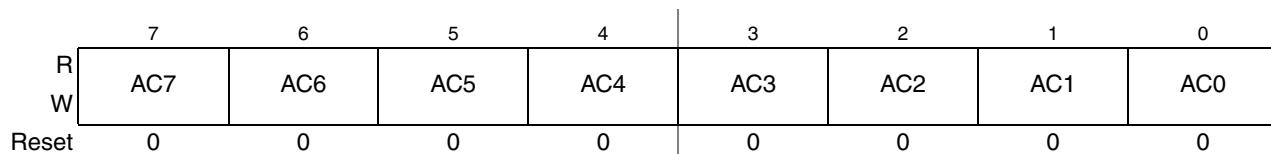
Writing to this register when in special test modes can alter the MSCAN functionality.

### 11.3.16 MSCAN Identifier Acceptance Registers (CANIDAR0-7)

On reception, each message is written into the background receive buffer. The CPU is only signalled to read the message if it passes the criteria in the identifier acceptance and identifier mask registers (accepted); otherwise, the message is overwritten by the next message (dropped).

The acceptance registers of the MSCAN are applied on the IDR0-IDR3 registers (see [Section 11.4.1, “Identifier Registers \(IDR0-IDR3\)](#)) of incoming messages in a bit by bit manner (see [Section 11.5.2.4, “Identifier Acceptance Filter”](#)).

For extended identifiers, all four acceptance and mask registers are applied. For standard identifiers, only the first two (CANIDAR0/1, CANIDMR0/1) are applied.



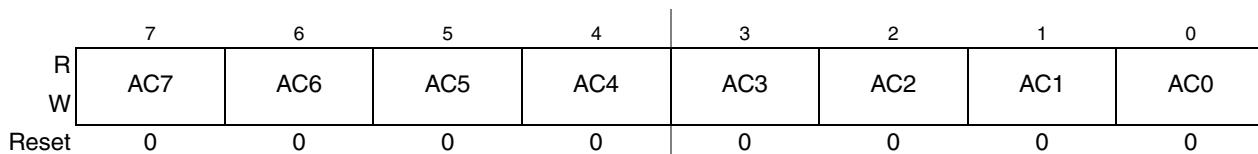
**Figure 11-18. MSCAN Identifier Acceptance Registers (First Bank) — CANIDAR0—CANIDAR3**

Read: Anytime

Write: Anytime in initialization mode ( $\text{INITRQ} = 1$  and  $\text{INITAK} = 1$ )

**Table 11-20. CANIDAR0—CANIDAR3 Register Field Descriptions**

Field	Description
7:0 AC[7:0]	<b>Acceptance Code Bits</b> — AC[7:0] comprise a user-defined sequence of bits with which the corresponding bits of the related identifier register (IDRn) of the receive message buffer are compared. The result of this comparison is then masked with the corresponding identifier mask register.



**Figure 11-19. MSCAN Identifier Acceptance Registers (Second Bank) — CANIDAR4—CANIDAR7**

Read: Anytime

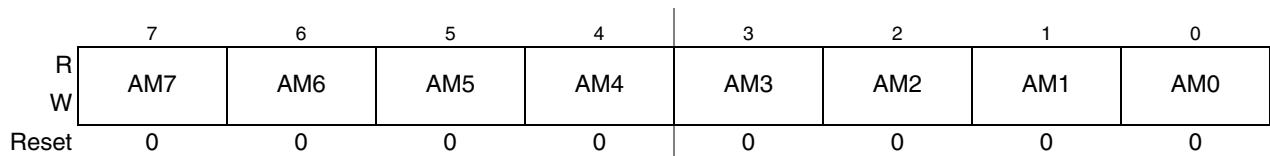
Write: Anytime in initialization mode (INITRQ = 1 and INITAK = 1)

**Table 11-21. CANIDAR4–CANIDAR7 Register Field Descriptions**

Field	Description
7:0 AC[7:0]	<b>Acceptance Code Bits</b> — AC[7:0] comprise a user-defined sequence of bits with which the corresponding bits of the related identifier register (IDRn) of the receive message buffer are compared. The result of this comparison is then masked with the corresponding identifier mask register.

### 11.3.17 MSCAN Identifier Mask Registers (CANIDMR0–CANIDMR7)

The identifier mask register specifies which of the corresponding bits in the identifier acceptance register are relevant for acceptance filtering. To receive standard identifiers in 32 bit filter mode, it is required to program the last three bits (AM[2:0]) in the mask registers CANIDMR1 and CANIDMR5 to don't care. To receive standard identifiers in 16 bit filter mode, it is required to program the last three bits (AM[2:0]) in the mask registers CANIDMR1, CANIDMR3, CANIDMR5, and CANIDMR7 to don't care.



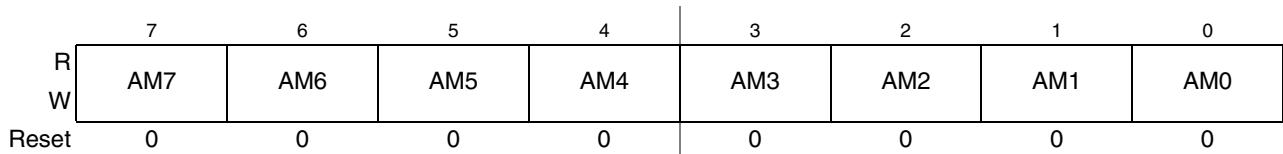
**Figure 11-20. MSCAN Identifier Mask Registers (First Bank) — CANIDMR0–CANIDMR3**

Read: Anytime

Write: Anytime in initialization mode (INITRQ = 1 and INITAK = 1)

**Table 11-22. CANIDMR0–CANIDMR3 Register Field Descriptions**

Field	Description
7:0 AM[7:0]	<b>Acceptance Mask Bits</b> — If a particular bit in this register is cleared, this indicates that the corresponding bit in the identifier acceptance register must be the same as its identifier bit before a match is detected. The message is accepted if all such bits match. If a bit is set, it indicates that the state of the corresponding bit in the identifier acceptance register does not affect whether or not the message is accepted. 0 Match corresponding acceptance code register and identifier bits 1 Ignore corresponding acceptance code register bit (don't care)



**Figure 11-21. MSCAN Identifier Mask Registers (Second Bank) — CANIDMR4–CANIDMR7**

Read: Anytime

Write: Anytime in initialization mode (INITRQ = 1 and INITAK = 1)

**Table 11-23. CANIDMR4–CANIDMR7 Register Field Descriptions**

Field	Description
7:0 AM[7:0]	<b>Acceptance Mask Bits</b> — If a particular bit in this register is cleared, this indicates that the corresponding bit in the identifier acceptance register must be the same as its identifier bit before a match is detected. The message is accepted if all such bits match. If a bit is set, it indicates that the state of the corresponding bit in the identifier acceptance register does not affect whether or not the message is accepted. 0 Match corresponding acceptance code register and identifier bits 1 Ignore corresponding acceptance code register bit (don't care)

## 11.4 Programmer's Model of Message Storage

The following section details the organization of the receive and transmit message buffers and the associated control registers.

To simplify the programmer interface, the receive and transmit message buffers have the same outline. Each message buffer allocates 16 bytes in the memory map containing a 13 byte data structure.

An additional transmit buffer priority register (TBPR) is defined for the transmit buffers. Within the last two bytes of this memory map, the MSCAN stores a special 16-bit time stamp, which is sampled from an internal timer after successful transmission or reception of a message. This feature is only available for transmit and receiver buffers if the TIME bit is set (see [Section 11.3.1, “MSCAN Control Register 0 \(CANCTL0\)”](#)).

The time stamp register is written by the MSCAN. The CPU can only read these registers.

**Table 11-24. Message Buffer Organization**

Offset Address	Register	Access
0x00X0	Identifier Register 0	
0x00X1	Identifier Register 1	
0x00X2	Identifier Register 2	
0x00X3	Identifier Register 3	
0x00X4	Data Segment Register 0	
0x00X5	Data Segment Register 1	
0x00X6	Data Segment Register 2	
0x00X7	Data Segment Register 3	
0x00X8	Data Segment Register 4	
0x00X9	Data Segment Register 5	
0x00XA	Data Segment Register 6	
0x00XB	Data Segment Register 7	
0x00XC	Data Length Register	
0x00XD	Transmit Buffer Priority Register <sup>1</sup>	

**Table 11-24. Message Buffer Organization**

Offset Address	Register	Access
0x00XE	Time Stamp Register (High Byte) <sup>2</sup>	
0x00XF	Time Stamp Register (Low Byte) <sup>3</sup>	

<sup>1</sup> Not applicable for receive buffers<sup>2</sup> Read-only for CPU<sup>3</sup> Read-only for CPU

[Figure 11-22](#) shows the common 13-byte data structure of receive and transmit buffers for extended identifiers. The mapping of standard identifiers into the IDR registers is shown in [Figure 11-23](#).

All bits of the receive and transmit buffers are x out of reset because of RAM-based implementation<sup>1</sup>. All reserved or unused bits of the receive and transmit buffers always read x.

Register Name	Bit 7	6	5	4	3	2	1	Bit0	
IDR0	R W	ID28	ID27	ID26	ID25	ID24	ID23	ID22	ID21
IDR1	R W	ID20	ID19	ID18	SRR <sup>(1)</sup>	IDE <sup>(1)</sup>	ID17	ID16	ID15
IDR2	R	ID14	ID13	ID12	ID11	ID10	ID9	ID8	ID7
IDR3	R W	ID6	ID5	ID4	ID3	ID2	ID1	ID0	RTR <sup>2</sup>
DSR0	R	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
DSR1	R W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
DSR2	R	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0

**Figure 11-22. Receive/Transmit Message Buffer — Extended Identifier Mapping**

1. Exception: The transmit priority registers are 0 out of reset.

Register Name	Bit 7	6	5	4	3	2	1	Bit0	
DSR3	R W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
DSR4	R W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
DSR5	R W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
DSR6	R W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
DSR7	R W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
DLR	R W					DLC3	DLC2	DLC1	DLC0

= Unused, always read x

**Figure 11-22. Receive/Transmit Message Buffer — Extended Identifier Mapping**

<sup>1</sup> SRR and IDE are 1s.

<sup>2</sup> The position of RTR differs between extended and standard identifier mapping.

Read: For transmit buffers, anytime when TXEx flag is set (see [Section 11.3.7, “MSCAN Transmitter Flag Register \(CANTFLG\)”](#)) and the corresponding transmit buffer is selected in CANTBSEL (see [Section 11.3.11, “MSCAN Transmit Buffer Selection Register \(CANTBSEL\)”](#)). For receive buffers, only when RXF flag is set (see [Section 11.3.5, “MSCAN Receiver Flag Register \(CANRFLG\)”](#)).

Write: For transmit buffers, anytime when TXEx flag is set (see [Section 11.3.7, “MSCAN Transmitter Flag Register \(CANTFLG\)”](#)) and the corresponding transmit buffer is selected in CANTBSEL (see [Section 11.3.11, “MSCAN Transmit Buffer Selection Register \(CANTBSEL\)”](#)). Unimplemented for receive buffers.

Reset: Undefined (0x00XX) because of RAM-based implementation

Register Name	Bit 7	6	5	4	3	2	1	Bit 0	
IDR0	R W	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3
IDR1	R W	ID2	ID1	ID0	RTR <sup>1</sup>	IDE <sup>2</sup>			
IDR2	R W								
IDR3	R W								

= Unused, always read x

**Figure 11-23. Receive/Transmit Message Buffer — Standard Identifier Mapping**

<sup>1</sup> The position of RTR differs between extended and standard identifier mapping.

<sup>2</sup> IDE is 0.

### 11.4.1 Identifier Registers (IDR0–IDR3)

The identifier registers for an extended format identifier consist of a total of 32 bits; ID[28:0], SRR, IDE, and RTR bits. The identifier registers for a standard format identifier consist of a total of 13 bits; ID[10:0], RTR, and IDE bits.

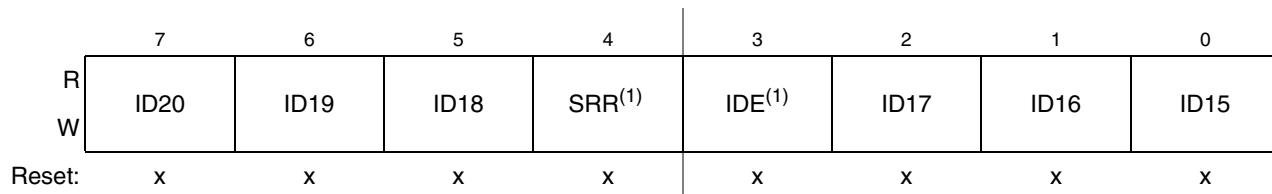
#### 11.4.1.1 IDR0–IDR3 for Extended Identifier Mapping

	7	6	5	4		3	2	1	0
R W	ID28	ID27	ID26	ID25		ID24	ID23	ID22	ID21
Reset:	x	x	x	x		x	x	x	x

**Figure 11-24. Identifier Register 0 (IDR0) — Extended Identifier Mapping**

**Table 11-25. IDR0 Register Field Descriptions — Extended**

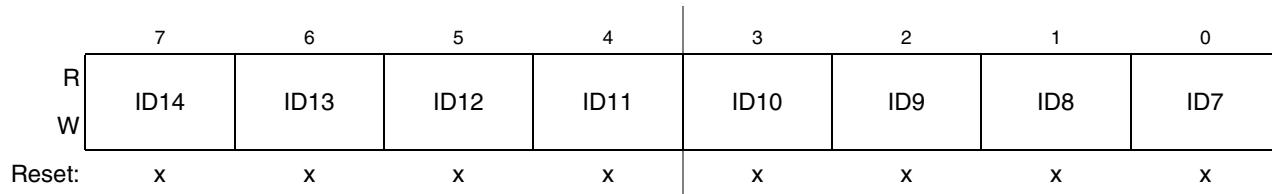
Field	Description
7:0 ID[28:21]	Extended Format Identifier — The identifiers consist of 29 bits (ID[28:0]) for the extended format. ID28 is the most significant bit and is transmitted first on the CAN bus during the arbitration procedure. The priority of an identifier is defined to be highest for the smallest binary number.

**Figure 11-25. Identifier Register 1 (IDR1) — Extended Identifier Mapping**

<sup>1</sup> SRR and IDE are 1s.

**Table 11-26. IDR1 Register Field Descriptions — Extended**

Field	Description
7:5 ID[20:18]	<b>Extended Format Identifier</b> — The identifiers consist of 29 bits (ID[28:0]) for the extended format.
4 SRR	<b>Substitute Remote Request</b> — This fixed recessive bit is used only in extended format. It must be set to 1 by the user for transmission buffers and is stored as received on the CAN bus for receive buffers.
3 IDE	<b>ID Extended</b> — This flag indicates whether the extended or standard identifier format is applied in this buffer. In the case of a receive buffer, the flag is set as received and indicates to the CPU how to process the buffer identifier registers. In the case of a transmit buffer, the flag indicates to the MSCAN what type of identifier to send. 0 Standard format (11 bit) 1 Extended format (29 bit)
2:0 ID[17:15]	<b>Extended Format Identifier</b> — The identifiers consist of 29 bits (ID[28:0]) for the extended format.

**Figure 11-26. Identifier Register 2 (IDR2) — Extended Identifier Mapping****Table 11-27. IDR2 Register Field Descriptions — Extended**

Field	Description
7:0 ID[14:7]	<b>Extended Format Identifier</b> — The identifiers consist of 29 bits (ID[28:0]) for the extended format.

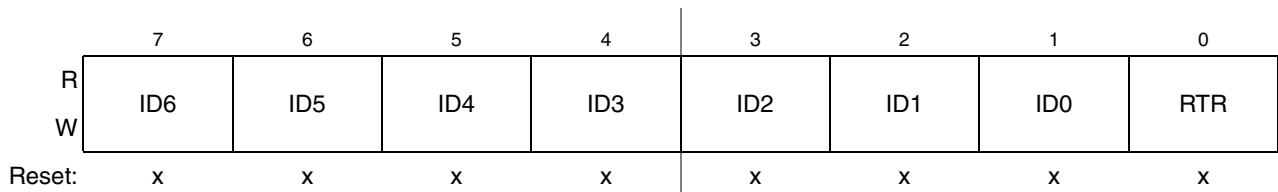


Figure 11-27. Identifier Register 3 (IDR3) — Extended Identifier Mapping

Table 11-28. IDR3 Register Field Descriptions — Extended

Field	Description
7:1 ID[6:0]	<b>Extended Format Identifier</b> — The identifiers consist of 29 bits (ID[28:0]) for the extended format.
0 RTR	<b>Remote Transmission Request</b> — This flag reflects the status of the remote transmission request bit in the CAN frame. In the case of a receive buffer, it indicates the status of the received frame and supports the transmission of an answering frame in software. In the case of a transmit buffer, this flag defines the setting of the RTR bit to be sent. 0 Data frame 1 Remote frame

## 11.4.2 IDR0–IDR3 for Standard Identifier Mapping

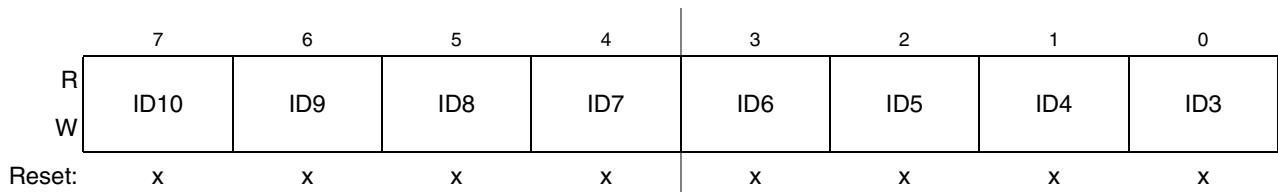
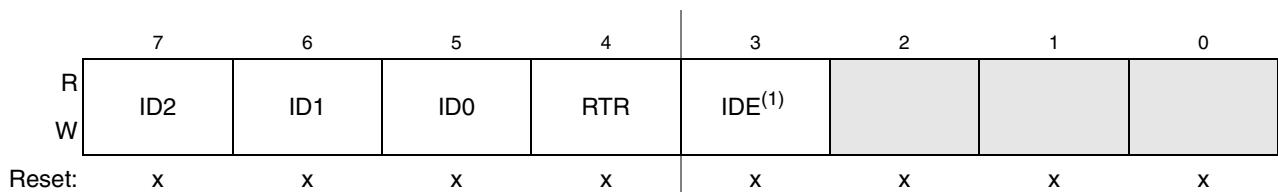


Figure 11-28. Identifier Register 0 — Standard Mapping

Table 11-29. IDR0 Register Field Descriptions — Standard

Field	Description
7:0 ID[10:3]	<b>Standard Format Identifier</b> — The identifiers consist of 11 bits (ID[10:0]) for the standard format. ID10 is the most significant bit and is transmitted first on the CAN bus during the arbitration procedure. The priority of an identifier is defined to be highest for the smallest binary number. See also ID bits in <a href="#">Table 11-30</a> .



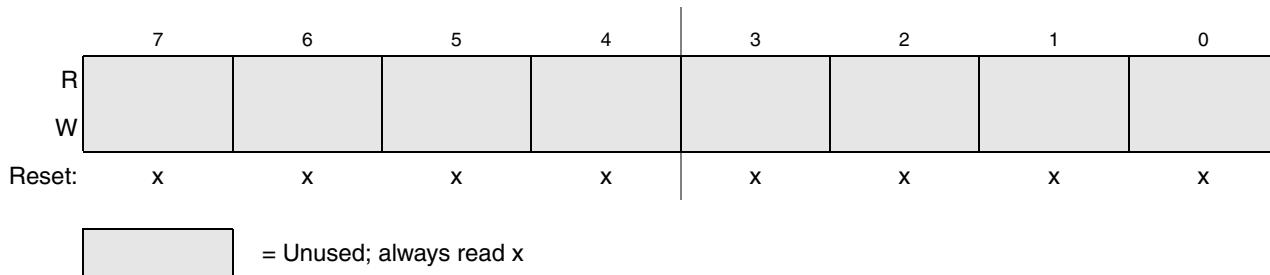
= Unused; always read x

Figure 11-29. Identifier Register 1 — Standard Mapping

<sup>1</sup> IDE is 0.

**Table 11-30. IDR1 Register Field Descriptions**

Field	Description
7:5 ID[2:0]	<b>Standard Format Identifier</b> — The identifiers consist of 11 bits (ID[10:0]) for the standard format. ID10 is the most significant bit and is transmitted first on the CAN bus during the arbitration procedure. The priority of an identifier is defined to be highest for the smallest binary number. See also ID bits in <a href="#">Table 11-29</a> .
4 RTR	<b>Remote Transmission Request</b> — This flag reflects the status of the Remote Transmission Request bit in the CAN frame. In the case of a receive buffer, it indicates the status of the received frame and supports the transmission of an answering frame in software. In the case of a transmit buffer, this flag defines the setting of the RTR bit to be sent. 0 Data frame 1 Remote frame
3 IDE	<b>ID Extended</b> — This flag indicates whether the extended or standard identifier format is applied in this buffer. In the case of a receive buffer, the flag is set as received and indicates to the CPU how to process the buffer identifier registers. In the case of a transmit buffer, the flag indicates to the MSCAN what type of identifier to send. 0 Standard format (11 bit) 1 Extended format (29 bit)

**Figure 11-30. Identifier Register 2 — Standard Mapping****Figure 11-31. Identifier Register 3 — Standard Mapping**

### 11.4.3 Data Segment Registers (DSR0-7)

The eight data segment registers, each with bits DB[7:0], contain the data to be transmitted or received. The number of bytes to be transmitted or received is determined by the data length code in the corresponding DLR register.

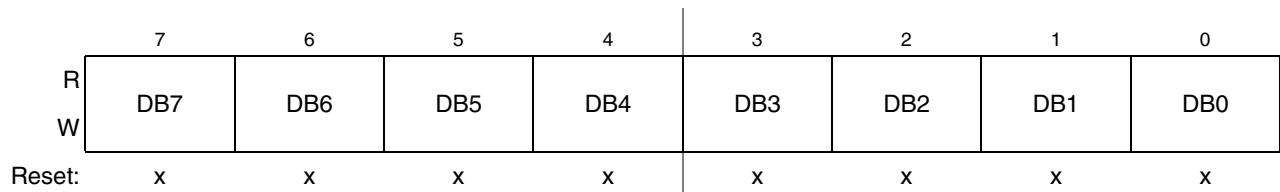


Figure 11-32. Data Segment Registers (DSR0–DSR7) — Extended Identifier Mapping

Table 11-31. DSR0–DSR7 Register Field Descriptions

Field	Description
7:0 DB[7:0]	Data bits 7:0

#### 11.4.4 Data Length Register (DLR)

This register keeps the data length field of the CAN frame.



= Unused; always read x

Figure 11-33. Data Length Register (DLR) — Extended Identifier Mapping

Table 11-32. DLR Register Field Descriptions

Field	Description
3:0 DLC[3:0]	<b>Data Length Code Bits</b> — The data length code contains the number of bytes (data byte count) of the respective message. During the transmission of a remote frame, the data length code is transmitted as programmed while the number of transmitted data bytes is always 0. The data byte count ranges from 0 to 8 for a data frame. <a href="#">Table 11-33</a> shows the effect of setting the DLC bits.

**Table 11-33. Data Length Codes**

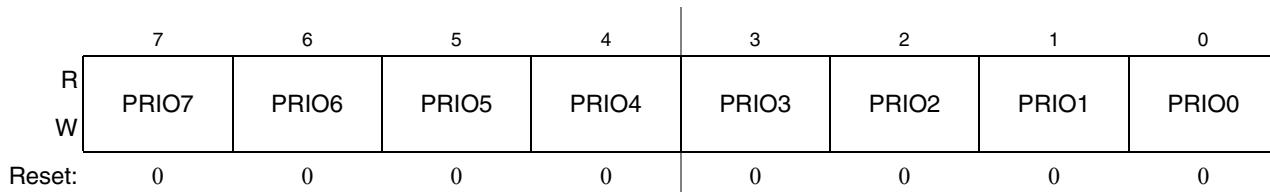
Data Length Code				Data Byte Count
DLC3	DLC2	DLC1	DLC0	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8

### 11.4.5 Transmit Buffer Priority Register (TBPR)

This register defines the local priority of the associated message transmit buffer. The local priority is used for the internal prioritization process of the MSCAN and is defined to be highest for the smallest binary number. The MSCAN implements the following internal prioritization mechanisms:

- All transmission buffers with a cleared TXEx flag participate in the prioritization immediately before the SOF (start of frame) is sent.
- The transmission buffer with the lowest local priority field wins the prioritization.

In cases of more than one buffer having the same lowest priority, the message buffer with the lower index number wins.

**Figure 11-34. Transmit Buffer Priority Register (TBPR)**

Read: Anytime when TXEx flag is set (see [Section 11.3.7, “MSCAN Transmitter Flag Register \(CANTFLG\)”](#)) and the corresponding transmit buffer is selected in CANTBSEL (see [Section 11.3.11, “MSCAN Transmit Buffer Selection Register \(CANTBSEL\)”](#)).

Write: Anytime when TXEx flag is set (see [Section 11.3.7, “MSCAN Transmitter Flag Register \(CANTFLG\)”](#)) and the corresponding transmit buffer is selected in CANTBSEL (see [Section 11.3.11, “MSCAN Transmit Buffer Selection Register \(CANTBSEL\)”](#)).

### 11.4.6 Time Stamp Register (TSRH–TSRL)

If the TIME bit is enabled, the MSCAN writes a time stamp to the respective registers in the active transmit or receive buffer as soon as a message has been acknowledged on the CAN bus (see [Section 11.3.1](#),

“MSCAN Control Register 0 (CANCTL0)”). The time stamp is written on the bit sample point for the recessive bit of the ACK delimiter in the CAN frame. In case of a transmission, the CPU can only read the time stamp after the respective transmit buffer has been flagged empty.

The timer value, which is used for stamping, is taken from a free running internal CAN bit clock. A timer overrun is not indicated by the MSCAN. The timer is reset (all bits set to 0) during initialization mode. The CPU can only read the time stamp registers.

	7	6	5	4		3	2	1	0
R	TSR15	TSR14	TSR13	TSR12		TSR11	TSR10	TSR9	TSR8
W									
Reset:	x	x	x	x		x	x	x	x

Figure 11-35. Time Stamp Register — High Byte (TSRH)

	7	6	5	4		3	2	1	0
R	TSR7	TSR6	TSR5	TSR4		TSR3	TSR2	TSR1	TSR0
W									
Reset:	x	x	x	x		x	x	x	x

Figure 11-36. Time Stamp Register — Low Byte (TSRL)

Read: Anytime when TXEx flag is set (see [Section 11.3.7, “MSCAN Transmitter Flag Register \(CANTFLG\)”](#)) and the corresponding transmit buffer is selected in CANTBSEL (see [Section 11.3.11, “MSCAN Transmit Buffer Selection Register \(CANTBSEL\)”](#)).

Write: Unimplemented

## 11.5 Functional Description

### 11.5.1 General

This section provides a complete functional description of the MSCAN. It describes each of the features and modes listed in the introduction.

## 11.5.2 Message Storage

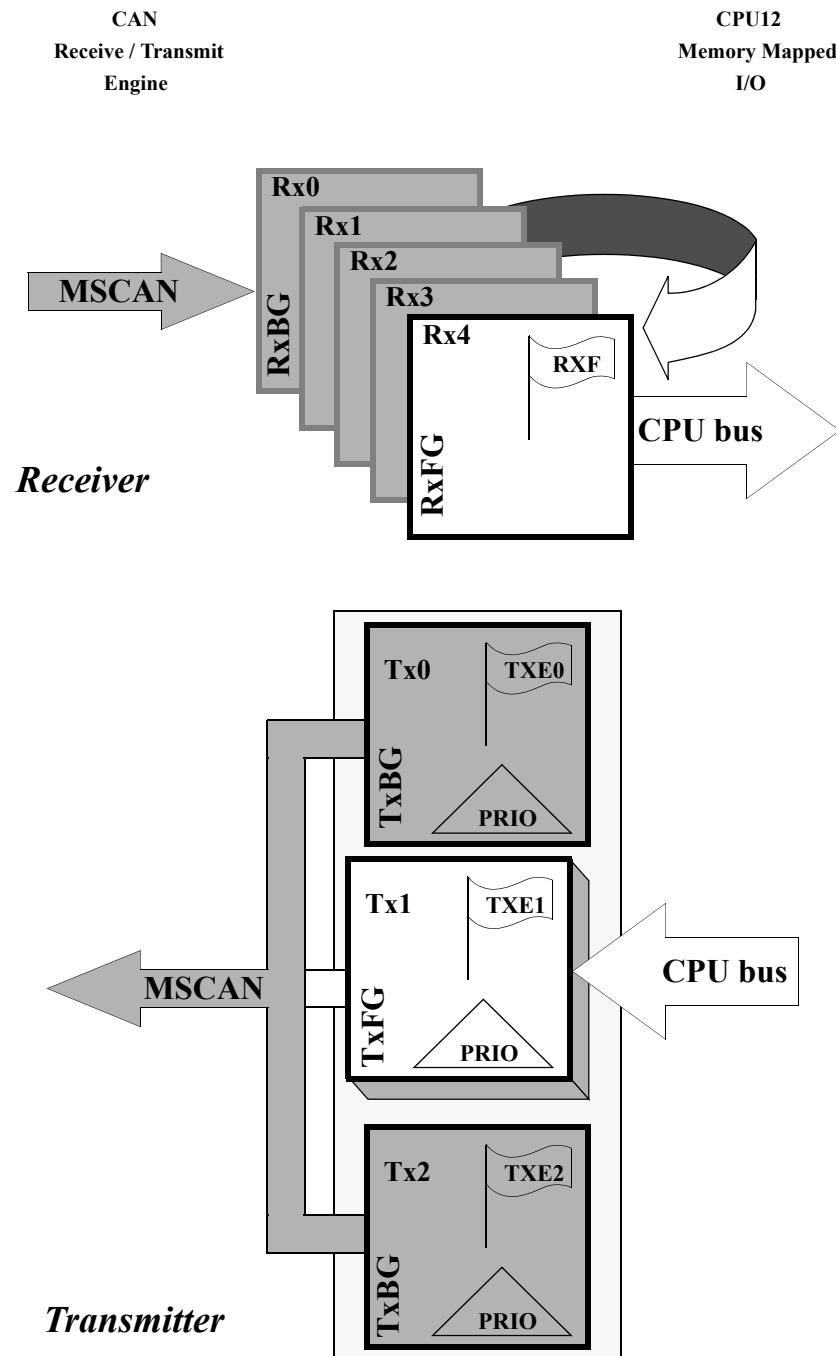


Figure 11-37. User Model for Message Buffer Organization

MSCAN facilitates a sophisticated message storage system that addresses the requirements of a broad range of network applications.

### 11.5.2.1 Message Transmit Background

Modern application layer software is built upon two fundamental assumptions:

- Any CAN node is able to send out a stream of scheduled messages without releasing the CAN bus between the two messages. Such nodes arbitrate for the CAN bus immediately after sending the previous message and only release the CAN bus in case of lost arbitration.
- The internal message queue within any CAN node is organized such that the highest priority message is sent out first, if more than one message is ready to be sent.

The behavior described in the bullets above cannot be achieved with a single transmit buffer. That buffer must be reloaded immediately after the previous message is sent. This loading process lasts a finite amount of time and must be completed within the inter-frame sequence (IFS) to be able to send an uninterrupted stream of messages. Even if this is feasible for limited CAN bus speeds, it requires that the CPU reacts with short latencies to the transmit interrupt.

A double buffer scheme de-couples the reloading of the transmit buffer from the actual message sending and, therefore, reduces the reactivity requirements of the CPU. Problems can arise if the sending of a message is finished while the CPU re-loads the second buffer. No buffer would then be ready for transmission, and the CAN bus would be released.

At least three transmit buffers are required to meet the first of the above requirements under all circumstances. The MSCAN has three transmit buffers.

The second requirement calls for some sort of internal prioritization that the MSCAN implements with the local priority concept described in [Section 11.5.2.2, “Transmit Structures”](#).

### 11.5.2.2 Transmit Structures

The MSCAN triple transmit buffer scheme optimizes real-time performance by allowing multiple messages to be set up in advance. The three buffers are arranged as shown in [Figure 11-37](#).

All three buffers have a 13-byte data structure similar to the outline of the receive buffers (see [Section 11.4, “Programmer’s Model of Message Storage”](#)). An additional [Section 11.4.5, “Transmit Buffer Priority Register \(TBPR\)](#) contains an 8-bit local priority field (PRIO) (see [Section 11.4.5, “Transmit Buffer Priority Register \(TBPR\)”](#)). The remaining two bytes are used for time stamping of a message, if required (see [Section 11.4.6, “Time Stamp Register \(TSRH–TSRL\)”](#)).

To transmit a message, the CPU must identify an available transmit buffer, which is indicated by a set transmitter buffer empty (TXEx) flag (see [Section 11.3.7, “MSCAN Transmitter Flag Register \(CANTFLG\)”](#)). If a transmit buffer is available, the CPU must set a pointer to this buffer by writing to the CANTBSEL register (see [Section 11.3.11, “MSCAN Transmit Buffer Selection Register \(CANTBSEL\)”](#)). This makes the respective buffer accessible within the CANTXFG address space (see [Section 11.4, “Programmer’s Model of Message Storage”](#)). The algorithmic feature associated with the CANTBSEL register simplifies the transmit buffer selection. In addition, this scheme makes the handler software simpler because only one address area is applicable for the transmit process, and the required address space is minimized.

The CPU then stores the identifier, the control bits, and the data content into one of the transmit buffers. Finally, the buffer is flagged as ready for transmission by clearing the associated TXE flag.

The MSCAN then schedules the message for transmission and signals the successful transmission of the buffer by setting the associated TXE flag. A transmit interrupt (see [Section 11.5.7.2, “Transmit Interrupt”](#)) is generated<sup>1</sup> when TXEx is set and can be used to drive the application software to re-load the buffer.

If more than one buffer is scheduled for transmission when the CAN bus becomes available for arbitration, the MSCAN uses the local priority setting of the three buffers to determine the prioritization. For this purpose, every transmit buffer has an 8-bit local priority field (PRIO). The application software programs this field when the message is set up. The local priority reflects the priority of this particular message relative to the set of messages being transmitted from this node. The lowest binary value of the PRIO field is defined to be the highest priority. The internal scheduling process takes place when the MSCAN arbitrates for the CAN bus. This is also the case after the occurrence of a transmission error.

When a high priority message is scheduled by the application software, it may become necessary to abort a lower priority message in one of the three transmit buffers. Because messages that are already in transmission cannot be aborted, the user must request the abort by setting the corresponding abort request bit (ABTRQ) (see [Section 11.3.9, “MSCAN Transmitter Message Abort Request Register \(CANTARQ\)”](#)). The MSCAN then grants the request, if possible, by:

1. Setting the corresponding abort acknowledge flag (ABTAK) in the CANTAAK register.
2. Setting the associated TXE flag to release the buffer.
3. Generating a transmit interrupt. The transmit interrupt handler software can determine from the setting of the ABTAK flag whether the message was aborted (ABTAK = 1) or sent (ABTAK = 0).

### 11.5.2.3 Receive Structures

The received messages are stored in a five stage input FIFO. The five message buffers are alternately mapped into a single memory area (see [Figure 11-37](#)). The background receive buffer (RxBG) is exclusively associated with the MSCAN, but the foreground receive buffer (RxFG) is addressable by the CPU (see [Figure 11-37](#)). This scheme simplifies the handler software because only one address area is applicable for the receive process.

All receive buffers have a size of 15 bytes to store the CAN control bits, the identifier (standard or extended), the data contents, and a time stamp, if enabled (see [Section 11.4, “Programmer’s Model of Message Storage”](#)).

The receiver full flag (RXF) (see [Section 11.3.5, “MSCAN Receiver Flag Register \(CANRFLG”](#)) signals the status of the foreground receive buffer. When the buffer contains a correctly received message with a matching identifier, this flag is set.

On reception, each message is checked to see whether it passes the filter (see [Section 11.5.2.4, “Identifier Acceptance Filter”](#)) and simultaneously is written into the active RxBG. After successful reception of a valid message, the MSCAN shifts the content of RxBG into the receiver FIFO<sup>2</sup>, sets the RXF flag, and generates a receive interrupt (see [Section 11.5.7.3, “Receive Interrupt”](#)) to the CPU<sup>3</sup>. The user’s receive handler must read the received message from the RxFG and then reset the RXF flag to acknowledge the interrupt and to release the foreground buffer. A new message, which can follow immediately after the IFS

1. The transmit interrupt occurs only if not masked. A polling scheme can be applied on TXEx also.  
 2. Only if the RXF flag is not set.  
 3. The receive interrupt occurs only if not masked. A polling scheme can be applied on RXF also.

field of the CAN frame, is received into the next available RxBG. If the MSCAN receives an invalid message in its RxBG (wrong identifier, transmission errors, etc.) the actual contents of the buffer is over-written by the next message. The buffer is then not shifted into the FIFO.

When the MSCAN module is transmitting, the MSCAN receives its own transmitted messages into the background receive buffer, RxBG, but does not shift it into the receiver FIFO, generate a receive interrupt, or acknowledge its own messages on the CAN bus. The exception to this rule is in loopback mode (see [Section 11.3.2, “MSCAN Control Register 1 \(CANCTL1\)”](#)) where the MSCAN treats its own messages exactly like all other incoming messages. The MSCAN receives its own transmitted messages in the event that it loses arbitration. If arbitration is lost, the MSCAN must be prepared to become a receiver.

An overrun condition occurs when all receive message buffers in the FIFO are filled with correctly received messages with accepted identifiers and another message is correctly received from the CAN bus with an accepted identifier. The latter message is discarded. An error interrupt with overrun indication is generated if enabled (see [Section 11.5.7.5, “Error Interrupt”](#)). The MSCAN remains able to transmit messages while the receiver FIFO is full, but all incoming messages are discarded. As soon as a receive buffer in the FIFO is available again, new valid messages are accepted.

#### 11.5.2.4 Identifier Acceptance Filter

The MSCAN identifier acceptance registers (see [Section 11.3.12, “MSCAN Identifier Acceptance Control Register \(CANIDAC\)”](#)) define the acceptable patterns of the standard or extended identifier (ID[10:0] or ID[28:0]). Any of these bits can be marked don't care in the MSCAN identifier mask registers (see [Section 11.3.17, “MSCAN Identifier Mask Registers \(CANIDMR0–CANIDMR7\)”](#)).

A filter hit is indicated to the application software by a set receive buffer full flag (RXF = 1) and three bits in the CANIDAC register (see [Section 11.3.12, “MSCAN Identifier Acceptance Control Register \(CANIDAC\)”](#)). These identifier hit flags (IDHIT[2:0]) clearly identify the filter section that caused the acceptance. They simplify the application software's task to identify the cause of the receiver interrupt. If more than one hit occurs (two or more filters match), the lower hit has priority.

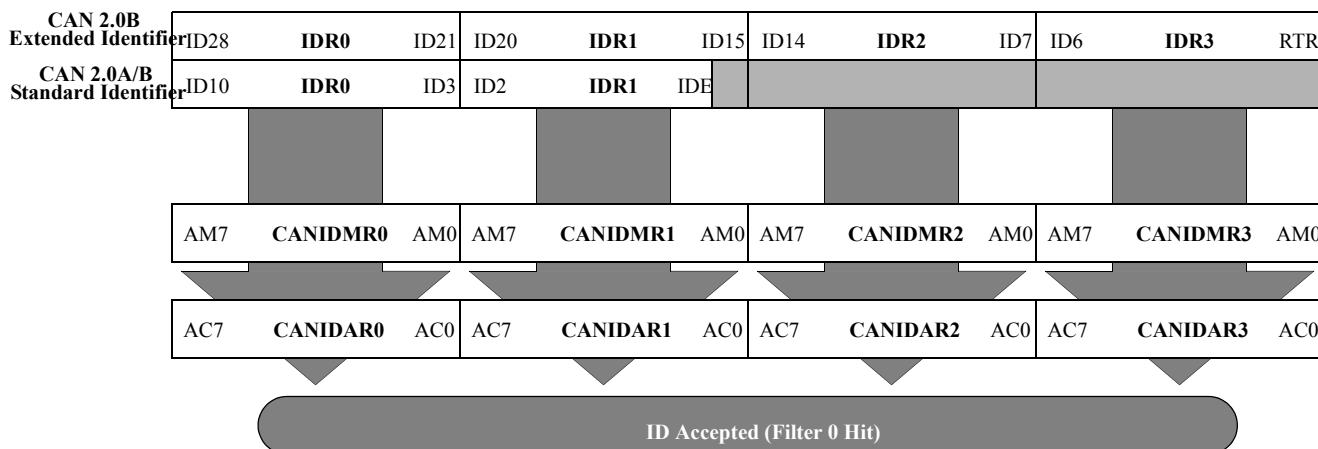
A flexible programmable generic identifier acceptance filter has been introduced to reduce the CPU interrupt loading. The filter is programmable to operate in four different modes (see Bosch CAN 2.0A/B protocol specification):

- Two identifier acceptance filters, each to be applied to:
  - The full 29 bits of the extended identifier and to the following bits of the CAN 2.0B frame:
    - Remote transmission request (RTR)
    - Identifier extension (IDE)
    - Substitute remote request (SRR)
  - The 11 bits of the standard identifier plus the RTR and IDE bits of the CAN 2.0A/B messages<sup>1</sup>. This mode implements two filters for a full length CAN 2.0B compliant extended identifier. [Figure 11-38](#) shows how the first 32-bit filter bank (CANIDAR0–CANIDAR3, CANIDMR0–CANIDMR3) produces a filter 0 hit. Similarly, the second filter bank (CANIDAR4–CANIDAR7, CANIDMR4–CANIDMR7) produces a filter 1 hit.

<sup>1</sup>Although this mode can be used for standard identifiers, it is recommended to use the four or eight identifier acceptance filters for standard identifiers

- Four identifier acceptance filters, each to be applied to :
  - a) the 14 most significant bits of the extended identifier plus the SRR and IDE bits of CAN 2.0B messages or
  - b) the 11 bits of the standard identifier, the RTR and IDE bits of CAN 2.0A/B messages.

[Figure 11-39](#) shows how the first 32-bit filter bank (CANIDAR0–CANIDA3, CANIDMR0–3CANIDMR) produces filter 0 and 1 hits. Similarly, the second filter bank (CANIDAR4–CANIDAR7, CANIDMR4–CANIDMR7) produces filter 2 and 3 hits.
- Eight identifier acceptance filters, each to be applied to the first 8 bits of the identifier. This mode implements eight independent filters for the first 8 bits of a CAN 2.0A/B compliant standard identifier or a CAN 2.0B compliant extended identifier. [Figure 11-40](#) shows how the first 32-bit filter bank (CANIDAR0–CANIDAR3, CANIDMR0–CANIDMR3) produces filter 0 to 3 hits. Similarly, the second filter bank (CANIDAR4–CANIDAR7, CANIDMR4–CANIDMR7) produces filter 4 to 7 hits.
- Closed filter. No CAN message is copied into the foreground buffer RxFG, and the RXF flag is never set.



**Figure 11-38. 32-bit Maskable Identifier Acceptance Filter**

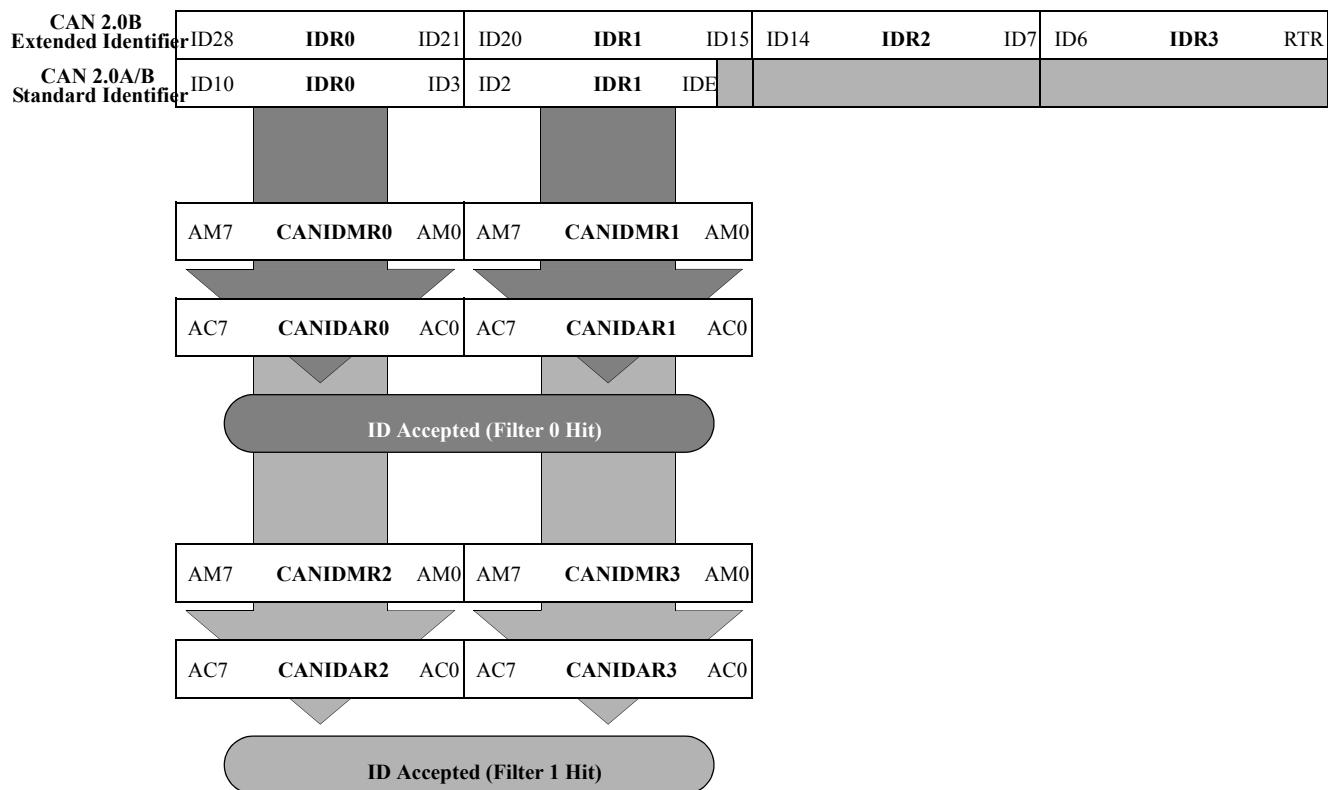
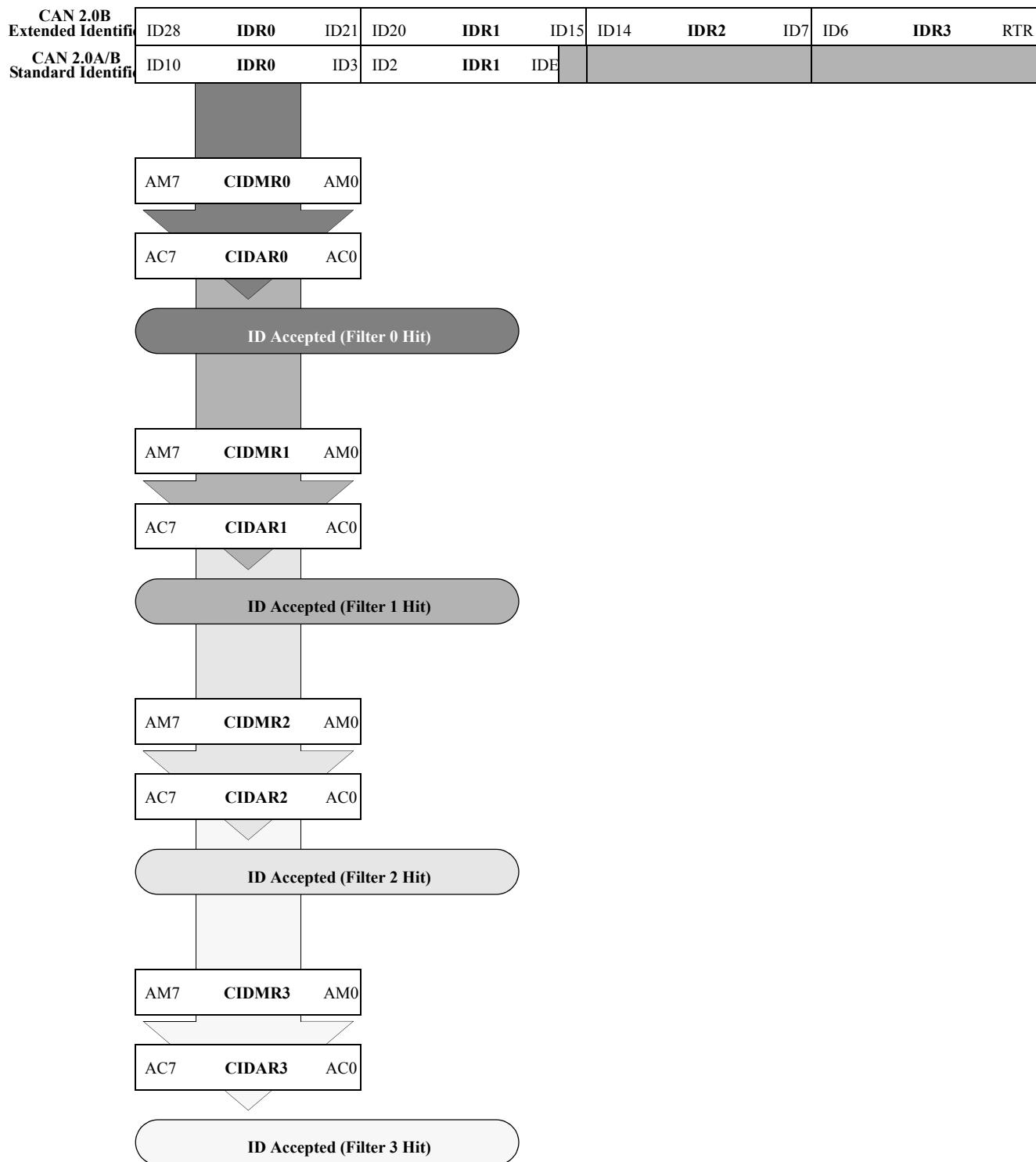


Figure 11-39. 16-bit Maskable Identifier Acceptance Filters

**Figure 11-40. 8-bit Maskable Identifier Acceptance Filters**

MSCAN filter uses three sets of registers to provide the filter configuration. Firstly, the CANIDAC register determines the configuration of the banks into filter sizes and number of filters. Secondly, registers CANIDMR0/1/2/3 determine those bits on which the filter operates by placing a 0 at the appropriate bit

position in the filter register. Finally, registers CANIDAR0/1/2/3 determine the value of those bits determined by CANIDMR0/1/2/3.

For instance in the case of the filter value of:

0001x1001x0

The CANIDMR0/1/2/3 register would be configured as:

00001000010

and so all message identifier bits except bit 1 and bit 6 would be compared against the CANIDAR0/1/2/3 registers. These would be configured as:

00010100100

In this case bits 1 and 6 are set to 0, but it is equally valid to set them to 1 because they are ignored.

### 11.5.2.5 Identifier Acceptance Filters example

As described above, filters work by comparisons to individual bits in the CAN message identifier field. The filter checks each one of the eleven bits of a standard CAN message identifier. Suppose a filter value of 0001x1001x0. In this simple example, there are only three possible CAN messages.

Filter value: 0001x1001x0

Message 1: 00011100110

Message 2: 00110100110

Message 3: 00010100100

Message 2 is rejected because its third most significant bit is not 0 – 001. The filter is simply a convenient way of defining the set of messages that the CPU must receive. For full 29-bits of an extended CAN message identifier, the filter identifies two sets of messages: one set that it receives and one set that it rejects. Alternatively, the filter may be split into two. This allows the MSCAN to examine only the first 16 bits of a message identifier, but allows two separate filters to perform the checking. See the example below:

Filter value A: 0001x1001x0

Filter value B: 00x101x01x0

Message 1: 00011100110

Message 2: 00110100110

Message 3: 00010100100

MSCAN accepts all three messages. Filter A accepts messages 1 and 3 as before and filter B accepts message 2. In practice, it is unimportant which filter accepts the message. Messages accepted by either are placed in the input buffer. A message may be accepted by more than one filter.

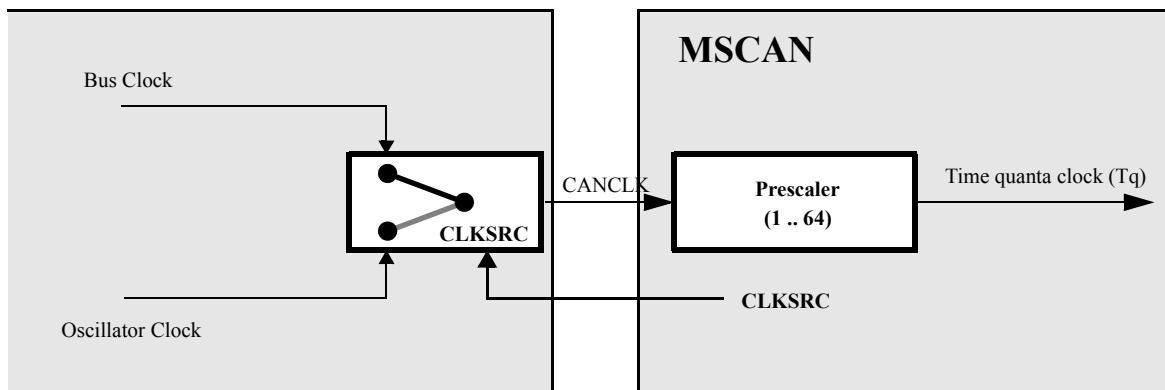
### 11.5.2.6 Protocol Violation Protection

The MSCAN protects the user from accidentally violating the CAN protocol through programming errors. The protection logic implements the following features:

- The receive and transmit error counters cannot be written or otherwise manipulated.
- All registers that control the configuration of the MSCAN cannot be modified while the MSCAN is on-line. The MSCAN has to be in Initialization Mode. The corresponding INITRQ/INITAK handshake bits in the CANCTL0/CANCTL1 registers (see [Section 11.3.1, “MSCAN Control Register 0 \(CANCTL0\)”](#)) serve as a lock to protect the following registers:
  - MSCAN control 1 register (CANCTL1)
  - MSCAN bus timing registers 0 and 1 (CANBTR0, CANBTR1)
  - MSCAN identifier acceptance control register (CANIDAC)
  - MSCAN identifier acceptance registers (CANIDAR0–CANIDAR7)
  - MSCAN identifier mask registers (CANIDMR0–CANIDMR7)
- The TXCAN pin is immediately forced to a recessive state when the MSCAN goes into the power down mode or initialization mode (see [Section 11.5.5.6, “MSCAN Power Down Mode,”](#) and [Section 11.5.5.5, “MSCAN Initialization Mode”](#)).
- The MSCAN enable bit (CANE) is writable only once in normal system operation modes, which provides further protection against inadvertently disabling the MSCAN.

### 11.5.2.7 Clock System

[Figure 11-41](#) shows the structure of the MSCAN clock generation circuitry.



**Figure 11-41. MSCAN Clocking Scheme**

The clock source bit (CLKSRC) in the CANCTL1 register ([11.3.2/11-6](#)) defines whether the internal CANCLK is connected to the output of a crystal oscillator (oscillator clock) or to the bus clock.

The clock source has to be chosen such that the tight oscillator tolerance requirements (up to 0.4%) of the CAN protocol are met. Additionally, for high CAN bus rates (1 Mbps), a 45% to 55% duty cycle of the clock is required.

If the bus clock is generated from a PLL, it is recommended to select the oscillator clock rather than the bus clock due to jitter considerations, especially at the faster CAN bus rates. PLL lock may also be too wide to ensure adequate clock tolerance.

For microcontrollers without a clock and reset generator (CRG), CANCLK is driven from the crystal oscillator (oscillator clock).

A programmable prescaler generates the time quanta ( $T_q$ ) clock from CANCLK. A time quantum is the atomic unit of time handled by the MSCAN.

Eqn. 11-2

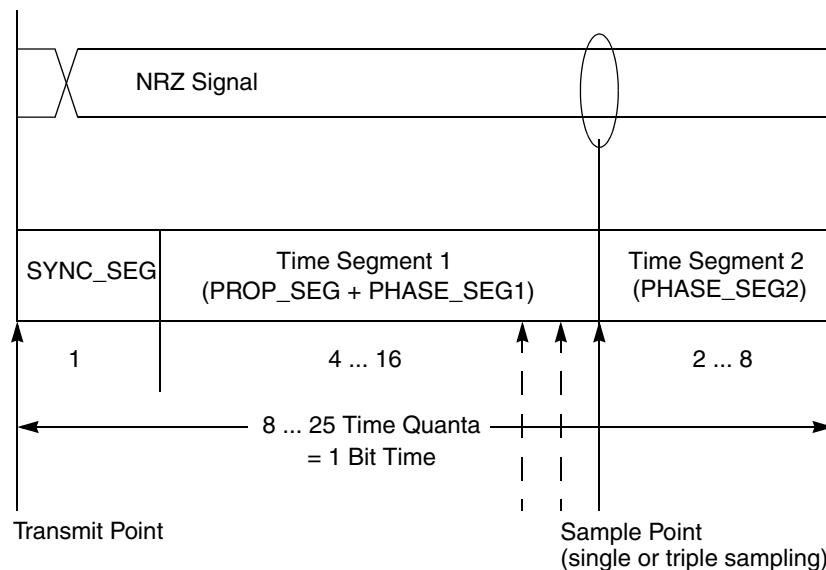
$$T_q = \frac{f_{CANCLK}}{(\text{Prescaler} \cdot \text{value})}$$

A bit time is subdivided into three segments as described in the Bosch CAN specification. (see Figure 11-42):

- SYNC\_SEG: This segment has a fixed length of one time quantum. Signal edges are expected to happen within this section.
  - Time Segment 1: This segment includes the PROP\_SEG and the PHASE\_SEG1 of the CAN standard. It can be programmed by setting the parameter TSEG1 to consist of 4 to 16 time quanta.
  - Time Segment 2: This segment represents the PHASE\_SEG2 of the CAN standard. It can be programmed by setting the TSEG2 parameter to be 2 to 8 time quanta long.

Eqn. 11-3

$$\text{Bit Rate} = \frac{f_{Tq}}{(\text{number} \cdot \text{of} \cdot \text{Time} \cdot \text{Quanta})}$$



**Table 11-34. Time Segment Syntax**

Syntax	Description
SYNC_SEG	System expects transitions to occur on the CAN bus during this period.
Transmit Point	A node in transmit mode transfers a new value to the CAN bus at this point.
Sample Point	A node in receive mode samples the CAN bus at this point. If the three samples per bit option is selected, then this point marks the position of the third sample.

The synchronization jump width (see the Bosch CAN specification for details) can be programmed in a range of 1 to 4 time quanta by setting the SJW parameter.

The SYNC\_SEG, TSEG1, TSEG2, and SJW parameters are set by programming the MSCAN bus timing registers (CANBTR0, CANBTR1) (see [Section 11.3.3, “MSCAN Bus Timing Register 0 \(CANBTR0\)”](#) and [Section 11.3.4, “MSCAN Bus Timing Register 1 \(CANBTR1\)”](#)).

[Table 11-35](#) gives an overview of the CAN compliant segment settings and the related parameter values.

#### NOTE

It is the user’s responsibility to ensure the bit time settings are in compliance with the CAN standard.

**Table 11-35. CAN Standard Compliant Bit Time Segment Settings**

Time Segment 1	TSEG1	Time Segment 2	TSEG2	Synchronization Jump Width	SJW
5 .. 10	4 .. 9	2	1	1 .. 2	0 .. 1
4 .. 11	3 .. 10	3	2	1 .. 3	0 .. 2
5 .. 12	4 .. 11	4	3	1 .. 4	0 .. 3
6 .. 13	5 .. 12	5	4	1 .. 4	0 .. 3
7 .. 14	6 .. 13	6	5	1 .. 4	0 .. 3
8 .. 15	7 .. 14	7	6	1 .. 4	0 .. 3
9 .. 16	8 .. 15	8	7	1 .. 4	0 .. 3

### 11.5.3 Timer Link

The MSCAN generates an internal time stamp when a valid frame is received or transmitted and the TIME bit is enabled. Because the CAN specification defines a frame to be valid if no errors occur before the end of frame (EOF) field is transmitted successfully, the actual value of an internal timer is written at EOF to the appropriate time stamp position within the transmit buffer. For receive frames, the time stamp is written to the receive buffer.

## 11.5.4 Modes of Operation

### 11.5.4.1 Normal Modes

The MSCAN module behaves as described within this specification in all normal system operation modes.

### 11.5.4.2 Special Test Modes

The MSCAN module behaves as described within this specification in all special system test modes (Analog/Functional test modes, for example).

### 11.5.4.3 Emulation Modes

In all emulation modes, the MSCAN module behaves like normal system operation modes as described within this specification.

### 11.5.4.4 Listen-Only Mode

In an optional CAN bus monitoring mode (listen-only), the CAN node is able to receive valid data frames and valid remote frames, but it sends only recessive bits on the CAN bus. In addition, it cannot start a transmission. If the MAC sub-layer is required to send a dominant bit (ACK bit, overload flag, or active error flag), the bit is rerouted internally so that the MAC sub-layer monitors this dominant bit, although the CAN bus may remain in recessive state externally.

### 11.5.4.5 Security Modes

The MSCAN module has no security features.

### 11.5.4.6 Loopback Self Test Mode

Loopback self test mode is sometimes used to check software, independent of connections in the external system, to help isolate system problems. The RXCAN input pin is ignored and the TXCAN output goes to the recessive state (logic 1). The MSCAN behaves as it does normally when transmitting and treats its own transmitted message as a message received from a remote node. In this state, the MSCAN ignores the bit sent during the ACK slot in the CAN frame acknowledge field to ensure proper reception of its own message. Transmit and receive interrupts are generated.

## 11.5.5 Low-Power Options

If the MSCAN is disabled (CANE = 0), the MSCAN clocks are stopped for power saving.

If the MSCAN is enabled (CANE = 1), the MSCAN has two additional modes with reduced power consumption: sleep and power down mode. In sleep mode, power consumption is reduced by stopping all clocks except those to access the registers from the CPU side. In power down mode, all clocks are stopped and no power is consumed.

**Table 11-36** summarizes the combinations of MSCAN and CPU modes. A particular combination of modes is entered by the given settings on the CSWAI and SLPRQ/SLPAK bits.

For all modes, an MSCAN wake-up interrupt can occur only if the MSCAN is in sleep mode ( $\text{SLPRQ} = 1$  and  $\text{SLPAK} = 1$ ), wake-up functionality is enabled ( $\text{WUPE} = 1$ ), and the wake-up interrupt is enabled ( $\text{WUPIE} = 1$ ).

**Table 11-36. CPU vs. MSCAN Operating Modes**

CPU Mode	MSCAN Mode			
	Normal	Reduced Power Consumption		
		Sleep	Power Down	Disabled (CANE=0)
Run	$\text{CSWAI} = X^1$ $\text{SLPRQ} = 0$ $\text{SLPAK} = 0$	$\text{CSWAI} = X$ $\text{SLPRQ} = 1$ $\text{SLPAK} = 1$	—	$\text{CSWAI} = X$ $\text{SLPRQ} = X$ $\text{SLPAK} = X$
Wait	$\text{CSWAI} = 0$ $\text{SLPRQ} = 0$ $\text{SLPAK} = 0$	$\text{CSWAI} = 0$ $\text{SLPRQ} = 1$ $\text{SLPAK} = 1$	$\text{CSWAI} = 1$ $\text{SLPRQ} = X$ $\text{SLPAK} = X$	$\text{CSWAI} = X$ $\text{SLPRQ} = X$ $\text{SLPAK} = X$
Stop3	—	$\text{CSWAI} = X^2$ $\text{SLPRQ} = 1$ $\text{SLPAK} = 1$	$\text{CSWAI} = X$ $\text{SLPRQ} = 0$ $\text{SLPAK} = 0$	$\text{CSWAI} = X$ $\text{SLPRQ} = X$ $\text{SLPAK} = X$
Stop2	—	—	$\text{CSWAI} = X$ $\text{SLPRQ} = X$ $\text{SLPAK} = X$	$\text{CSWAI} = X$ $\text{SLPRQ} = X$ $\text{SLPAK} = X$

<sup>1</sup> X means don't care.

<sup>2</sup> For a safe wake up from Sleep mode, SLPRQ and SLPAK must be set to 1 before going into Stop3 mode.

### 11.5.5.1 Operation in Run Mode

As shown in [Table 11-36](#), only MSCAN sleep mode is available as low power option when the CPU is in run mode.

### 11.5.5.2 Operation in Wait Mode

The WAIT instruction puts the MCU in a low power consumption stand-by mode. If the CSWAI bit is set, additional power can be saved in power down mode because the CPU clocks are stopped. After leaving this power down mode, the MSCAN restarts its internal controllers and enters normal mode again.

While the CPU is in wait mode, the MSCAN can be operated in normal mode and generate interrupts (registers can be accessed via background debug mode). The MSCAN can also operate in any of the low-power modes depending on the values of the SLPRQ/SLPAK and CSWAI bits as seen in [Table 11-36](#).

### 11.5.5.3 Operation in Stop Mode

The STOP instruction puts the MCU in a low power consumption stand-by mode. In stop2 mode, the MSCAN is set in power down mode regardless of the value of the SLPRQ/SLPAK. In stop3 mode, power down or sleep modes are determined by the SLPRQ/SLPAK values set prior to entering stop3. CSWAI bits has no function in any of the stop modes. See [Table 11-36](#).

#### 11.5.5.4 MSCAN Sleep Mode

The CPU can request the MSCAN to enter this low power mode by asserting the SLPRQ bit in the CANCTL0 register. The time when the MSCAN enters sleep mode depends on a fixed synchronization delay and its current activity:

- If there are one or more message buffers scheduled for transmission (TXEx = 0), the MSCAN continues to transmit until all transmit message buffers are empty (TXEx = 1, transmitted successfully or aborted) and then goes into sleep mode.
- If the MSCAN is receiving, it continues to receive and goes into sleep mode as soon as the CAN bus next becomes idle.
- If the MSCAN is neither transmitting nor receiving, it immediately goes into sleep mode.

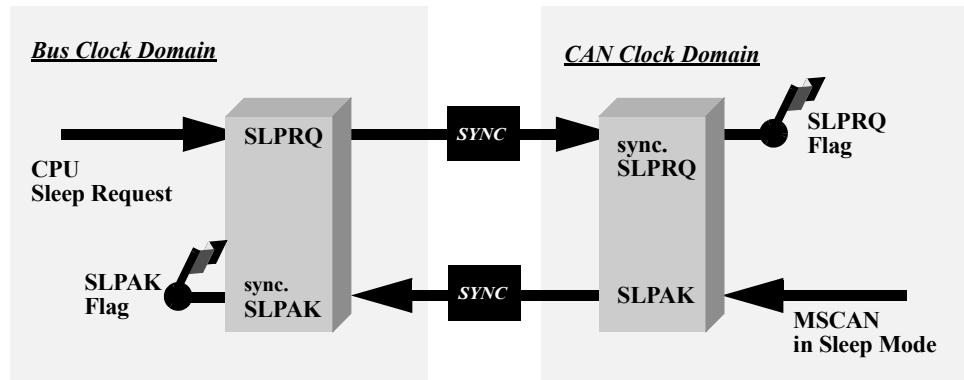


Figure 11-43. Sleep Request / Acknowledge Cycle

#### NOTE

The application software must avoid setting up a transmission (by clearing one or more TXEx flag(s)) and immediately request sleep mode (by setting SLPRQ). Whether the MSCAN starts transmitting or goes into sleep mode directly depends on the exact sequence of operations.

If sleep mode is active, the SLPRQ and SLPAK bits are set (Figure 11-43). The application software must use SLPAK as a handshake indication for the request (SLPRQ) to go into sleep mode.

When in sleep mode (SLPRQ = 1 and SLPAK = 1), the MSCAN stops its internal clocks. However, clocks that allow register accesses from the CPU side continue to run.

If the MSCAN is in bus-off state, it stops counting the 128 occurrences of 11 consecutive recessive bits due to the stopped clocks. The TXCAN pin remains in a recessive state. If RXF = 1, the message can be read and RXF can be cleared. Shifting a new message into the foreground buffer of the receiver FIFO (RxFG) does not take place while in sleep mode.

It is possible to access the transmit buffers and to clear the associated TXE flags. No message abort takes place while in sleep mode.

If the WUPE bit in CANCTL0 is not asserted, the MSCAN masks any activity it detects on CAN. The RXCAN pin is therefore held internally in a recessive state. This locks the MSCAN in sleep mode (Figure 11-44).

The MSCAN is able to leave sleep mode (wake up) only when:

- CAN bus activity occurs and WUPE = 1

or

- the CPU clears the SLPRQ bit

#### NOTE

The CPU cannot clear the SLPRQ bit before sleep mode (SLPRQ = 1 and SLPAK = 1) is active.

After wake-up, the MSCAN waits for 11 consecutive recessive bits to synchronize to the CAN bus. As a consequence, if the MSCAN is woken-up by a CAN frame, this frame is not received.

The receive message buffers (RxFG and RxBG) contain messages if they were received before sleep mode was entered. All pending actions are executed upon wake-up; copying of RxBG into RxFG, message aborts and message transmissions. If the MSCAN remains in bus-off state after sleep mode was exited, it continues counting the 128 occurrences of 11 consecutive recessive bits.

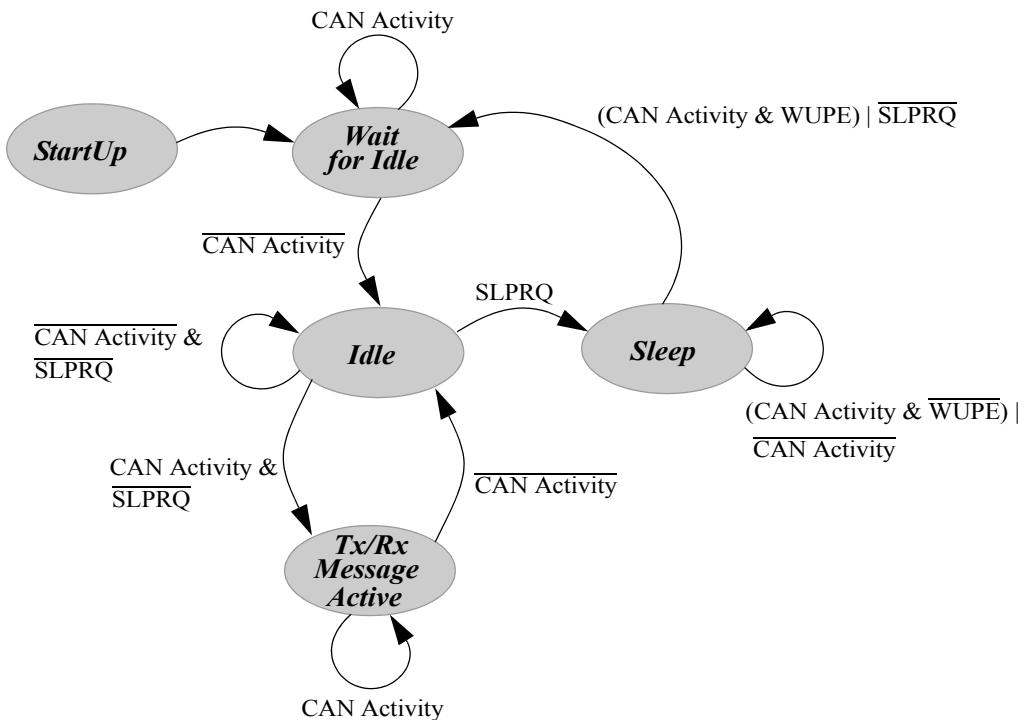


Figure 11-44. Simplified State Transitions for Entering/Leaving Sleep Mode

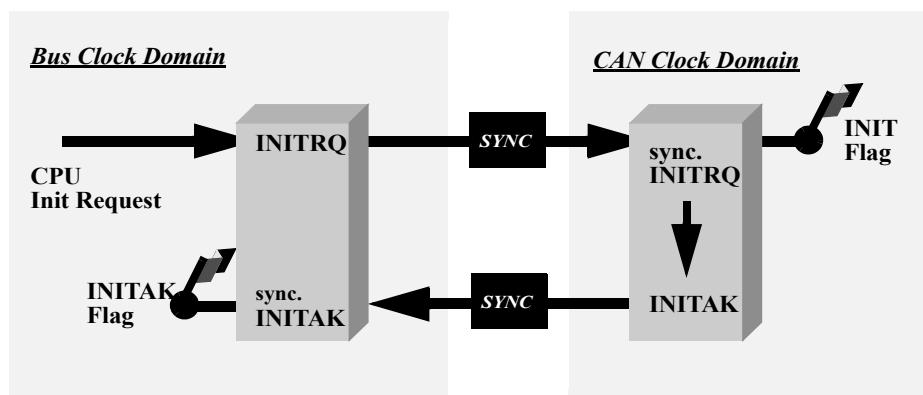
#### 11.5.5.5 MSCAN Initialization Mode

In initialization mode, any on-going transmission or reception is immediately aborted and synchronization to the CAN bus is lost, potentially causing CAN protocol violations. To protect the CAN bus system from fatal consequences of violations, the MSCAN immediately drives the TXCAN pin into a recessive state.

**NOTE**

The user is responsible for ensuring that the MSCAN is not active when initialization mode is entered. The recommended procedure is to bring the MSCAN into sleep mode ( $\text{SLPRQ} = 1$  and  $\text{SLPAK} = 1$ ) before setting the  $\text{INITRQ}$  bit in the  $\text{CANCTL0}$  register. Otherwise, the abort of an on-going message can cause an error condition and can impact other CAN bus devices.

In initialization mode, the MSCAN is stopped. However, interface registers remain accessible. This mode is used to reset the  $\text{CANCTL0}$ ,  $\text{CANRFLG}$ ,  $\text{CANRIER}$ ,  $\text{CANTFLG}$ ,  $\text{CANTIER}$ ,  $\text{CANTARQ}$ ,  $\text{CANTAAK}$ , and  $\text{CANTBSEL}$  registers to their default values. In addition, the MSCAN enables the configuration of the  $\text{CANBTR0}$ ,  $\text{CANBTR1}$  bit timing registers;  $\text{CANIDAC}$ ; and the  $\text{CANIDAR}$ ,  $\text{CANIDMR}$  message filters. See [Section 11.3.1, “MSCAN Control Register 0 \(CANCTL0\),”](#) for a detailed description of the initialization mode.



**Figure 11-45. Initialization Request/Acknowledge Cycle**

Due to independent clock domains within the MSCAN,  $\text{INITRQ}$  must be synchronized to all domains by using a special handshake mechanism. This handshake causes additional synchronization delay (see [Section Figure 11-45., “Initialization Request/Acknowledge Cycle”](#)).

If there is no message transfer ongoing on the CAN bus, the minimum delay is two additional bus clocks and three additional CAN clocks. When all parts of the MSCAN are in initialization mode, the  $\text{INITAK}$  flag is set. The application software must use  $\text{INITAK}$  as a handshake indication for the request ( $\text{INITRQ}$ ) to go into initialization mode.

**NOTE**

The CPU cannot clear  $\text{INITRQ}$  before initialization mode ( $\text{INITRQ} = 1$  and  $\text{INITAK} = 1$ ) is active.

### 11.5.5.6 MSCAN Power Down Mode

The MSCAN is in power down mode ([Table 11-36](#)) when

- CPU is in stop mode
- or
- CPU is in wait mode and the  $\text{CSWAI}$  bit is set

When entering the power down mode, the MSCAN immediately stops all ongoing transmissions and receptions, potentially causing CAN protocol violations. To protect the CAN bus system from fatal consequences of violations to the above rule, the MSCAN immediately drives the TXCAN pin into a recessive state.

#### **NOTE**

The user is responsible for ensuring that the MSCAN is not active when power down mode is entered. The recommended procedure is to bring the MSCAN into Sleep mode before the STOP or WAIT instruction (if CSWAI is set) is executed. Otherwise, the abort of an ongoing message can cause an error condition and impact other CAN bus devices.

In power down mode, all clocks are stopped and no registers can be accessed. If the MSCAN was not in sleep mode before power down mode became active, the module performs an internal recovery cycle after powering up. This causes some fixed delay before the module enters normal mode again.

#### **11.5.5.7 Programmable Wake-Up Function**

The MSCAN can be programmed to wake up the MSCAN as soon as CAN bus activity is detected (see control bit WUPE in [Section 11.3.1, “MSCAN Control Register 0 \(CANCTL0\)”](#)). The sensitivity to existing CAN bus action can be modified by applying a low-pass filter function to the RXCAN input line while in sleep mode (see control bit WUPM in [Section 11.3.2, “MSCAN Control Register 1 \(CANCTL1\)”](#)).

This feature can be used to protect the MSCAN from wake-up due to short glitches on the CAN bus lines. Such glitches can result from—for example—electromagnetic interference within noisy environments.

#### **11.5.6 Reset Initialization**

The reset state of each individual bit is listed in [Section 11.3, “Register Definition,”](#) which details all the registers and their bit-fields.

#### **11.5.7 Interrupts**

This section describes all interrupts originated by the MSCAN. It documents the enable bits and generated flags. Each interrupt is listed and described separately.

##### **11.5.7.1 Description of Interrupt Operation**

The MSCAN supports four interrupt vectors (see [Table 11-37](#)), any of which can be individually masked (for details see sections from [Section 11.3.6, “MSCAN Receiver Interrupt Enable Register \(CANRIER\),”](#) to [Section 11.3.8, “MSCAN Transmitter Interrupt Enable Register \(CANTIER\)”](#)).

#### **NOTE**

The dedicated interrupt vector addresses are defined in the [Resets and Interrupts chapter](#).

**Table 11-37. Interrupt Vectors**

<b>Interrupt Source</b>	<b>CCR Mask</b>	<b>Local Enable</b>
Wake-Up Interrupt (WUPIF)	1 bit	CANRIER (WUPIE)
Error Interrupts Interrupt (CSCIF, OVRIF)	1 bit	CANRIER (CSCIE, OVRIE)
Receive Interrupt (RXF)	1 bit	CANRIER (RXFIE)
Transmit Interrupts (TXE[2:0])	1 bit	CANTIER (TXEIE[2:0])

### 11.5.7.2 Transmit Interrupt

At least one of the three transmit buffers is empty (not scheduled) and can be loaded to schedule a message for transmission. The TXEx flag of the empty message buffer is set.

### 11.5.7.3 Receive Interrupt

A message is successfully received and shifted into the foreground buffer (RxFG) of the receiver FIFO. This interrupt is generated immediately after receiving the EOF symbol. The RXF flag is set. If there are multiple messages in the receiver FIFO, the RXF flag is set as soon as the next message is shifted to the foreground buffer.

### 11.5.7.4 Wake-Up Interrupt

A wake-up interrupt is generated if activity on the CAN bus occurs during MSCAN internal sleep mode. WUPE (see [Section 11.3.1, “MSCAN Control Register 0 \(CANCTL0\)”](#)) must be enabled.

### 11.5.7.5 Error Interrupt

An error interrupt is generated if an overrun of the receiver FIFO, error, warning, or bus-off condition occurs. [Section 11.3.5, “MSCAN Receiver Flag Register \(CANRFLG\)”](#) indicates one of the following conditions:

- Overrun — An overrun condition of the receiver FIFO as described in [Section 11.5.2.3, “Receive Structures,”](#) occurred.
- CAN Status Change — The actual value of the transmit and receive error counters control the CAN bus state of the MSCAN. As soon as the error counters skip into a critical range (Tx/Rx-warning, Tx/Rx-error, bus-off) the MSCAN flags an error condition. The status change, which caused the error condition, is indicated by the TSTAT and RSTAT flags (see [Section 11.3.5, “MSCAN Receiver Flag Register \(CANRFLG\)”](#) and [Section 11.3.6, “MSCAN Receiver Interrupt Enable Register \(CANRIER\)”](#)).

### 11.5.7.6 Interrupt Acknowledge

Interrupts are directly associated with one or more status flags in the [Section 11.3.5, “MSCAN Receiver Flag Register \(CANRFLG\)”](#) or the [Section 11.3.7, “MSCAN Transmitter Flag Register \(CANTFLG\)”](#). Interrupts are pending as long as one of the corresponding flags is set. The flags in CANRFLG and CANTFLG must be reset within the interrupt handler to handshake the interrupt. The flags are reset by writing a 1 to the corresponding bit position. A flag cannot be cleared if the respective condition prevails.

**NOTE**

It must be guaranteed that the CPU clears only the bit causing the current interrupt. For this reason, bit manipulation instructions (BSET) must not be used to clear interrupt flags. These instructions may cause accidental clearing of interrupt flags set after entering the current interrupt service routine.

### **11.5.7.7 Recovery from Stop or Wait**

The MSCAN can recover from stop or wait via the wake-up interrupt. This interrupt can only occur if the MSCAN was in sleep mode ( $SLPRQ = 1$  and  $SLPAK = 1$ ) before entering power down mode, the wake-up option is enabled ( $WUPE = 1$ ), and the wake-up interrupt is enabled ( $WUPIE = 1$ ).

## **11.6 Initialization/Application Information**

### **11.6.1 MSCAN initialization**

The procedure to initially start up the MSCAN module out of reset is as follows:

1. Assert CANE
2. Write to the configuration registers in initialization mode
3. Clear INITRQ to leave initialization mode and enter normal mode

If the configuration of registers that are writable in initialization mode needs to be changed only when the MSCAN module is in normal mode:

1. Bring the module into sleep mode by setting SLPRQ and awaiting SLPAK to assert after the CAN bus becomes idle.
2. Enter initialization mode: assert INITRQ and await INITAK
3. Write to the configuration registers in initialization mode
4. Clear INITRQ to leave initialization mode and continue in normal mode

### **11.6.2 Bus-Off Recovery**

The bus-off recovery is user configurable. The bus-off state can be exited automatically or on user request.

For reasons of backwards compatibility, the MSCAN defaults to automatic recovery after reset. In this case, the MSCAN becomes error active again after counting 128 occurrences of 11 consecutive recessive bits on the CAN bus (See the Bosch CAN specification for details).

- If the MSCAN is configured for user request (BORM set in [Section 11.3.2, “MSCAN Control Register 1 \(CANCTL1\)”](#)), the recovery from bus-off starts after both independent events have become true: 128 occurrences of 11 consecutive recessive bits on the CAN bus have been monitored
- BOHOLD in [Section 11.3.13, “MSCAN Miscellaneous Register \(CANMISC\)](#) has been cleared by the user.



# Chapter 12

## Serial Communication Interface (S08SCIV4)

### 12.1 Introduction

The MCF51JM128 series include two independent serial communications interface (SCI) modules that are sometimes called universal asynchronous receiver/transmitters (UARTs). Typically, these systems are used to connect to the RS232 serial input/output (I/O) port of a personal computer or workstation, but they can also be used to communicate with other embedded controllers.

A flexible, 13-bit, modulo-based baud-rate generator supports a broad range of standard baud rates beyond 115.2 kbaud. Transmit and receive operations within the same SCI use a common baud rate, and each SCI module has a separate baud-rate generator.

The receiver employs an advanced data sampling technique that ensures reliable communication and noise detection. Hardware parity, receiver wakeup, and double-buffering on transmit and receive are also included.

#### NOTE

Ignore any references to stop1 low-power mode in this chapter, because this device does not support it.

For details on low-power mode operation, refer to [Table 3-2](#) in [Chapter 3, “Modes of Operation”](#).

## 12.1.1 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

## 12.1.2 Modes of Operation

See [Section 12.3, “Functional Description,”](#) for details concerning SCI operation in these modes:

- 8- and 9-bit data modes
- Stop mode operation
- Loop mode
- Single-wire mode

### 12.1.3 Block Diagram

Figure 12-1 shows the transmitter portion of the SCI.

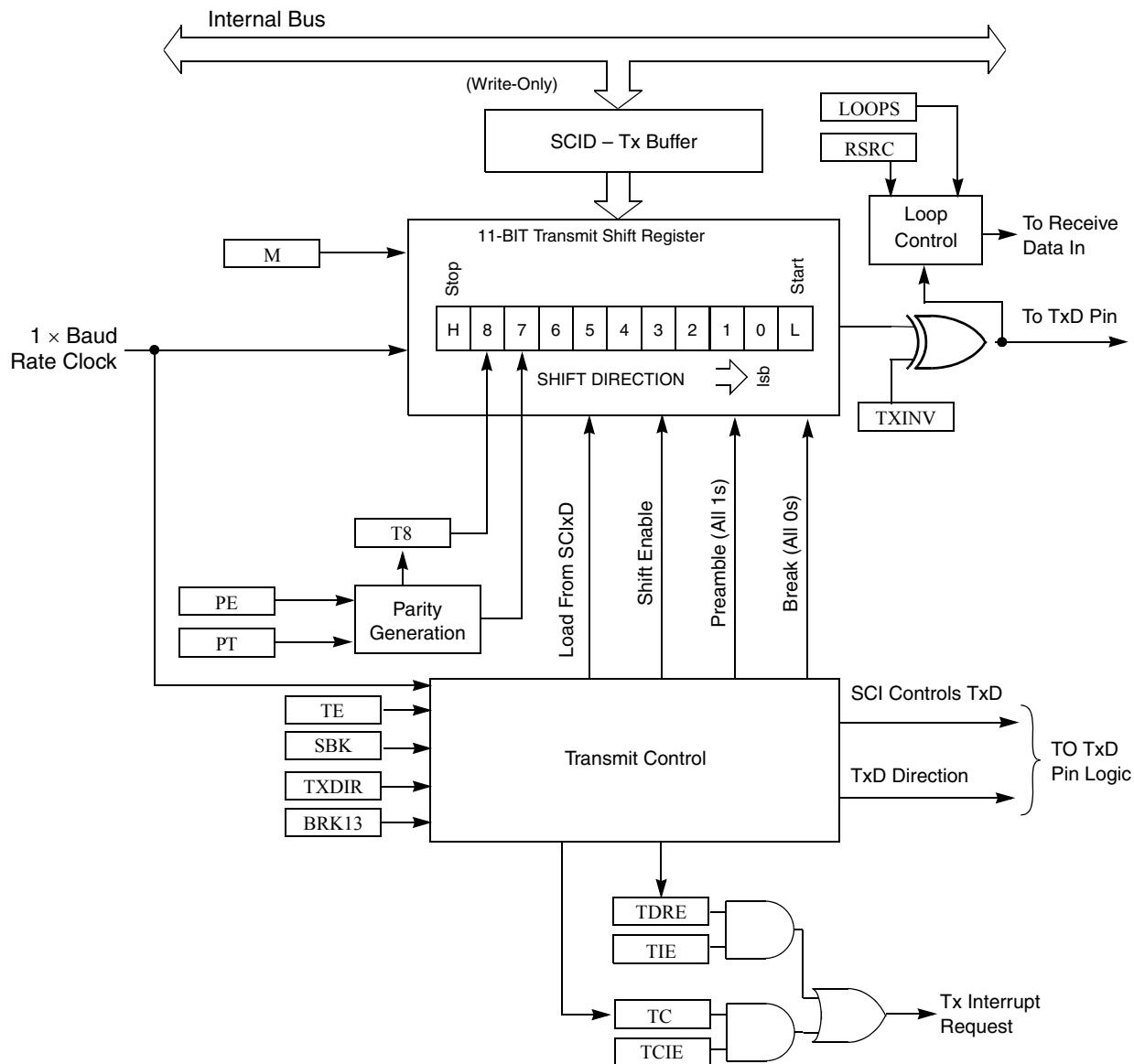


Figure 12-1. SCI Transmitter Block Diagram

Figure 12-2 shows the receiver portion of the SCI.

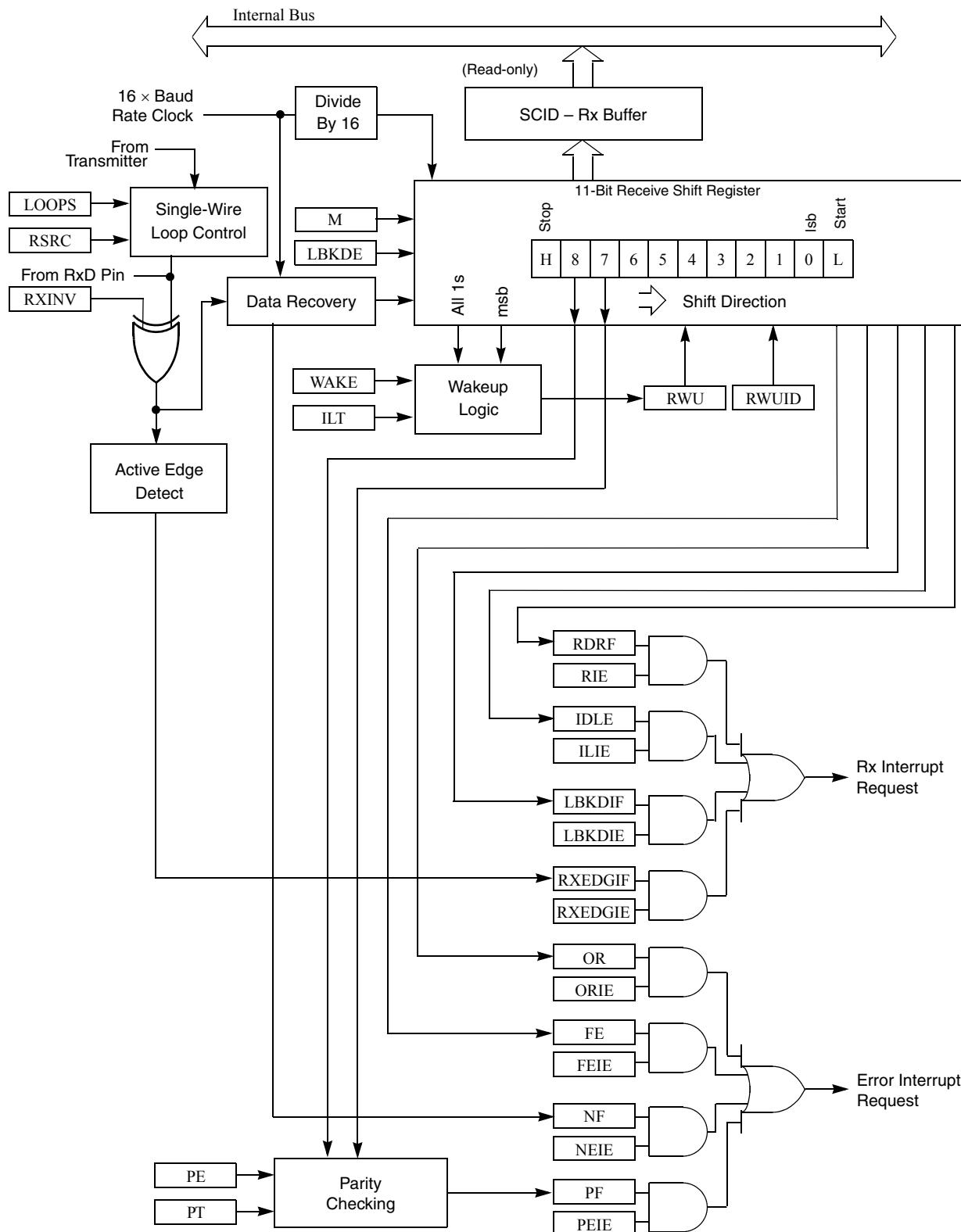


Figure 12-2. SCI Receiver Block Diagram

## 12.2 Register Definition

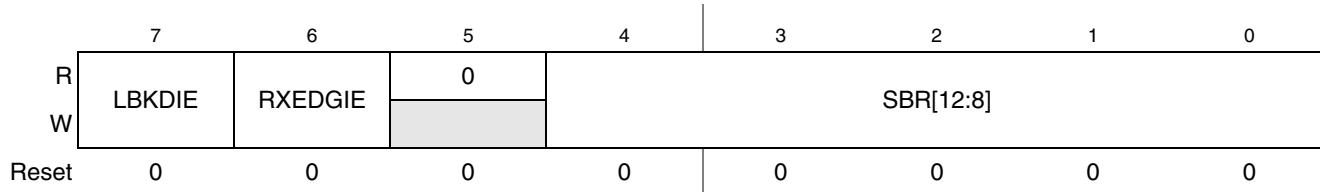
The SCI has eight 8-bit registers to control baud rate, select SCI options, report SCI status, and for transmit/receive data.

Refer to the direct-page register summary in the [memory](#) chapter of this document 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.

### 12.2.1 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.

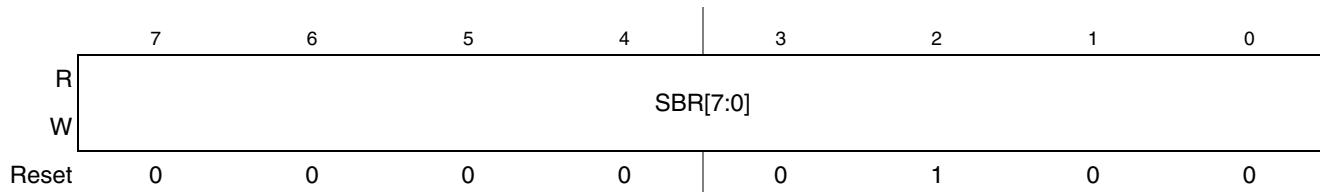
SCIxBDL is reset to a non-zero value, so after reset the baud rate generator remains disabled until the first time the receiver or transmitter is enabled (RE or TE bits in SCIxC2 are written to 1).



**Figure 12-3. SCI Baud Rate Register (SCIxBDH)**

**Table 12-1. 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 BUSCLK/(16×BR). See also BR bits in <a href="#">Table 12-2</a> .



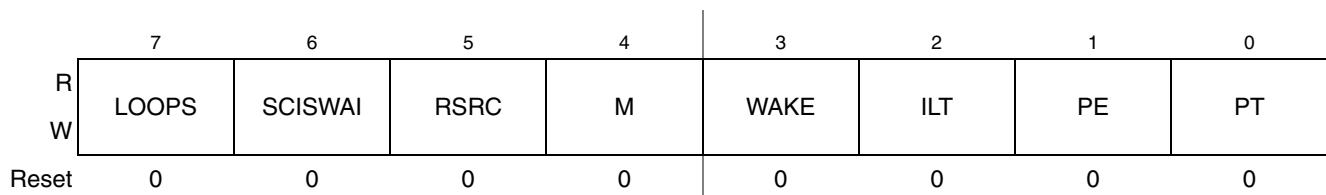
**Figure 12-4. SCI Baud Rate Register (SCIxBDL)**

**Table 12-2. 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 BUSCLK/(16×BR). See also BR bits in <a href="#">Table 12-1</a> .

## 12.2.2 SCI Control Register 1 (SCIxC1)

This read/write register controls various optional features of the SCI system.

**Figure 12-5. SCI Control Register 1 (SCIxC1)****Table 12-3. SCIxC1 Field Descriptions**

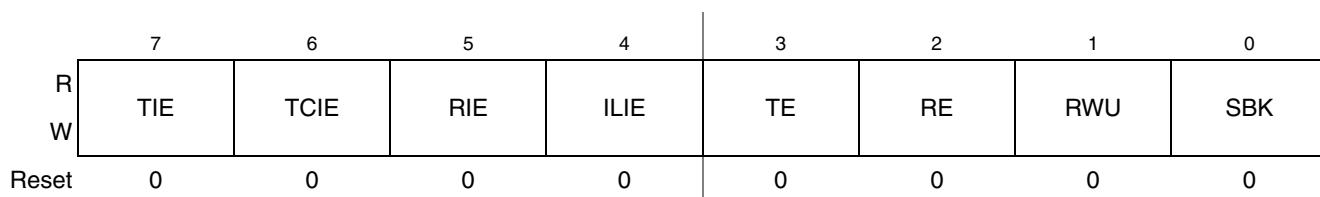
Field	Description
7 LOOPS	Loop Mode Select. Selects between loop back modes and normal 2-pin full-duplex modes. When LOOPS is set, the transmitter output is internally connected to the receiver input. 0 Normal operation — RxD and TxD use separate pins. 1 Loop mode or single-wire mode where transmitter outputs are internally connected to receiver input. (See <a href="#">RSRC</a> 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 12.3.3.2, “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 12.3.3.2.1, “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 12-3. 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.

### 12.2.3 SCI Control Register 2 (SCIx C2)

This register can be read or written at any time.

**Figure 12-6. SCI Control Register 2 (SCIx C2)****Table 12-4. SCIx C2 Field Descriptions**

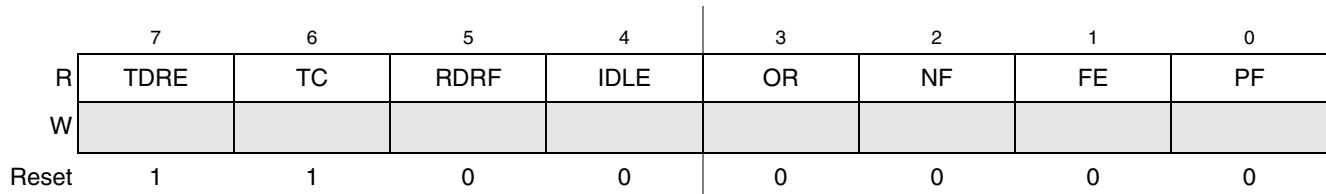
Field	Description
7 TIE	Transmit Interrupt Enable (for TDRE) 0 Hardware interrupts from TDRE disabled (use polling). 1 Hardware interrupt requested when TDRE flag is 1.
6 TCIE	Transmission Complete Interrupt Enable (for TC) 0 Hardware interrupts from TC disabled (use polling). 1 Hardware interrupt requested 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 12-4. 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 12.3.2.1, “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 12.3.3.2, “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 12.3.2.1, “Send Break and Queued Idle”</a> for more details. 0 Normal transmitter operation. 1 Queue break character(s) to be sent.

## 12.2.4 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 12-7. SCI Status Register 1 (SCIxS1)**

**Table 12-5. SCIxS1 Field Descriptions**

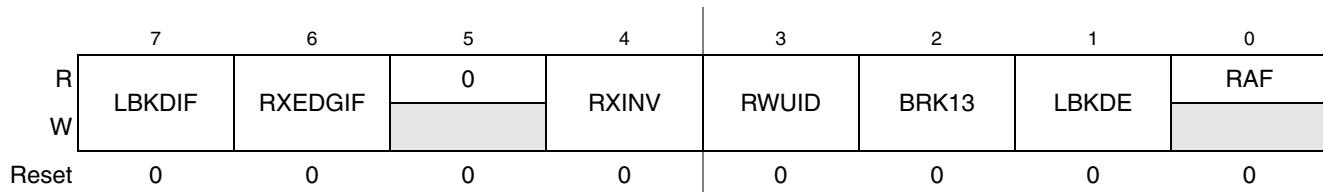
<b>Field</b>	<b>Description</b>
7 TDRE	Transmit Data Register Empty Flag. TDRE is set out of reset and when a transmit data value transfers from the transmit data buffer to the transmit shifter, leaving room for a new character in the buffer. To clear TDRE, read SCIxS1 with TDRE set and then write to 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 SCIxC2</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 12-5. 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.

## 12.2.5 SCI Status Register 2 (SCIxS2)

This register contains one read-only status flag.

**Figure 12-8. SCI Status Register 2 (SCIxS2)****Table 12-6. 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)

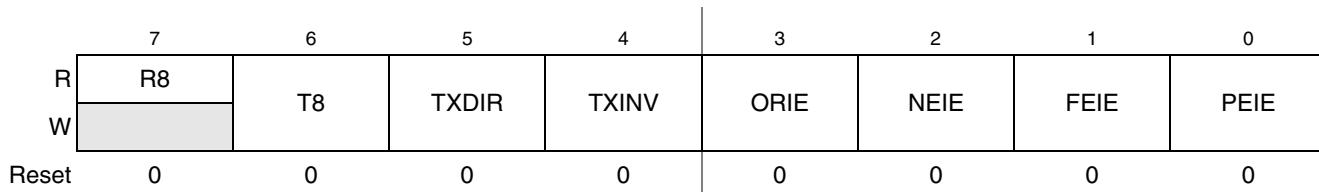
**Table 12-6. SCIxS2 Field Descriptions (continued)**

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.

## 12.2.6 SCI Control Register 3 (SCIxC3)

**Figure 12-9. SCI Control Register 3 (SCIxC3)****Table 12-7. 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 12-7. SCIx3 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.

### 12.2.7 SCI Data Register (SCIxD)

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

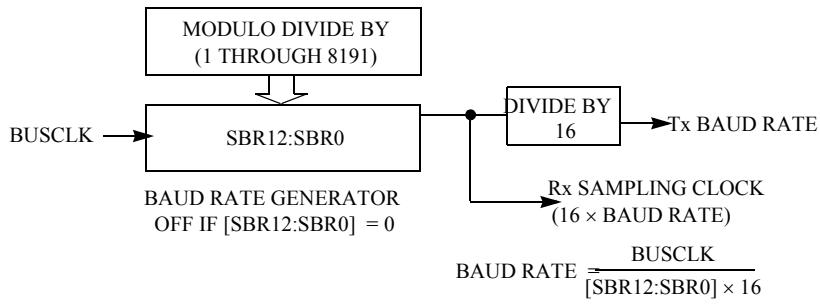
**Figure 12-10. SCI Data Register (SCIxD)**

### 12.3 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.

#### 12.3.1 Baud Rate Generation

As shown in [Figure 12-11](#), the clock source for the SCI baud rate generator is the bus-rate clock.

**Figure 12-11. 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.

### 12.3.2 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 12-1](#).

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 SCIxC2. 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 (SCIxD).

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 SCIxD.

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.

### 12.3.2.1 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 12-8. 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

### 12.3.3 Receiver Functional Description

In this section, the receiver block diagram ([Figure 12-2](#)) 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 12.3.4, "Interrupts and Status Flags,"](#) for more details about flag clearing.

### 12.3.3.1 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.

### 12.3.3.2 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.

### 12.3.3.2.1 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.

### 12.3.3.2.2 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.

## 12.3.4 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).

### 12.3.5 Additional SCI Functions

The following sections describe additional SCI functions.

#### 12.3.5.1 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.

### 12.3.5.2 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.

### 12.3.5.3 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.

### 12.3.5.4 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 13

## Inter-Integrated Circuit (S08IICV2)

### 13.1 Introduction

The inter-integrated circuit (IIC) provides a method of communication between a number of devices. 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 bus 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.

#### NOTE

The SDA and SCL should not be driven above  $V_{DD}$ . These pins are pseudo open-drain containing a protection diode to  $V_{DD}$ .

For details on low-power mode operation, refer to [Table 3-2](#) in [Chapter 3, “Modes of Operation”](#).

### 13.1.1 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
- Acknowledge bit generation/detection
- Bus busy detection
- General call recognition
- 10-bit address extension

### 13.1.2 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 wake-up 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.

### 13.1.3 Block Diagram

Figure 13-1 is a block diagram of the IIC.

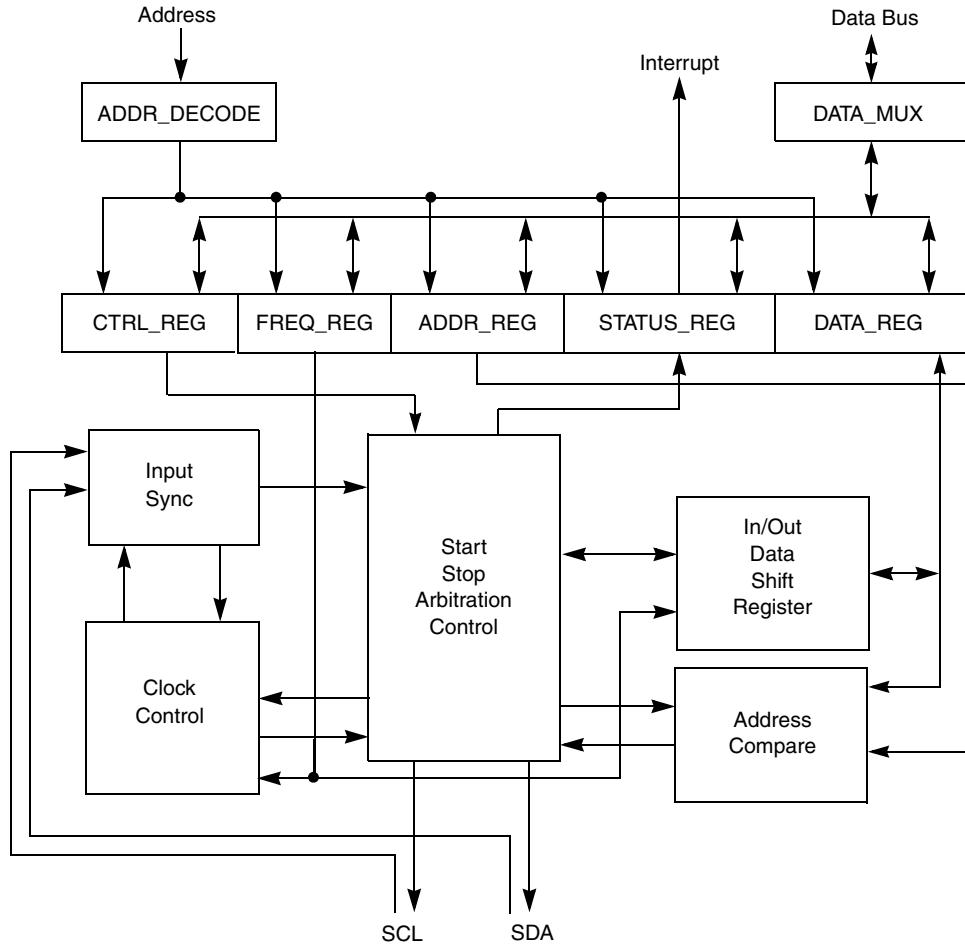


Figure 13-1. IIC Functional Block Diagram

## 13.2 External Signal Description

This section describes each user-accessible pin signal.

### 13.2.1 SCL — Serial Clock Line

The bidirectional SCL is the serial clock line of the IIC system.

### 13.2.2 SDA — Serial Data Line

The bidirectional SDA is the serial data line of the IIC system.

## 13.3 Register Definition

This section consists of the IIC register descriptions in address order.

Refer to the direct-page register summary in the [memory](#) chapter of this document for the absolute address assignments for all IIC 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.

### 13.3.1 IIC Address Register (IICxA)

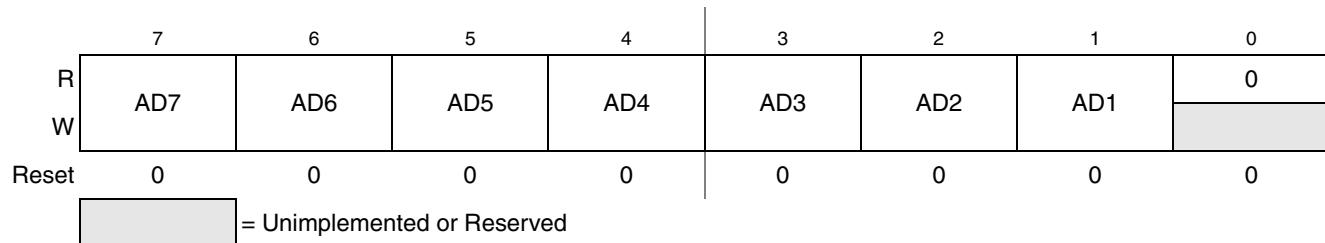


Figure 13-2. IIC Address Register (IICxA)

Table 13-1. IICxA Field Descriptions

Field	Description
7–1 AD[7:1]	Slave Address. The AD[7:1] field contains the slave address to be used by the IIC module. This field is used on the 7-bit address scheme and the lower seven bits of the 10-bit address scheme.

### 13.3.2 IIC Frequency Divider Register (IICxF)

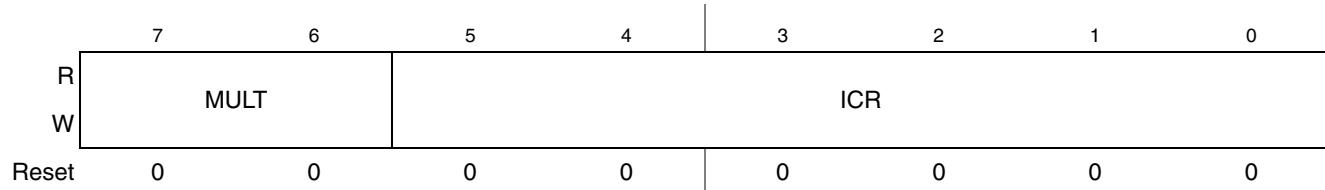


Figure 13-3. IIC Frequency Divider Register (IICxF)

**Table 13-2. IICxF Field Descriptions**

Field	Description
7–6 MULT	IIC Multiplier Factor. The MULT bits define the multiplier factor, mul. This factor, along with the SCL divider, generates 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	IIC Clock Rate. The ICR bits are used to prescale the bus clock for bit rate selection. These bits and the MULT bits determine the IIC baud rate, the SDA hold time, the SCL Start hold time, and the SCL Stop hold time. <a href="#">Table 13-4</a> provides the SCL divider and hold values for corresponding values of the ICR.  The SCL divider multiplied by multiplier factor mul generates IIC baud rate.  $\text{IIC baud rate} = \frac{\text{bus speed (Hz)}}{\text{mul} \times \text{SCLdivider}} \quad \text{Eqn. 13-1}$ SDA hold time is the delay from the falling edge of SCL (IIC clock) to the changing of SDA (IIC data).  $\text{SDA hold time} = \text{bus period (s)} \times \text{mul} \times \text{SDA hold value} \quad \text{Eqn. 13-2}$ 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).  $\text{SCL Start hold time} = \text{bus period (s)} \times \text{mul} \times \text{SCL Start hold value} \quad \text{Eqn. 13-3}$ 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).  $\text{SCL Stop hold time} = \text{bus period (s)} \times \text{mul} \times \text{SCL Stop hold value} \quad \text{Eqn. 13-4}$

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 100kbps.

**Table 13-3. Hold Time Values for 8 MHz Bus Speed**

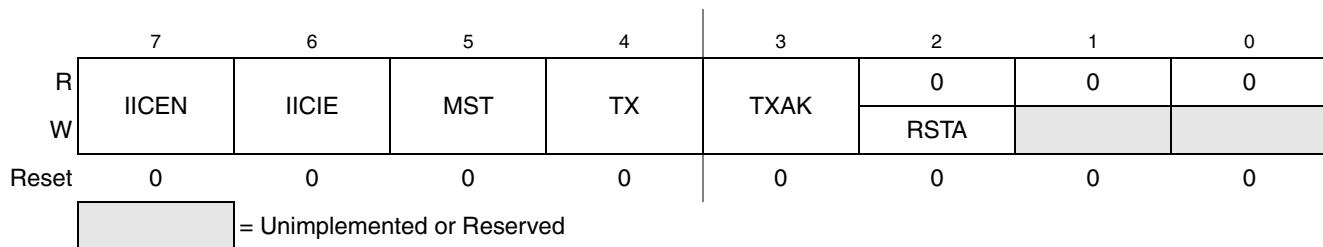
MULT	ICR	Hold Times (μs)		
		SDA	SCL Start	SCL Stop
0x2	0x00	3.500	4.750	5.125
0x1	0x07	2.500	4.250	5.125
0x1	0x0B	2.250	4.000	5.250
0x0	0x14	2.125	4.000	5.250
0x0	0x18	1.125	3.000	5.500

**Table 13-4. 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>SCL 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

### 13.3.3 IIC Control Register (IICxC1)



**Figure 13-4. IIC Control Register (IICxC1)**

**Table 13-5. IICxC1 Field Descriptions**

Field	Description
7 IICEN	IIC Enable. The IICEN bit determines whether the IIC module is enabled. 0 IIC is not enabled 1 IIC is enabled
6 IICIE	IIC Interrupt Enable. The IICIE bit determines whether an IIC interrupt is requested. 0 IIC interrupt request not enabled 1 IIC interrupt request enabled
5 MST	Master Mode Select. The MST bit changes from a 0 to a 1 when 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	Transmit Mode Select. 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	Transmit Acknowledge Enable. This bit specifies the value driven onto the SDA during data acknowledge cycles for master and slave receivers. 0 An acknowledge signal is sent out to the bus after receiving one data byte 1 No acknowledge signal response is sent
2 RSTA	Repeat start. Writing a 1 to this bit generates a repeated start condition provided it is the current master. This bit is always read as cleared. Attempting a repeat at the wrong time results in loss of arbitration.

### 13.3.4 IIC Status Register (IICxS)

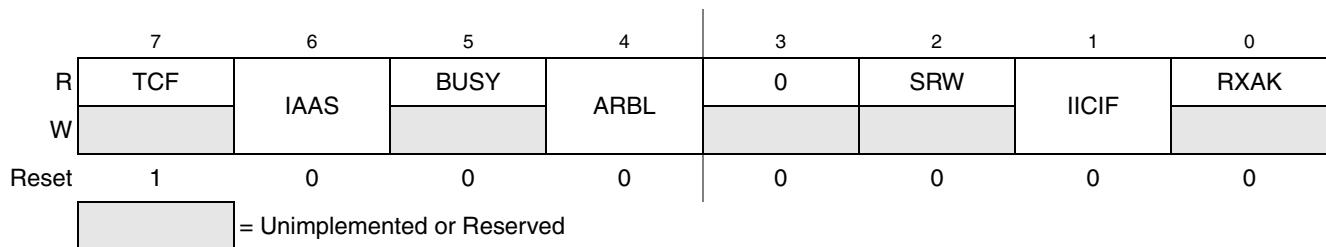


Figure 13-5. IIC Status Register (IICxS)

Table 13-6. IICxS Field Descriptions

Field	Description
7 TCF	Transfer Complete Flag. This bit is set on the completion of a byte transfer. 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 IICxD register in receive mode or writing to the IICxD in transmit mode. 0 Transfer in progress 1 Transfer complete
6 IAAS	Addressed as a Slave. The IAAS bit is set when the calling address matches the programmed slave address or when the GCAEN bit is set and a general call is received. Writing the IICxC register clears this bit. 0 Not addressed 1 Addressed as a slave
5 BUSY	Bus Busy. 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	Arbitration Lost. 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	Slave Read/Write. 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
1 IICIF	IIC Interrupt Flag. 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 completes</li><li>• Match of slave address to calling address</li><li>• Arbitration lost</li></ul> 0 No interrupt pending 1 Interrupt pending
0 RXAK	Receive Acknowledge. When the RXAK bit is low, it indicates an acknowledge signal has been 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

### 13.3.5 IIC Data I/O Register (IICxD)

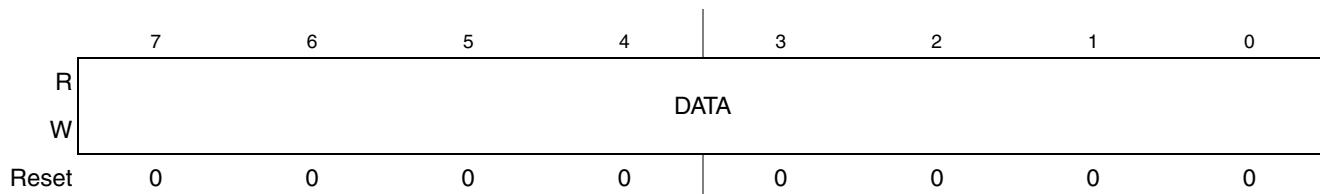


Figure 13-6. IIC Data I/O Register (IICxD)

Table 13-7. IICxD Field Descriptions

Field	Description
7-0 DATA	<b>Data</b> — In master transmit mode, when data is written to the IICxD, 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 IICxD 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 IICxC must correctly reflect the desired direction of transfer in master and slave modes for the transmission to begin. For instance, if the IIC is configured for master transmit but a master receive is desired, reading the IICxD does not initiate the receive.

Reading the IICxD returns the last byte received while the IIC is configured in master receive or slave receive modes. The IICxD does not reflect every byte transmitted on the IIC bus, nor can software verify that a byte has been written to the IICxD correctly by reading it back.

In master transmit mode, the first byte of data written to IICxD following assertion of MST 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).

### 13.3.6 IIC Control Register 2 (IICxC2)

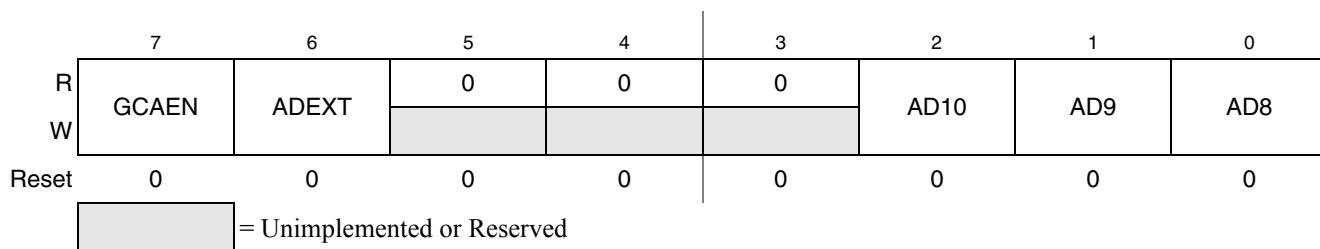


Figure 13-7. IIC Control Register (IICxC2)

**Table 13-8. IICxC2 Field Descriptions**

Field	Description
7 GCAEN	General Call Address Enable. The GCAEN bit enables or disables general call address. 0 General call address is disabled 1 General call address is enabled
6 ADEXT	Address Extension. 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]	Slave Address. 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.

## 13.4 Functional Description

This section provides a complete functional description of the IIC module.

### 13.4.1 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 13-8](#).

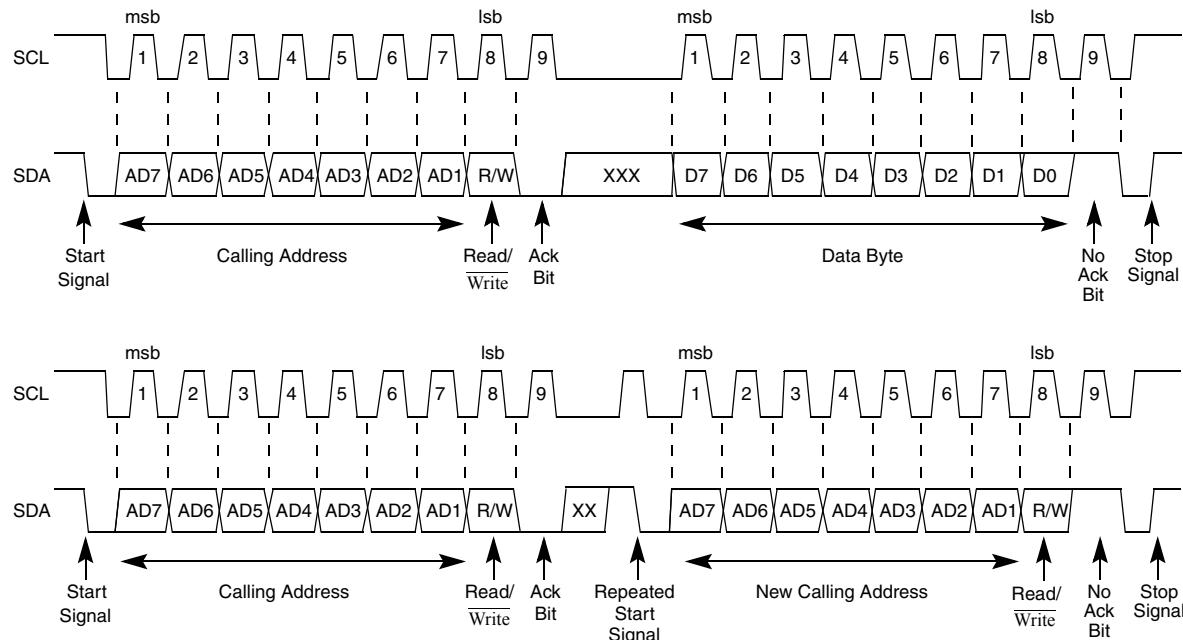


Figure 13-8. IIC Bus Transmission Signals

### 13.4.1.1 Start Signal

When the bus is free, no master device is engaging the bus (SCL and SDA lines are at logical high), a master may initiate communication by sending a start signal. As shown in Figure 13-8, 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.

### 13.4.1.2 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 13-8).

No two slaves in the system may have the same address. If the IIC module is the master, it must not transmit an address 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 operates correctly even if it is being addressed by another master.

### 13.4.1.3 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 13-8](#). There is one clock pulse on SCL for each data bit, the msb being transferred first. Each data byte is followed by a 9th (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.

### 13.4.1.4 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 13-8](#)).

The master can generate a stop even if the slave has generated an acknowledge at which point the slave must release the bus.

### 13.4.1.5 Repeated Start Signal

As shown in [Figure 13-8](#), 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.

### 13.4.1.6 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.

### 13.4.1.7 Clock Synchronization

Because wire-AND logic is performed on the SCL line, a high-to-low transition on the SCL line affects all the devices connected on the bus. The devices start counting their low period and after a device's clock has gone low, it holds the SCL line low until the clock high state is reached. However, the change of low to high in this device clock may not change the state of the SCL line if another device clock remains within its low period. Therefore, synchronized clock SCL is held low by the device with the longest low period. Devices with shorter low periods enter a high wait state during this time (see Figure 13-9). 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.

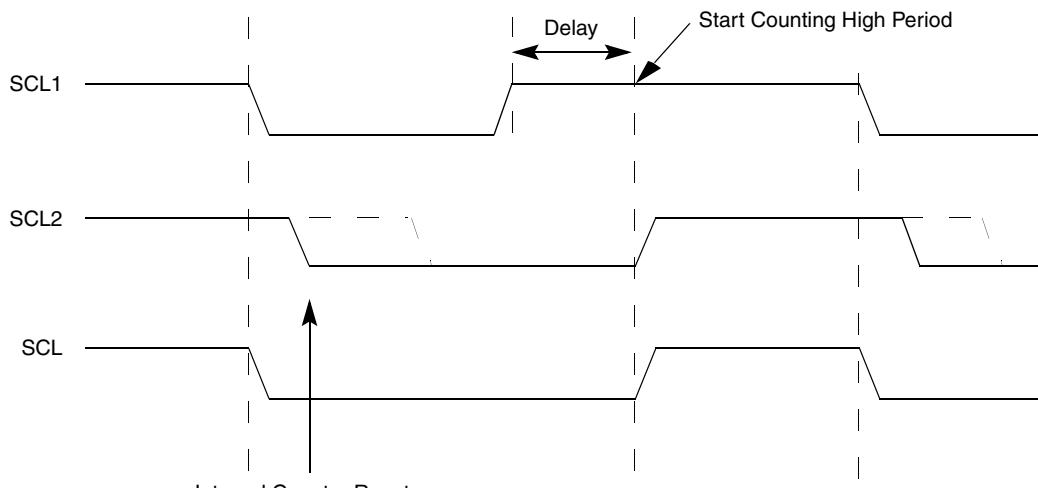


Figure 13-9. IIC Clock Synchronization

### 13.4.1.8 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 a case, it halts the bus clock and forces the master clock into wait states until the slave releases the SCL line.

### 13.4.1.9 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.

## 13.4.2 10-bit Address

For 10-bit addressing, 0x11110 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.

### 13.4.2.1 Master-Transmitter Addresses a Slave-Receiver

The transfer direction is not changed (see [Table 13-9](#)). 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. More than one device can find a match and generate an acknowledge (A1). Then, each slave that finds a match compares the eight bits of the second byte of the slave address with its own address. Only one slave finds a match and generates 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 13-9. 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 sees an IIC interrupt. Software must ensure the contents of IICD are ignored and not treated as valid data for this interrupt.

### 13.4.2.2 Master-Receiver Addresses a Slave-Transmitter

The transfer direction is changed after the second R/W bit (see [Table 13-10](#)). 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 has been 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 13-10. 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 sees an IIC interrupt. Software must ensure the contents of IICD are ignored and not treated as valid data for this interrupt.

### 13.4.3 General Call Address

General calls can be requested in 7-bit address or 10-bit address. If the GCAEN bit is set, the IIC matches the general call address as well as its own slave address. When the IIC responds to a general call, it acts as a slave-receiver and the IAAS bit is set after the address cycle. Software must read the IICD register after the first byte transfer to determine whether the address matches its own slave address or a general call. If the value is 00, the match is a general call. If the GCAEN bit is clear, the IIC ignores any data supplied from a general call address by not issuing an acknowledgement.

## 13.5 Resets

The IIC is disabled after reset. The IIC cannot cause an MCU reset.

## 13.6 Interrupts

The IIC generates a single interrupt.

An interrupt from the IIC is generated when any of the events in [Table 13-11](#) 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. You can determine the interrupt type by reading the status register.

**Table 13-11. 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

### 13.6.1 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.

### 13.6.2 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.

### 13.6.3 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 writing a 1 to it.

## 13.7 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: IICA
  - 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 13-11](#)

### 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 13-11](#)
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.)

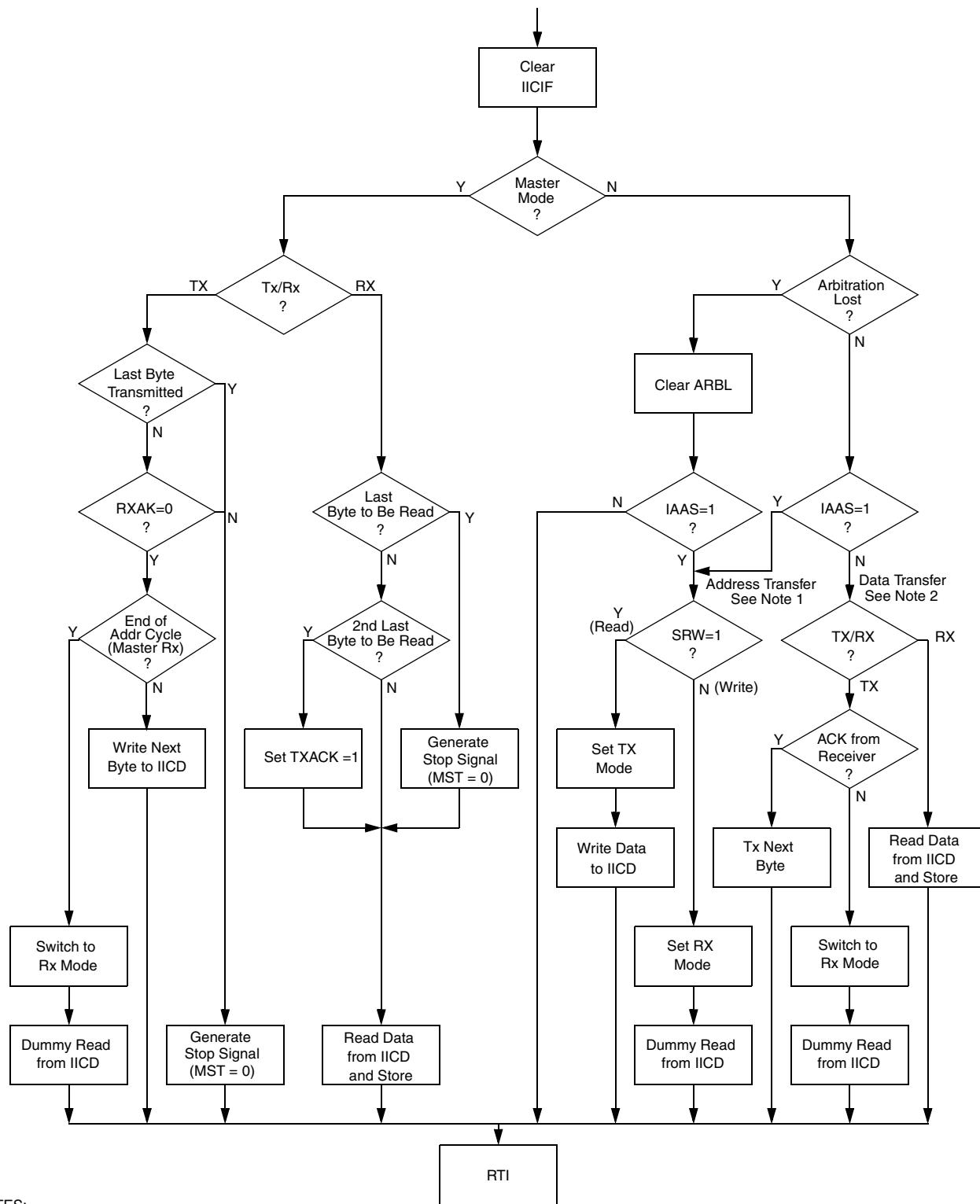
### Module Use

The routine shown in [Figure 13-11](#) can manage 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

IICA	AD[7:1] 0							
When addressed as a slave (in slave mode), the module responds to this address								
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								

**Figure 13-10. IIC Module Quick Start**



## 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 sees 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.

Figure 13-11. Typical IIC Interrupt Routine

# 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.

The SPI also supports a data length of 8 or 16 bits and includes a hardware match feature for the receive data buffer.

#### NOTE

SPI1 does not have a FIFO; the FIFO is present only on SPI2.

#### 14.1.1 Features

The SPI includes these distinctive features:

- Master mode or slave mode operation
- Full-duplex or single-wire bidirectional mode
- Programmable transmit bit rate
- Double-buffered transmit and receive data register
- Serial clock phase and polarity options
- Slave select output
- Mode fault error flag with CPU interrupt capability
- Control of SPI operation during wait mode
- Selectable MSB-first or LSB-first shifting
- Programmable 8- or 16-bit data transmission length
- Receive data buffer hardware match feature
- 64bit FIFO mode for high speed/large amounts of data transfers.

#### 14.1.2 Modes of Operation

For details on low-power mode operation, refer to Table 3-2 in Chapter 3, “Modes of Operation”.

The SPI functions in three modes, run, wait, and stop.

- Run Mode

This is the basic mode of operation.

- Wait Mode

SPI operation in wait mode is a configurable low power mode, controlled by the SPISWAI bit located in the SPIxC2 register. In wait mode, if the SPISWAI bit is clear, the SPI operates like in Run Mode. If the SPISWAI bit is set, the SPI goes into a power conservative state, with the SPI clock generation turned off. If the SPI is configured as a master, any transmission in progress stops, but is resumed after CPU goes into Run Mode. If the SPI is configured as a slave, reception and transmission of a byte continues, so that the slave stays synchronized to the master.

- Stop Mode

The SPI is inactive in stop3/stop4 mode for reduced power consumption. If the SPI is configured as a master, any transmission in progress stops, but is resumed after the CPU goes into Run Mode. If the SPI is configured as a slave, reception and transmission of a data continues, so that the slave stays synchronized to the master.

The SPI is completely disabled in all other stop modes. When the CPU wakes from these stop modes, all SPI register content is reset.

This is a high level description only, detailed descriptions of operating modes are contained in section [Section 14.4.10, “Low Power Mode Options.”](#)

## 14.1.3 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.

### 14.1.3.1 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 (SS pin). In this system, the master device has configured its SS pin as an optional slave select output.

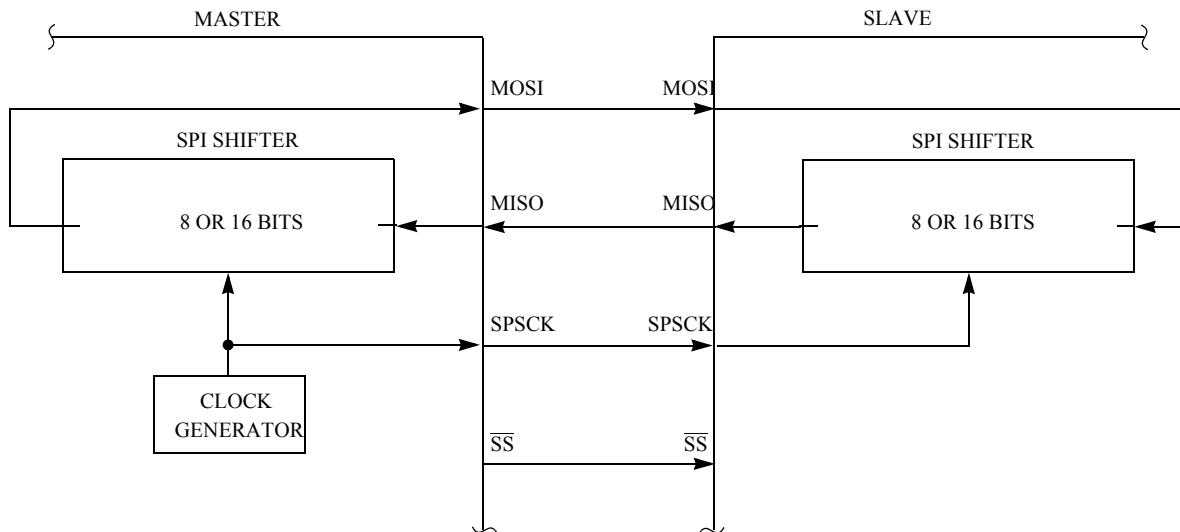


Figure 14-1. SPI System Connections

#### 14.1.3.2 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 SPIxDH:SPIxDL) and gets transferred to the SPI shift register at the start of a data transfer. After shifting in 8 or 16 bits (as determined by SPIMODE bit) of data, the data is transferred into the double-buffered receiver where it can be read (read from SPIxDH:SPIxDL). Pin multiplexing logic controls connections between MCU pins and the SPI module.

Additionally there is an 8-byte receive FIFO and an 8-byte transmit FIFO that once enabled provide features to allow less CPU interrupts to occur when transmitting/receiving high volume/high speed data. When FIFO mode is enabled, the SPI can function in 8-bit or 16-bit mode ( as per SPIMODE bit) and 3 additional flags help monitor the FIFO status and two of these flags can provide CPU interrupts.

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.

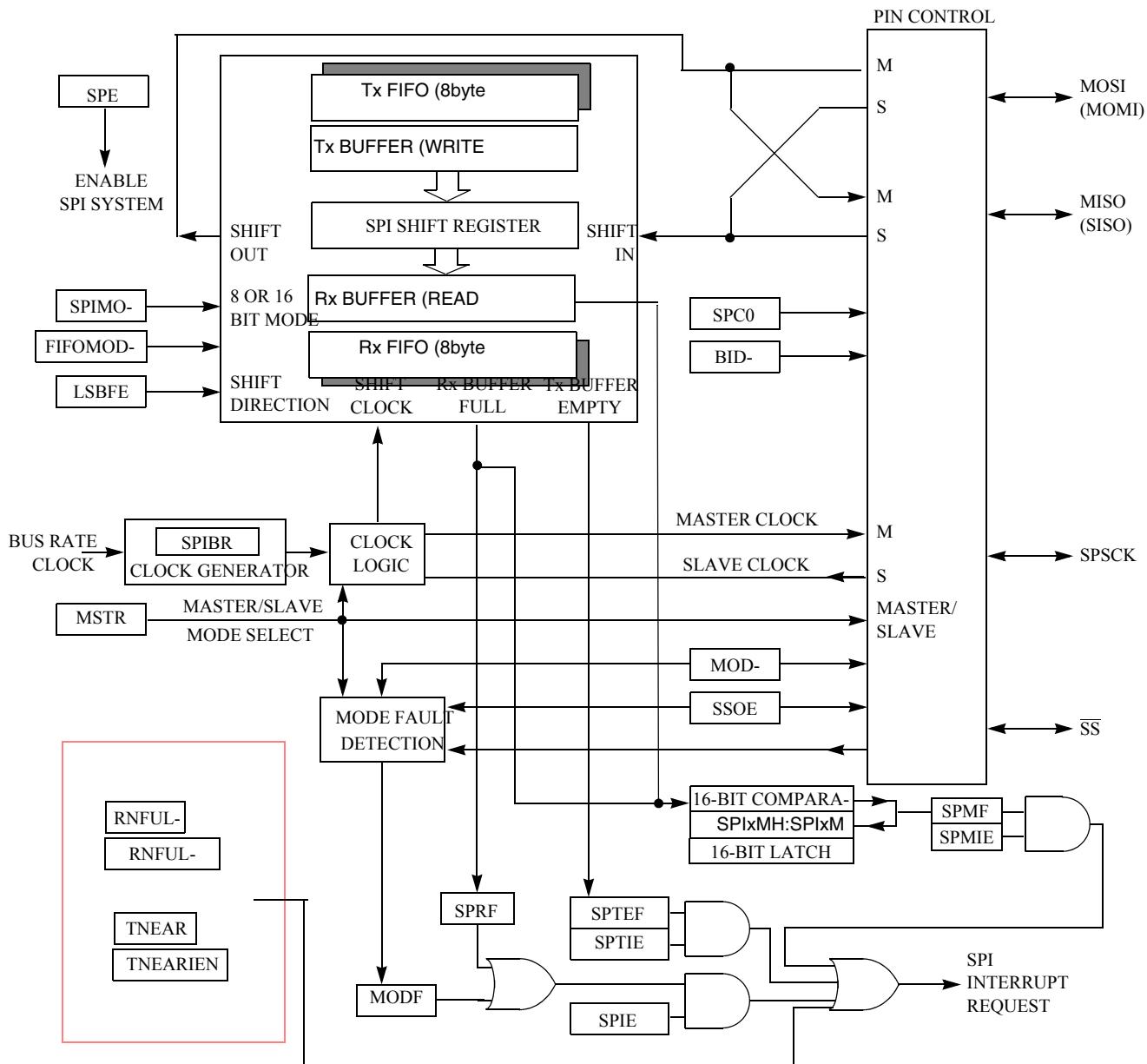


Figure 14-2. SPI Module Block Diagram

## 14.2 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 = 1 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 = 0, this pin is the serial data output. If SPC0 = 1 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 = 1 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 $\overline{\text{SS}}$ — 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 = 1, 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 Register Definition

The SPI has eight 8-bit registers to select SPI options, control baud rate, report SPI status, hold an SPI data match value, and for transmit/receive data.

Refer to the direct-page register summary in the Memory chapter of this data sheet 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.

### 14.3.1 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

Figure 14-3. SPI Control Register 1 (SPIxC1)

Table 14-1. SPIxC1 Field Descriptions

Field	Description
7 SPIE	<p><b>FIFOMODE=0</b>  <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</p> <p><b>FIFOMODE=1</b>  <b>SPI Read FIFO Full Interrupt Enable</b> — This bit when set enables the SPI to interrupt the CPU when the Receive FIFO is full. An interrupt occurs when SPRF flag is set or MODF is set.            0 Read FIFO Full Interrupts are disabled            1 Read FIFO Full Interrupts are enabled</p>
6 SPE	<p><b>SPI System Enable</b> — This bit enables the SPI system and dedicates the SPI port pins to SPI system functions. If SPE is cleared, SPI is disabled and forced into idle state, and all status bits in the SPIxS register are reset.</p> <p>0 SPI system inactive            1 SPI system enabled</p>
5 SPTIE	<p><b>SPI Transmit Interrupt Enable</b> —</p> <p><b>FIFOMODE=0</b>            This is the interrupt enable bit for SPI transmit buffer empty (SPTEF). An interrupt occurs when the SPI transmit buffer is empty (SPTEF is set)</p> <p><b>FIFOMODE=1</b>            This is the interrupt enable bit for SPI transmit FIFO empty (SPTEF). An interrupt occurs when the SPI transmit FIFO is empty (SPTEF is set)</p> <p>0 Interrupts from SPTEF inhibited (use polling)            1 When SPTEF is 1, hardware interrupt requested</p>
4 MSTR	<p><b>Master/Slave Mode Select</b> — This bit selects master or slave mode operation.</p> <p>0 SPI module configured as a slave SPI device            1 SPI module configured as a master SPI device</p>
3 CPOL	<p><b>Clock Polarity</b> — This bit selects an inverted or non-inverted SPI clock. To transmit data between SPI modules, the SPI modules must have identical CPOL values.</p> <p>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 14.4.6, “SPI Clock Formats”</a> for more details.</p> <p>0 Active-high SPI clock (idles low)            1 Active-low SPI clock (idles high)</p>

**Table 14-1. SPIxC1 Field Descriptions (continued)**

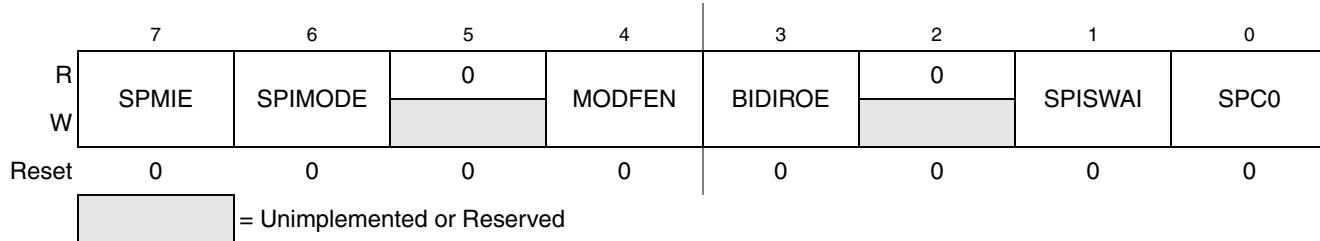
Field	Description
2 CPHA	<b>Clock Phase</b> — This bit selects one of two clock formats for different kinds of synchronous serial peripheral devices. Refer to <a href="#">Section 14.4.6, “SPI Clock Formats”</a> for more details. 0 First edge on SPSCK occurs at the middle of the first cycle of a data transfer 1 First edge on SPSCK occurs at the start of the first cycle of a data transfer
1 SSOE	<b>Slave Select Output Enable</b> — This bit is used in combination with the mode fault enable (MODFEN) bit in SPIxC2 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> — This bit does not affect the position of the MSB and LSB in the data register. Reads and writes of the data register always have the MSB in bit 7 (or bit 15 in 16-bit mode). 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

### 14.3.2 SPI Control Register 2 (SPIxC2)

This read/write register is used to control optional features of the SPI system. Bits 6 and 5 are not implemented and always read 0.

**Figure 14-4. SPI Control Register 2 (SPIxC2)****Table 14-3. SPIxC2 Register Field Descriptions**

Field	Description
7 SPMIE	<b>SPI Match Interrupt Enable</b> — This is the interrupt enable for the SPI receive data buffer hardware match (SPMF) function. 0 Interrupts from SPMF inhibited (use polling). 1 When SPMF = 1, requests a hardware interrupt.
6 SPIMODE	<b>SPI 8- or 16-bit Mode</b> — This bit allows the user to select an 8-bit or 16-bit SPI data transmission length. In master mode, a change of this bit aborts a transmission in progress, force the SPI system into idle state, and reset all status bits in the SPIxS register. Refer to section <a href="#">Section 14.4.5, “Data Transmission Length,”</a> for details. 0 8-bit SPI shift register, match register, and buffers. 1 16-bit SPI shift register, match register, and buffers.

**Table 14-3. SPIxC2 Register Field Descriptions (continued)**

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 $\overline{SS}$ pin is the slave select input.) In master mode, this bit determines how the $\overline{SS}$ pin is used (refer to <a href="#">Table 14-2</a> for details) 0 Mode fault function disabled, master $\overline{SS}$ pin reverts to general-purpose I/O not controlled by SPI 1 Mode fault function enabled, master $\overline{SS}$ 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 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> — This bit is used for power conservation while in wait. 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> — This bit enables bidirectional pin configurations as shown in <a href="#">Table 14-4</a> . 0 SPI uses separate pins for data input and data output. 1 SPI configured for single-wire bidirectional operation.

**Table 14-4. Bidirectional Pin Configurations**

Pin Mode	SPC0	BIDIROE	MISO	MOSI
<b>Master Mode of Operation</b>				
Normal	0	X	Master In	Master Out
Bidirectional	1	0	MISO not used by SPI	Master In
		1		Master I/O
<b>Slave Mode of Operation</b>				
Normal	0	X	Slave Out	Slaveln
Bidirectional	1	0	Slave In	MOSI not used by SPI
		1	Slave I/O	

### 14.3.3 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 14-5. SPI Baud Rate Register (SPIxBR)**

**Table 14-5. 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 14-6</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 14-18</a> ). See <a href="#">Section 14.4.7, “SPI Baud Rate Generation,”</a> for details.
2:0 SPR[2:0]	<b>SPI Baud Rate Divisor</b> — This 3-bit field selects one of eight divisors for the SPI baud rate divider as shown in <a href="#">Table 14-7</a> . The input to this divider comes from the SPI baud rate prescaler (see <a href="#">Figure 14-18</a> ). See <a href="#">Section 14.4.7, “SPI Baud Rate Generation,”</a> for details.

**Table 14-6. 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 14-7. 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

### 14.3.4 SPI Status Register (SPIxS)

This register has eight read-only status bits. Writes have no meaning or effect. This register has 4 additional

	7	6	5	4	3	2	1	0
R	SPRF	SPMF	SPTEF	MODF	RNFULLF	TNEAREF	TXFULLF	RFIFOEOF
W								
Reset	0	0	1	0	0	0 <sup>1</sup>	0	0 <sup>1</sup>
	= Unimplemented or Reserved							

Figure 14-6. SPI Status Register (SPIxS)

<sup>1</sup> Note: PoR values of TNEAREF and RFIFOEOF is 0. If status register is reset due to change of SPIMODE, FIFOMODE or SPE than, if FIFOMODE = 1, TNEAREF and RFIFOEOF resets to 1 else if FIFOMODE = 0, TNEAREF and RFIFOEOF resets to 0

Flags RNFULLF, TNEAREF, TXFULLF and RFIFOEOF provide mechanisms to support an 8-byte FIFO mode. When in 8byte FIFO mode the function of SPRF and SPTEF differs slightly from the normal buffered modes, mainly in how these flags are cleared by the amount available in the transmit and receive FIFOs

Table 14-8. SPIxS Register Field Descriptions

Field	Description
7 SPRF	<p><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 (SPIxDH:SPIxDL). SPRF is cleared by reading SPRF while it is set, then reading the SPI data register.</p> <p>0 No data available in the receive data buffer. 1 Data available in the receive data buffer.</p> <p><b>FIFOMODE=1</b></p> <p><b>SPI Read FIFO FULL Flag</b> — This bit indicates the status of the Read FIFO when FIFOMODE enabled. The SPRF is set when the read FIFO has received 64bits (4 words or 8bytes) of data from the shifter and there has been no CPU reads of SPIxDH:SPIxDL. SPRF is cleared by reading the SPI Data Register, which empties the FIFO, assuming another SPI message is not received.</p> <p>0 Read FIFO is not Full 1 Read FIFO is Full.</p>
6 SPMF	<p><b>SPI Match Flag</b> — SPMF is set after SPRF = 1 when the value in the receive data buffer matches the value in SPIxMH:SPIxML. To clear the flag, read SPMF when it is set, then write a 1 to it.</p> <p>0 Value in the receive data buffer does not match the value in SPIxMH:SPIxML registers. 1 Value in the receive data buffer matches the value in SPIxMH:SPIxML registers.</p>

**Table 14-8. SPIxS Register Field Descriptions**

Field	Description
5 SPTEF	<p><b>SPI Transmit Buffer Empty Flag</b> — This bit is set when the transmit data buffer is empty. It is cleared by reading SPIxS with SPTEF set, followed by writing a data value to the transmit buffer at SPIxDH:SPIxDL. SPIxS must be read with SPTEF = 1 before writing data to SPIxDH:SPIxDL or the SPIxDH:SPIxDL write is ignored. SPTEF is automatically set when all data from the transmit buffer transfers into the transmit shift register. For an idle SPI, data written to SPIxDH:SPIxDL is transferred to the shifter almost immediately so SPTEF is set within two bus cycles allowing a second data to be queued into the transmit buffer. After completion of the transfer of the data in the shift register, the queued data 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.</p> <p>0 SPI transmit buffer not empty 1 SPI transmit buffer empty</p> <p><b>FIFOMODE=1</b></p> <p><b>SPI Transmit FIFO Empty Flag</b> — <i>This bit when in FIFOMODE now changed to provide status of the FIFO rather than an 8or16-bit buffer.</i> This bit is set when the Transmit FIFO is empty. It is cleared by writing a data value to the transmit FIFO at SPIxDH:SPIxDL. SPTEF is automatically set when all data from transmit FIFO transfers into the transmit shift register. For an idle SPI, data written to SPIxDH:SPIxDL is transferred to the shifter almost immediately so SPTEF is set within two bus cycles, a second write of data to the SPIxDH:SPIxDL clears the SPTEF flag. After completion of the transfer of the data in the shift register, the queued data from the transmit FIFO automatically moves to the shifter and SPTEF is set only when all data written to the transmit FIFO has been transferred to the shifter. If no new data is waiting in the transmit FIFO, SPTEF simply remains set and no data moves from the buffer to the shifter.</p> <p>0 SPI FIFO not empty 1 SPI FIFO empty</p>
4 MODF	<p><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 = 1, MODFEN = 1, and SSOE = 0; otherwise, MODF is never set. MODF is cleared by reading MODF while it is 1, then writing to SPI control register 1 (SPIxC1).</p> <p>0 No mode fault error 1 Mode fault error detected</p>
3 RNFULLF	<p><b>Receive FIFO Nearly Full Flag</b> — This flag is set when more than three 16bit word or six 8bit bytes of data remain in the receive FIFO provided SPIxC3[4] = 0 or when more than two 16bit word or four 8bit bytes of data remain in the receive FIFO provided SPIxC3[4] = 1. It has no function if FIFOMODE=0.</p> <p>0 Receive FIFO has received less than 48bits/32bits (See SPIxC3[4]). 1 Receive FIFO has received 48bits/32bits(See SPIxC3[4]) or more.</p>
2 TNEAREF	<p><b>Transmit FIFO Nearly Empty Flag</b> — This flag is set when only one 16bit word or 2 8bit bytes of data remain in the transmit FIFO provided SPIxC3[5] = 0 or when only two 16bit word or 4 8bit bytes of data remain in the transmit FIFO provided SPIxC3[5] = 1. If FIFOMODE is not enabled this bit should be ignored.</p> <p>0 Transmit FIFO has more than 16bits/32bits (See SPIxC3[5]) left to transmit. 1 Transmit FIFO has 16bits/32 bits( See SPIxC3[5]) or less left to transmit</p>

**Table 14-8. SPIxS Register Field Descriptions**

Field	Description
1 TXFULLF	<b>Transmit FIFO Full Flag</b> - This bit indicates status of transmit fifo when fifomode is enabled. This flag is set when there are 8 bytes in transmit fifo. If FIFOMODE is not enabled this bit should be ignored. 0 Transmit FIFO has less than 8 bytes. 1 Transmit FIFO has 8 bytes of data.
0 RFIFOEF	<b>SPI Read FIFO Empty Flag</b> — This bit indicates the status of the Read FIFO when FIFOMODE enabled. If FIFOMODE is not enabled this bit should be ignored. 0 Read FIFO has data. Reads of the SPIxDH:SPIxDL registers in 16-bit mode or SPIxDL register in 8-bit mode empties the Read FIFO. 1 Read FIFO is empty.

For FIFO management there are two other important flags that are used to help make the operation more efficient when transferring large amounts of data. These are the Receive FIFO Nearly Full Flag (RNFULLF) and the Transmit FIFO Nearly Empty Flag (TNEAREF). Both of these flags provide a watermark feature of the FIFOs to allow continuous transmissions of data when running at high speed.

The RNFULLF flag can generate an interrupt if the RNFULLIEN bit in the SPIxC3 Register is set that allows the CPU to start emptying the Receive FIFO without delaying the reception of subsequent bytes. The user can also determine if all data in Receive FIFO has been read by monitoring the RFIFOEF flag.

The TNEAREF flag can generate an interrupt if the TNEARIEN bit n the SPIxC3 Register is set that allows the CPU to start filling the Transmit FIFO before it is empty and thus provide a mechanism to have no breaks in SPI transmission.

#### NOTE

SPIxS and both TX and RX fifos gets reset due to change in SPIMODE, FIFOMODE or SPE. PoR values of SPIxS are show in Figure 16-7.

Figure 14-7 and Figure 14-8 shows the reset values due to change of modes after PoR.

	7	6	5	4	3	2	1	0
R	SPRF	SPMF	SPTEF	MODF	RNFULLF	TNEAREF	TXFULLF	RFIFOEF
W								
Reset	0	0	1	0	0	0	0	0

**Figure 14-7. Reset values of SPIxS after PoR with FIFOMODE = 0**

	7	6	5	4	3	2	1	0
R	SPRF	SPMF	SPTEF	MODF	RNFULLF	TNEAREF	TXFULLF	RFIFOEF
W								
Reset	0	0	1	0	0	1	0	1

**Figure 14-8. Reset values of SPIxS after PoR with FIFOMODE = 1**

### 14.3.5 SPI Data Registers (SPIxDH:SPIxDL)

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

Figure 14-9. SPI Data Register High (SPIxDH)

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

Figure 14-10. SPI Data Register Low (SPIxDL)

The SPI data registers (SPIxDH:SPIxDL) are the input and output register for SPI data. A write to these registers writes to the transmit data buffer, allowing data to be queued and transmitted.

When the SPI is configured as a master, data queued in the transmit data buffer is transmitted immediately after the previous transmission has completed.

The SPI transmit buffer empty flag (SPTEF) in the SPIxS register indicates when the transmit data buffer is ready to accept new data. SPIxS must be read when SPTEF is set before writing to the SPI data registers or the write is ignored.

Data may be read from SPIxDH:SPIxDL 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.

In 8-bit mode, only SPIxDL is available. Reads of SPIxDH returns all 0s. Writes to SPIxDH are ignored.

In 16-bit mode, reading either byte (SPIxDH or SPIxDL) latches the contents of both bytes into a buffer where they remain latched until the other byte is read. Writing to either byte (SPIxDH or SPIxDL) latches the value into a buffer. When both bytes have been written, they are transferred as a coherent 16-bit value into the transmit data buffer.

### 14.3.6 SPI Match Registers (SPIxMH:SPIxML)

These read/write registers contain the hardware compare value, which sets the SPI match flag (SPMF) when the value received in the SPI receive data buffer equals the value in the SPIxMH:SPIxML registers.

In 8-bit mode, only SPIxML is available. Reads of SPIxMH returns all 0s. Writes to SPIxMH are ignored.

In 16-bit mode, reading either byte (SPIxMH or SPIxML) latches the contents of both bytes into a buffer where they remain latched until the other byte is read. Writing to either byte (SPIxMH or SPIxML) latches the value into a buffer. When both bytes have been written, they are transferred as a coherent value into the SPI match registers.

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

Reset      0      0      0      0      |      0      0      0      0      0

Figure 14-11. SPI Match Register High (SPIxMH)

	7	6	5	4		3	2	1	0
R	Bit 7	6	5	4		3	2	1	Bit 0
W	0	0	0	0		0	0	0	0

Reset      0      0      0      0      |      0      0      0      0      0

Figure 14-12. SPI Match Register Low (SPIxML)

### 14.3.7 SPI Control Register 3 (SPIxC3) — Enable FIFO Feature

The SPI Control Register 3 introduces a 64bit FIFO function on transmit and receive buffers to be utilised on the SPI. Utilising this FIFO feature allows the SPI to provide high speed transfers of large amounts of data without consuming large amounts of the CPU bandwidth.

Enabling this FIFO function affects the behaviour of some of the Read/Write Buffer flags in the SPIxS register. Namely:

- The SPRF of the SPIxS register is set when the Receive FIFO is filled and interrupts the CPU if the SPIE in the SPIxC1 register is set.

and

- The SPTEF of the SPIxS register is set when the Transmit FIFO is empty, and interrupts the CPU if the SPITIE bit is set in the SPIxC1 register. See SPIxC1 and SPIxS registers.

FIFO mode is enabled by setting the FIFOMODE bit, and provides the SPI with an 8-byte receive FIFO and an 8-byte transmit FIFO to reduce the amount of CPU interrupts for high speed/high volume data transfers.

Two interrupt enable bits TNEARIEN and RNFULLIEN provide CPU interrupts based on the watermark feature of the TNEARF and RNFULLF flags of the SPIxS register.

**Note:** This register has sixread/write control bits. Bits 7 thro' 6are not implemented and always read 0. Writes have no meaning or effect. Write to this register happens only when FIFOMODE bit is 1.

	7	6	5	4		3	2	1	0
R	0	0	TNEAREF MARK	RNFULL MARK		0	TNEARIEN	RNFULLIEN	FIFOMODE
W			0	0		0	0	0	0

Reset      0      0      0      0      |      0      0      0      0      0

 = Unimplemented or Reserved

Figure 14-13. SPI Control Register 3 (SPIxC3)

**Table 14-9. SPIxC3 Register Field Descriptions**

Field	Description
7–5	<b>Reserved, should be cleared.</b>
5 TNEAREF MARK	<b>Transmit FIFO Nearly Empty Water Mark</b> - This bit selects the mark after which TNEAREF flag is asserted. 0 TNEAREF is set when Transmit FIFO has 16bits or less. 1 TNEAREF is set when Transmit FIFO has 32bits or less.
4 RNFULLF MARK	<b>Receive FIFO Nearly Full Water Mark</b> - This bit selects the mark for which RNFULLF flag is asserted 0 RNFULLF is set when Receive FIFO has 48bits or more 1 RNFULLF is set when Receive FIFO has 32bits or more.
3	<b>Reserved, should be cleared.</b>
2 TNEARIEN	<b>Transmit FIFO Nearly Empty Interrupt Enable</b> — Writing to this bit enables the SPI to interrupt the CPU when the TNEAREF flag is set. This is an additional interrupt on the SPI and only interrupts the CPU if SPTIE in the SPIxC1 register is also set. This bit is ignored and has no function if FIFOMODE=0. 0 No interrupt on Transmit FIFO Nearly Empty Flag being set. 1 Enable interrupts on Transmit FIFO Nearly Empty Flag being set.
1 RNFULLIEN	<b>Receive FIFO Nearly Full Interrupt Enable</b> — Writing to this bit enables the SPI to interrupt the CPU when the RNEARFF flag is set. This is an additional interrupt on the SPI and only interrupts the CPU if SPIE in the SPIxC1 register is also set. This bit is ignored and has no function if FIFOMODE=0. 0 No interrupt on RNEARFF being set. 1 Enable interrupts on RNEARFF being set.
0 FIFOMODE	<b>SPI FIFO Mode Enable</b> — This bit enables the SPI to utilise a 64bit FIFO (8bytes 4 16-bit words) for transmit and receive buffers. 0 Buffer mode disabled. 1 Data available in the receive data buffer.

## 14.4 Functional Description

### 14.4.1 General

The SPI system is enabled by setting the SPI enable (SPE) bit in SPI Control Register 1. While the SPE bit is set, the four associated SPI port pins are dedicated to the SPI function as:

- Slave select ( $\overline{SS}$ )
- Serial clock (SPSCK)
- Master out/slave in (MOSI)
- Master in/slave out (MISO)

An SPI transfer is initiated in the master SPI device by reading the SPI status register (SPIxS) when SPTEF = 1 and then writing data to the transmit data buffer (write to SPIxDH:SPIxDL). When a transfer is complete, received data is moved into the receive data buffer. The SPIxDH:SPIxDL registers act as the SPI receive data buffer for reads and as the SPI transmit data buffer for writes.

The clock phase control bit (CPHA) and a clock polarity control bit (CPOL) in the SPI Control Register 1 (SPIxC1) select one of four possible clock formats to be used by the SPI system. The CPOL bit simply selects a non-inverted or inverted clock. The CPHA bit is used to accommodate two fundamentally different protocols by sampling data on odd numbered SPSCK edges or on even numbered SPSCK edges.

The SPI can be configured to operate as a master or as a slave. When the MSTR bit in SPI control register 1 is set, master mode is selected, when the MSTR bit is clear, slave mode is selected.

## 14.4.2 Master Mode

The SPI operates in master mode when the MSTR bit is set. Only a master SPI module can initiate transmissions. A transmission begins by reading the SPIxS register while SPTEF = 1 and writing to the master SPI data registers. If the shift register is empty, the byte immediately transfers to the shift register. The data begins shifting out on the MOSI pin under the control of the serial clock.

- SPSCK

The SPR2, SPR1, and SPR0 baud rate selection bits in conjunction with the SPPR2, SPPR1, and SPPR0 baud rate preselection bits in the SPI Baud Rate register control the baud rate generator and determine the speed of the transmission. The SPSCK pin is the SPI clock output. Through the SPSCK pin, the baud rate generator of the master controls the shift register of the slave peripheral.

- MOSI, MISO pin

In master mode, the function of the serial data output pin (MOSI) and the serial data input pin (MISO) is determined by the SPC0 and BIDIROE control bits.

- $\overline{\text{SS}}$  pin

If MODFEN and SSOE bit are set, the  $\overline{\text{SS}}$  pin is configured as slave select output. The  $\overline{\text{SS}}$  output becomes low during each transmission and is high when the SPI is in idle state.

If MODFEN is set and SSOE is cleared, the  $\overline{\text{SS}}$  pin is configured as input for detecting mode fault error. If the  $\overline{\text{SS}}$  input becomes low this indicates a mode fault error where another master tries to drive the MOSI and SPSCK lines. In this case, the SPI immediately switches to slave mode, by clearing the MSTR bit and also disables the slave output buffer MISO (or SISO in bidirectional mode). The result is that all outputs are disabled and SPSCK, MOSI and MISO are inputs. If a transmission is in progress when the mode fault occurs, the transmission is aborted and the SPI is forced into idle state.

This mode fault error also sets the mode fault (MODF) flag in the SPI Status Register (SPIxS). If the SPI interrupt enable bit (SPIE) is set when the MODF flag gets set, then an SPI interrupt sequence is also requested.

When a write to the SPI Data Register in the master occurs, there is a half SPSCK-cycle delay. After the delay, SPSCK is started within the master. The rest of the transfer operation differs slightly, depending on the clock format specified by the SPI clock phase bit, CPHA, in SPI Control Register 1 (see [Section 14.4.6, “SPI Clock Formats](#)).

### NOTE

A change of the bits CPOL, CPHA, SSOE, LSBFE, MODFEN, SPC0, BIDIROE with SPC0 set, SPIMODE, FIFOMODE, SPPR2-SPPR0, and SPR2-SPR0 in master mode aborts a transmission in progress and forces the SPI into idle state. The remote slave cannot detect this. Therefore, the master has to ensure that the remote slave is set back to idle state.

### 14.4.3 Slave Mode

The SPI operates in slave mode when the MSTR bit in SPI Control Register1 is clear.

- SPSCK

In slave mode, SPSCK is the SPI clock input from the master.

- MISO, MOSI pin

In slave mode, the function of the serial data output pin (MISO) and serial data input pin (MOSI) is determined by the SPC0 bit and BIDIROE bit in SPI Control Register 2.

- $\overline{\text{SS}}$  pin

The  $\overline{\text{SS}}$  pin is the slave select input. Before a data transmission occurs, the  $\overline{\text{SS}}$  pin of the slave SPI must be low.  $\overline{\text{SS}}$  must remain low until the transmission is complete. If  $\overline{\text{SS}}$  goes high, the SPI is forced into idle state.

The  $\overline{\text{SS}}$  input also controls the serial data output pin, if  $\overline{\text{SS}}$  is high (not selected), the serial data output pin is high impedance, and, if  $\overline{\text{SS}}$  is low the first bit in the SPI Data Register is driven out of the serial data output pin. Also, if the slave is not selected ( $\overline{\text{SS}}$  is high), then the SPSCK input is ignored and no internal shifting of the SPI shift register takes place.

Although the SPI is capable of duplex operation, some SPI peripherals are capable of only receiving SPI data in a slave mode. For these simpler devices, there is no serial data out pin.

#### NOTE

When peripherals with duplex capability are used, take care not to simultaneously enable two receivers whose serial outputs drive the same system slave's serial data output line.

As long as no more than one slave device drives the system slave's serial data output line, it is possible for several slaves to receive the same transmission from a master, although the master would not receive return information from all of the receiving slaves.

If the CPHA bit in SPI Control Register 1 is clear, odd numbered edges on the SPSCK input cause the data at the serial data input pin to be latched. Even numbered edges cause the value previously latched from the serial data input pin to shift into the LSB or MSB of the SPI shift register, depending on the LSBFE bit.

If the CPHA bit is set, even numbered edges on the SPSCK input cause the data at the serial data input pin to be latched. Odd numbered edges cause the value previously latched from the serial data input pin to shift into the LSB or MSB of the SPI shift register, depending on the LSBFE bit.

When CPHA is set, the first edge is used to get the first data bit onto the serial data output pin. When CPHA is clear and the  $\overline{\text{SS}}$  input is low (slave selected), the first bit of the SPI data is driven out of the serial data output pin. After the eighth (SPIMODE = 0) or sixteenth (SPIMODE = 1) shift, the transfer is considered complete and the received data is transferred into the SPI data registers. To indicate transfer is complete, the SPRF flag in the SPI Status Register is set.

**NOTE**

A change of the bits CPOL, CPHA, SSOE, LSBFE, MODFEN, SPC0, and BIDIROE with SPC0 set FIFOMODE and SPIMODE in slave mode corrupts a transmission in progress and has to be avoided.

#### 14.4.4 SPI FIFO MODE

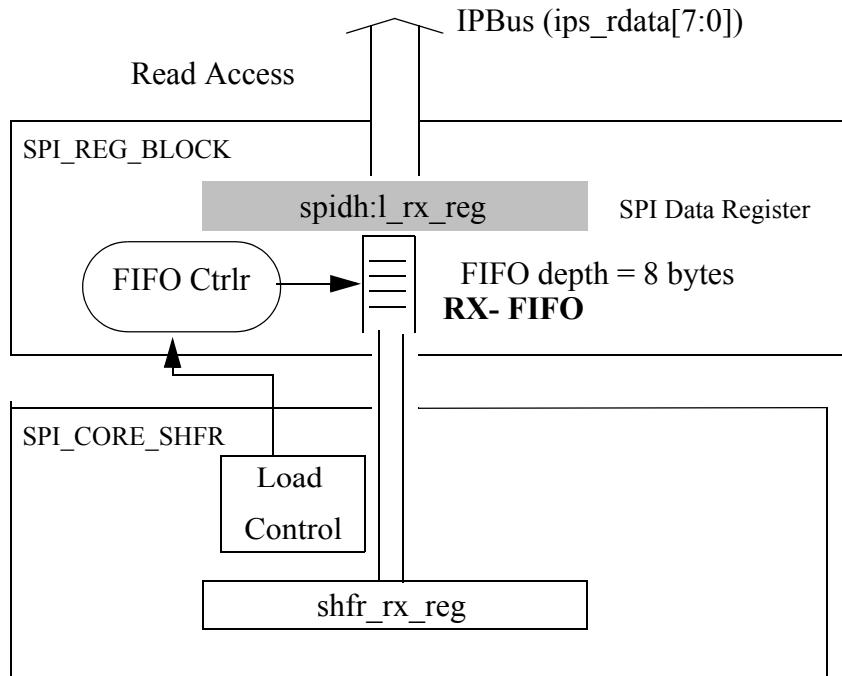


Figure 14-14. SPIDH:L read side structural overview in FIFO mode

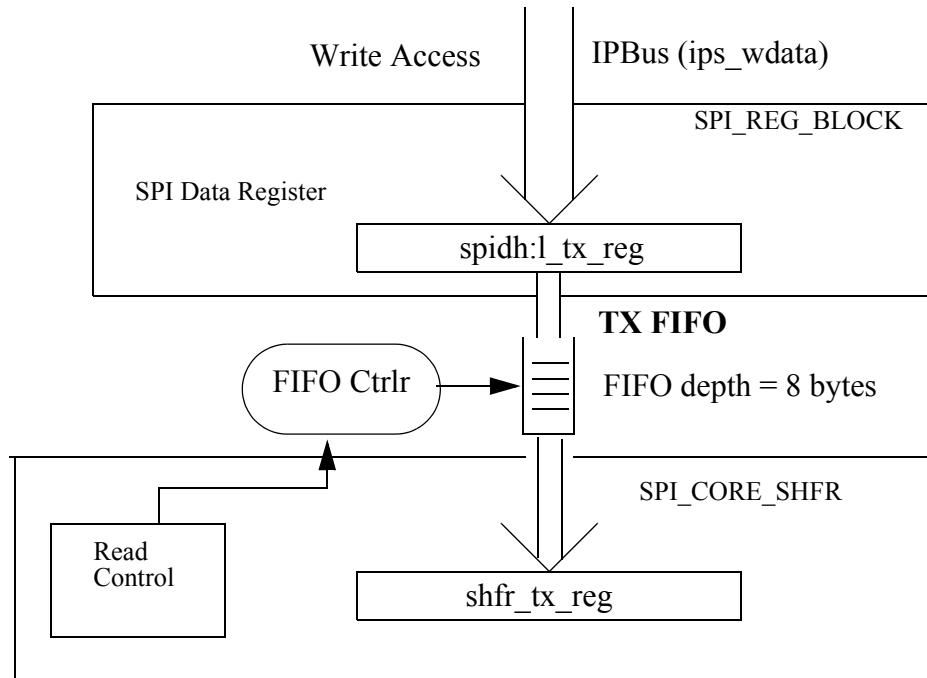


Figure 14-15. SPIDH:L write side structural overview in FIFO mode

SPI works in FIFO mode when SPIxC3[0] bit is set. When in FIFO mode SPI RX buffer and SPI TX buffer is replaced by a 8 byte deep fifo as shown in figure above.

### 14.4.5 Data Transmission Length

The SPI can support data lengths of 8 or 16 bits. The length can be configured with the SPIMODE bit in the SPIxC2 register.

In 8-bit mode (SPIMODE = 0), the SPI Data Register is comprised of one byte: SPIxDL. The SPI Match Register is also comprised of only one byte: SPIxML. Reads of SPIxDH and SPIxMH return zero. Writes to SPIxDH and SPIxMH are ignored.

In 16-bit mode (SPIMODE = 1), the SPI Data Register is comprised of two bytes: SPIxDH and SPIxDL. Reading either byte (SPIxDH or SPIxDL) latches the contents of both bytes into a buffer where they remain latched until the other byte is read. Writing to either byte (SPIxDH or SPIxDL) latches the value into a buffer. When both bytes have been written, they are transferred as a coherent 16-bit value into the transmit data buffer.

In 16-bit mode, the SPI Match Register is also comprised of two bytes: SPIxMH and SPIxML. Reading either byte (SPIxMH or SPIxML) latches the contents of both bytes into a buffer where they remain latched until the other byte is read. Writing to either byte (SPIxMH or SPIxML) latches the value into a buffer. When both bytes have been written, they are transferred as a coherent 16-bit value into the transmit data buffer.

Any switching between 8- and 16-bit data transmission length (controlled by SPIMODE bit) in master mode aborts a transmission in progress, force the SPI system into idle state, and reset all status bits in the SPIxS register. To initiate a transfer after writing to SPIMODE, the SPIxS register must be read with SPTEF = 1, and data must be written to SPIxDH:SPIxDL in 16-bit mode (SPIMODE = 1) or SPIxDL in 8-bit mode (SPIMODE = 0).

In slave mode, user software should write to SPIMODE only once to prevent corrupting a transmission in progress.

#### NOTE

Data can be lost if the data length is not the same for master and slave devices.

### 14.4.6 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 14-16](#) shows the clock formats when SPIMODE = 0 (8-bit mode) and CPHA = 1. 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 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.

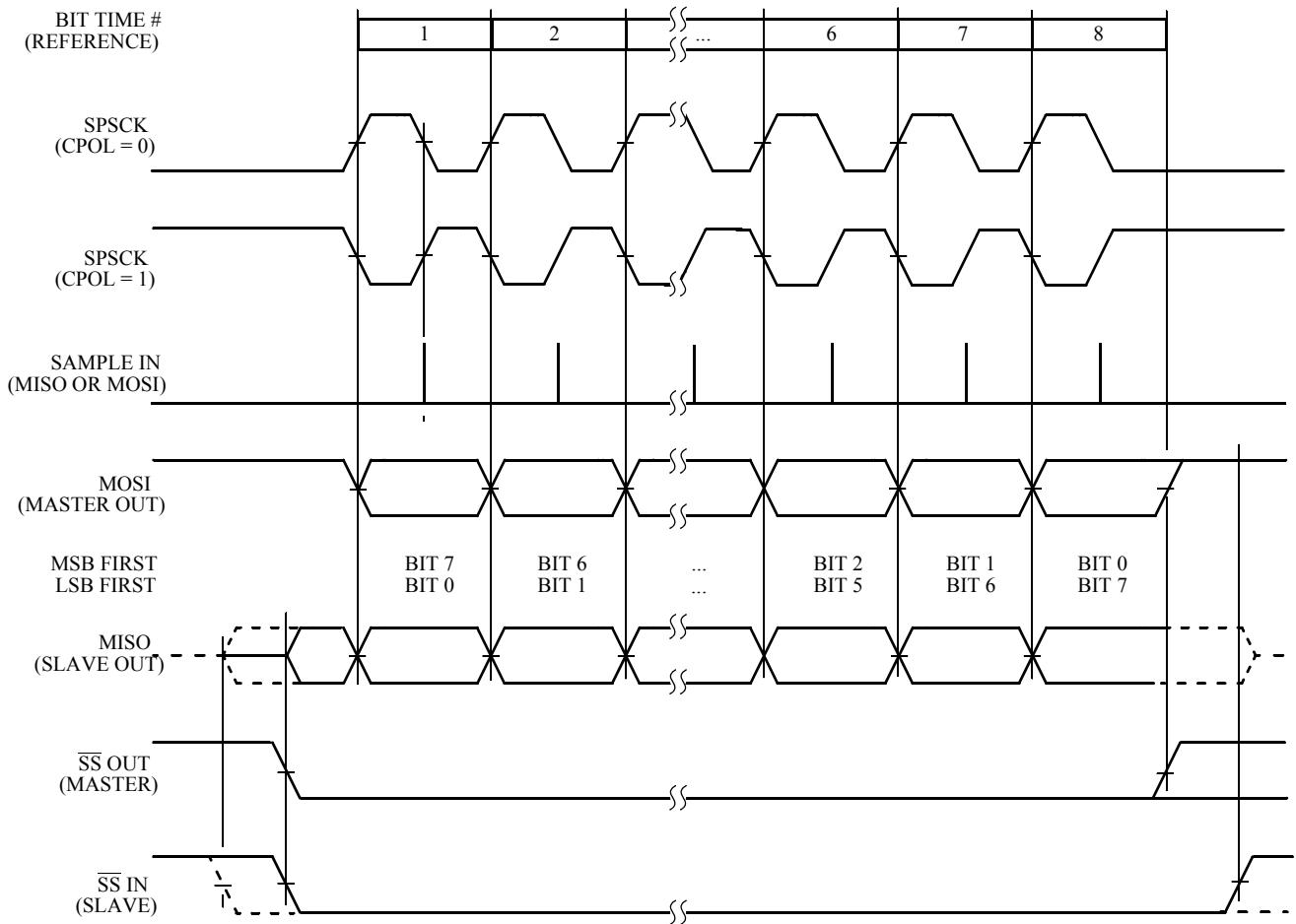


Figure 14-16. SPI Clock Formats (CPHA = 1)

When CPHA = 1, 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 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 that shifts in the bit value that was 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 = 1, the slave's  $\overline{SS}$  input is not required to go to its inactive high level between transfers.

Figure 14-17 shows the clock formats when SPIMODE = 0 and CPHA = 0. 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  $\overline{SS}$  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 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  $\overline{SS}$  IN waveform applies to the slave select input of a slave.

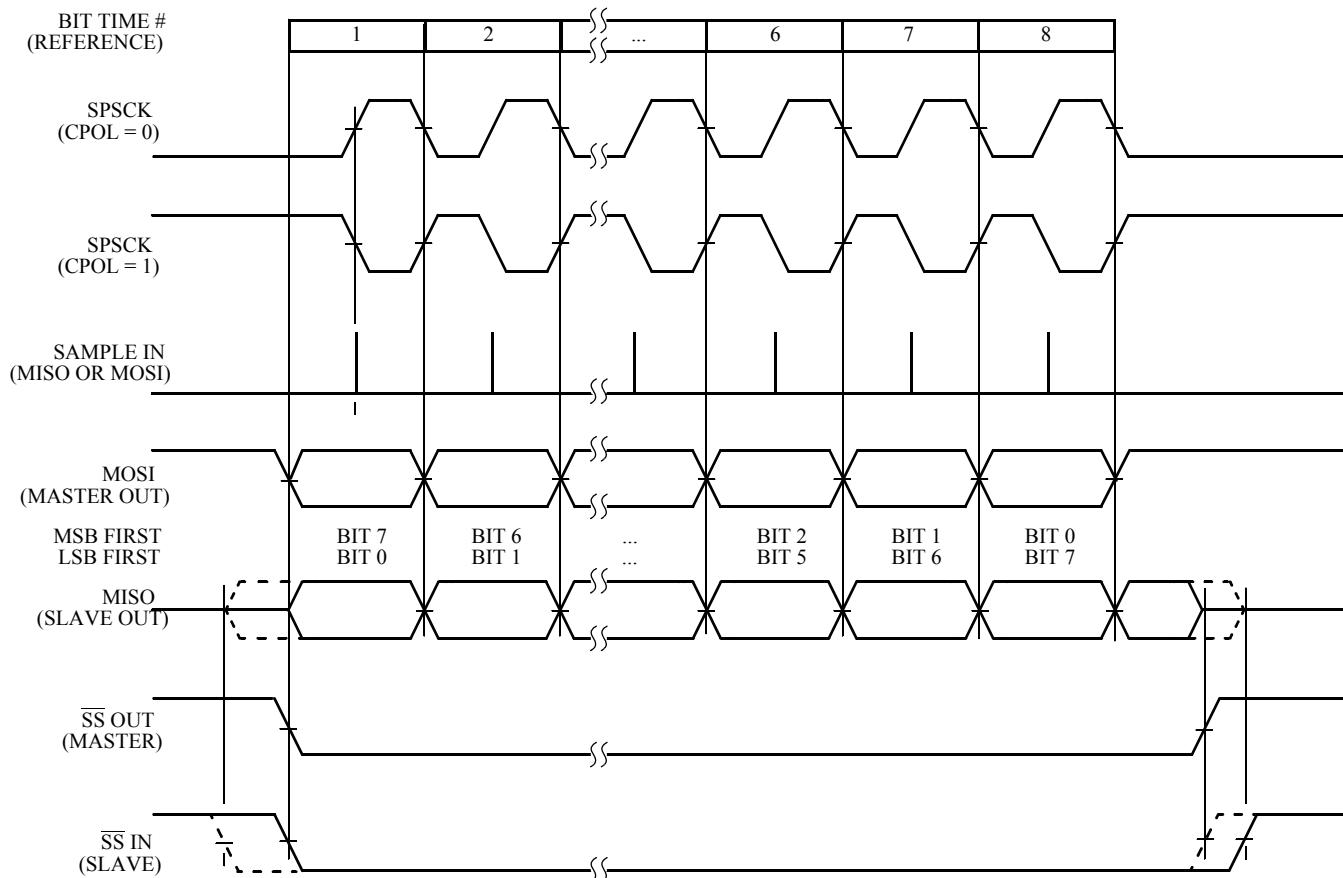


Figure 14-17. SPI Clock Formats (CPHA = 0)

When CPHA = 0, the slave begins to drive its MISO output with the first data bit value (MSB or LSB depending on LSBFE) when  $\overline{SS}$  goes to active low. The first SPSCK edge causes 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 that shifts in the bit value that was 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 = 0, the slave's  $\overline{SS}$  input must go to its inactive high level between transfers.

#### 14.4.7 SPI Baud Rate Generation

As shown in Figure 14-18, 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.

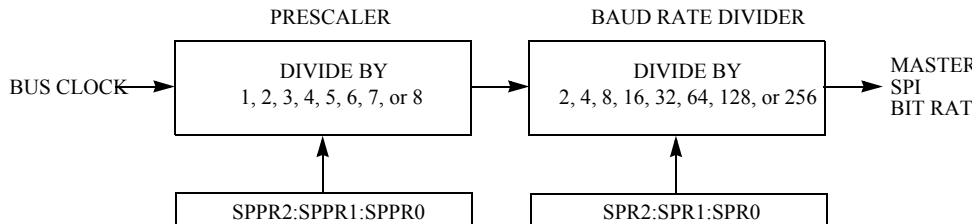
The baud rate generator is activated only when the SPI is in the master mode and a serial transfer is taking place. In the other cases, the divider is disabled to decrease  $I_{DD}$  current.

The baud rate divisor equation is as follows:

$$\text{BaudRateDivisor} = (\text{SPPR} + 1) \bullet 2^{(\text{SPR} + 1)}$$

The baud rate can be calculated with the following equation:

$$\text{Baud Rate} = \text{BusClock} / \text{BaudRateDivisor}$$



**Figure 14-18. SPI Baud Rate Generation**

## 14.4.8 Special Features

### 14.4.8.1 $\overline{\text{SS}}$ Output

The  $\overline{\text{SS}}$  output feature automatically drives the  $\overline{\text{SS}}$  pin low during transmission to select external devices and drives it high during idle to deselect external devices. When  $\overline{\text{SS}}$  output is selected, the  $\overline{\text{SS}}$  output pin is connected to the  $\overline{\text{SS}}$  input pin of the external device.

The  $\overline{\text{SS}}$  output is available only in master mode during normal SPI operation by asserting the SSOE and MODFEN bits as shown in [Table 14-2](#).

The mode fault feature is disabled while  $\overline{\text{SS}}$  output is enabled.

#### NOTE

Care must be taken when using the  $\overline{\text{SS}}$  output feature in a multi-master system because the mode fault feature is not available for detecting system errors between masters.

### 14.4.8.2 Bidirectional Mode (MOMI or SISO)

The bidirectional mode is selected when the SPC0 bit is set in SPI Control Register 2 (see [Table 14-10](#)). In this mode, the SPI uses only one serial data pin for the interface with external device(s). The MSTR bit decides which pin to use. The MOSI pin becomes the serial data I/O (MOMI) pin for the master mode, and the MISO pin becomes serial data I/O (SISO) pin for the slave mode. The MISO pin in master mode and MOSI pin in slave mode are not used by the SPI.

**Table 14-10. Normal Mode and Bidirectional Mode**

<b>When SPE = 1</b>	<b>Master Mode MSTR = 1</b>	<b>Slave Mode MSTR = 0</b>
<b>Normal Mode SPC0 = 0</b>	<pre> graph LR     SPI[SPI] -- "Serial Out" --&gt; MOSI[MOSI]     SPI -- "Serial In" --&gt; MISO[MISO]   </pre>	<pre> graph LR     SPI[SPI] -- "Serial In" --&gt; MOSI[MOSI]     SPI -- "Serial Out" --&gt; MISO[MISO]   </pre>
<b>Bidirectional Mode SPC0 = 1</b>	<pre> graph LR     SPI[SPI] -- "Serial Out" --&gt; BIDIROE[BIDIROE]     BIDIROE --&gt; MOMI[MOMI]     BIDIROE --&gt; SPI   </pre>	<pre> graph LR     SPI[SPI] -- "Serial In" --&gt; BIDIROE[BIDIROE]     BIDIROE --&gt; SISO[SISO]   </pre>

The direction of each serial I/O pin depends on the BIDIROE bit. If the pin is configured as an output, serial data from the shift register is driven out on the pin. The same pin is also the serial input to the shift register.

The SPSCK is output for the master mode and input for the slave mode.

The  $\overline{SS}$  is the input or output for the master mode, and it is always the input for the slave mode.

The bidirectional mode does not affect SPSCK and  $\overline{SS}$  functions.

#### NOTE

In bidirectional master mode, with mode fault enabled, data pins MISO and MOSI can be occupied by the SPI, though MOSI is normally used for transmissions in bidirectional mode and MISO is not used by the SPI. If a mode fault occurs, the SPI is automatically switched to slave mode, in this case MISO becomes occupied by the SPI and MOSI is not used. This has to be considered, if the MISO pin is used for another purpose.

### 14.4.9 Error Conditions

The SPI has one error condition:

- Mode fault error

#### 14.4.9.1 Mode Fault Error

If the  $\overline{SS}$  input becomes low while the SPI is configured as a master, it indicates a system error where more than one master may be trying to drive the MOSI and SPSCK lines simultaneously. This condition is not

permitted in normal operation, and the MODF bit in the SPI status register is set automatically provided the MODFEN bit is set.

In the special case where the SPI is in master mode and MODFEN bit is cleared, the  $\overline{SS}$  pin is not used by the SPI. In this special case, the mode fault error function is inhibited and MODF remains cleared. In case the SPI system is configured as a slave, the  $\overline{SS}$  pin is a dedicated input pin. Mode fault error doesn't occur in slave mode.

If a mode fault error occurs the SPI is switched to slave mode, with the exception that the slave output buffer is disabled. SPSCK, MISO and MOSI pins are forced to be high impedance inputs to avoid any possibility of conflict with another output driver. A transmission in progress is aborted and the SPI is forced into idle state.

If the mode fault error occurs in the bidirectional mode for a SPI system configured in master mode, output enable of the MOMI (MOSI in bidirectional mode) is cleared if it was set. No mode fault error occurs in the bidirectional mode for the SPI system configured in slave mode.

The mode fault flag is cleared automatically by a read of the SPI Status Register (with MODF set) followed by a write to SPI Control Register 1. If the mode fault flag is cleared, the SPI becomes a normal master or slave again.

## 14.4.10 Low Power Mode Options

### 14.4.10.1 SPI in Run Mode

In run mode with the SPI system enable (SPE) bit in the SPI control register clear, the SPI system is in a low-power, disabled state. SPI registers can be accessed, but clocks to the core of this module are disabled.

### 14.4.10.2 SPI in Wait Mode

SPI operation in wait mode depends upon the state of the SPISWAI bit in SPI Control Register 2.

- If SPISWAI is clear, the SPI operates normally when the CPU is in wait mode
- If SPISWAI is set, SPI clock generation ceases and the SPI module enters a power conservation state when the CPU is in wait mode.
  - If SPISWAI is set and the SPI is configured for master, any transmission and reception in progress stops at wait mode entry. The transmission and reception resumes when the SPI exits wait mode.
  - If SPISWAI is set and the SPI is configured as a slave, any transmission and reception in progress continues if the SPSCK continues to be driven from the master. This keeps the slave synchronized to the master and the SPSCK.

If the master transmits data while the slave is in wait mode, the slave continues to send out data consistent with the operation mode at the start of wait mode (i.e., if the slave is currently sending its SPIxDH:SPIxDL to the master, it continues sending the same byte. Otherwise, if the slave is currently sending the last data received byte from the master, it continues sending each previously receive data from the master byte).

**NOTE**

Care must be taken when expecting data from a master while the slave is in wait or stop3 mode. Even though the shift register continues to operate, the rest of the SPI is shut down (i.e. a SPRF interrupt is not generated until exiting stop or wait mode). Also, the data from the shift register is not copied into the SPIxDH:SPIxDL registers until after the slave SPI has exited wait or stop mode. A SPRF flag and SPIxDH:SPIxDL copy is only generated if wait mode is entered or exited during a transmission. If the slave enters wait mode in idle mode and exits wait mode in idle mode, neither a SPRF nor a SPIxDH:SPIxDL copy occurs.

**14.4.10.3 SPI in Stop Mode**

Stop3 mode is dependent on the SPI system. Upon entry to stop3 mode, the SPI module clock is disabled (held high or low). If the SPI is in master mode and exchanging data when the CPU enters stop mode, the transmission is frozen until the CPU exits stop mode. After stop, data to and from the external SPI is exchanged correctly. In slave mode, the SPI stays synchronized with the master.

The stop mode is not dependent on the SPISWAI bit.

In all other stop modes, the SPI module is completely disabled. After stop, all registers are reset to their default values, and the SPI module must be re-initialized.

**14.4.10.4 Reset**

The reset values of registers and signals are described in [Section 14.3, “Register Definition,”](#) that details the registers and their bit-fields.

- If a data transmission occurs in slave mode after reset without a write to SPIxDH:SPIxDL, it transmits garbage or the data last received from the master before the reset.
- Reading from the SPIxDH:SPIxDL after reset always reads zeros.

**14.4.10.5 TNEAREF**

TNEAREF flag is set when only one 16bit word or 2 8bit bytes of data remain in the transmit FIFO provided SPIxC3[5] = 0 or when only two 16bit word or 4 8bit bytes of data remain in the transmit FIFO provided SPIxC3[5] =1. If FIFOMODE is not enabled this bit should be ignored.

Clearing of this interrupt depends on state of SPIxC3[3] and the status of TNEAREF as described [Section 14.3.4, “SPI Status Register \(SPIxS\)](#)

**14.4.10.6 RNFULLF**

RNFULLF is set when more than three 16bit word or six 8bit bytes of data remain in the receive FIFO provided SPIxC3[4] = 0 or when more than two 16bit word or four 8bit bytes of data remain in the receive FIFO provided SPIxC3[4] = 1.

Clearing of this interrupt depends on state of SPIxC3[3] and the status of RNFULLF as described [Section 14.3.4, “SPI Status Register \(SPIxS\).](#)

#### 14.4.10.7 Interrupts

The SPI only originates interrupt requests when the SPI is enabled (SPE bit in SPIxC1 set). The following is a description of how the SPI makes a request and how the MCU should acknowledge that request. The interrupt vector offset and interrupt priority are chip dependent.

### 14.4.11 SPI Interrupts

There are four flag bits, three 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). The SPI match interrupt enable mask bit (SPIMIE) enables interrupts from the SPI match flag (SPMF). 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) should check the flag bits to determine what event caused the interrupt. The service routine should also clear the flag bit(s) before returning from the ISR (usually near the beginning of the ISR).

#### 14.4.11.1 MODF

MODF occurs when the master detects an error on the  $\overline{SS}$  pin. The master SPI must be configured for the MODF feature (see [Table 14-2](#)). After MODF is set, the current transfer is aborted and the following bit is changed:

- MSTR=0, The master bit in SPIxC1 resets.

The MODF interrupt is reflected in the status register MODF flag. Clearing the flag also clears the interrupt. This interrupt stays active while the MODF flag is set. MODF has an automatic clearing process described in [Section 14.3.4, “SPI Status Register \(SPIxS\).](#)”

#### 14.4.11.2 SPRF

SPRF occurs when new data has been received and copied to the SPI receive data buffer. In 8-bit mode, SPRF is set only after all 8 bits have been shifted out of the shift register and into SPIxDL. In 16-bit mode, SPRF is set only after all 16 bits have been shifted out of the shift register and into SPIxDH:SPIxDL.

After SPRF is set, it does not clear until it is serviced. SPRF has an automatic clearing process described in [Section 14.3.4, “SPI Status Register \(SPIxS\).](#)” In the event that the SPRF is not serviced before the end of the next transfer (i.e. SPRF remains active throughout another transfer), the latter transfers are ignored and no new data is copied into the SPIxDH:SPIxDL.

### 14.4.11.3 SPTEF

SPTEF occurs when the SPI transmit buffer is ready to accept new data. In 8-bit mode, SPTEF is set only after all 8 bits have been moved from SPIxDL into the shifter. In 16-bit mode, SPTEF is set only after all 16 bits have been moved from SPIxDH:SPIxDL into the shifter.

After SPTEF is set, it does not clear until it is serviced. SPTEF has an automatic clearing process described in [Section 14.3.4, “SPI Status Register \(SPIxS\)](#).

### 14.4.11.4 SPMF

SPMF occurs when the data in the receive data buffer is equal to the data in the SPI match register. In 8-bit mode, SPMF is set only after bits 8–0 in the receive data buffer are determined to be equivalent to the value in SPIxML. In 16-bit mode, SPMF is set after bits 15–0 in the receive data buffer are determined to be equivalent to the value in SPIxMH:SPIxML.

## 14.5 Initialization/Application Information

### 14.5.1 SPI Module Initialization Example

#### 14.5.1.1 Initialization Sequence

Before the SPI module can be used for communication, an initialization procedure must be carried out, as follows:

1. Update control register 1 (SPIxC1) to enable the SPI and to control interrupt enables. This register also sets the SPI as master or slave, determines clock phase and polarity, and configures the main SPI options.
2. Update control register 2 (SPIxC2) to enable additional SPI functions such as the SPI match interrupt feature, the master mode-fault function, and bidirectional mode output. 8- or 16-bit mode select and other optional features are controlled here as well.
3. Update the baud rate register (SPIxBR) to set the prescaler and bit rate divisor for an SPI master.
4. Update the hardware match register (SPIxMH:SPIxML) with the value to be compared to the receive data register for triggering an interrupt if hardware match interrupts are enabled.
5. In the master, read SPIxS while SPTEF = 1, and then write to the transmit data register (SPIxDH:SPIxDL) to begin transfer.

#### 14.5.1.2 Pseudo—Code Example

In this example, the SPI module is set up for master mode with only hardware match interrupts enabled. The SPI runs in 16-bit mode at a maximum baud rate of bus clock divided by 2. Clock phase and polarity are set for an active-high SPI clock where the first edge on SPSCK occurs at the start of the first cycle of a data transfer.

**SPIxC1=0x54(%01010100)**

Bit 7	SPIE	= 0	Disables receive and mode fault interrupts
Bit 6	SPE	= 1	Enables the SPI system
Bit 5	SPTIE	= 0	Disables SPI transmit interrupts
Bit 4	MSTR	= 1	Sets the SPI module as a master SPI device
Bit 3	CPOL	= 0	Configures SPI clock as active-high
Bit 2	CPHA	= 1	First edge on SPSCK at start of first data transfer cycle
Bit 1	SSOE	= 0	Determines $\overline{SS}$ pin function when mode fault enabled
Bit 0	LSBFE	= 0	SPI serial data transfers start with most significant bit

**SPIxC2 = 0xC0(%11000000)**

Bit 7	SPMIE	= 1	SPI hardware match interrupt enabled
Bit 6	SPIMODE	= 1	Configures SPI for 16-bit mode
Bit 5		= 0	Unimplemented
Bit 4	MODFEN	= 0	Disables mode fault function
Bit 3	BIDIROE	= 0	SPI data I/O pin acts as input
Bit 2		= 0	Unimplemented
Bit 1	SPISWAI	= 0	SPI clocks operate in wait mode
Bit 0	SPC0	= 0	uses separate pins for data input and output

**SPIxBR = 0x00(%00000000)**

Bit 7		= 0	Unimplemented
Bit 6:4		= 000	Sets prescale divisor to 1
Bit 3		= 0	Unimplemented
Bit 2:0		= 000	Sets baud rate divisor to 2

**SPIxS = 0x00(%00000000)**

Bit 7	SPRF	= 0	Flag is set when receive data buffer is full
Bit 6	SPMF	= 0	Flag is set when SPIMH/L = receive data buffer
Bit 5	SPTEF	= 0	Flag is set when transmit data buffer is empty
Bit 4	MODF	= 0	Mode fault flag for master mode
Bit 3:0		= 0	Unimplemented

**SPIxMH = 0xXX**

In 16-bit mode, this register holds bits 8–15 of the hardware match buffer. In 8-bit mode, writes to this register are ignored.

**SPIxML = 0xXX**

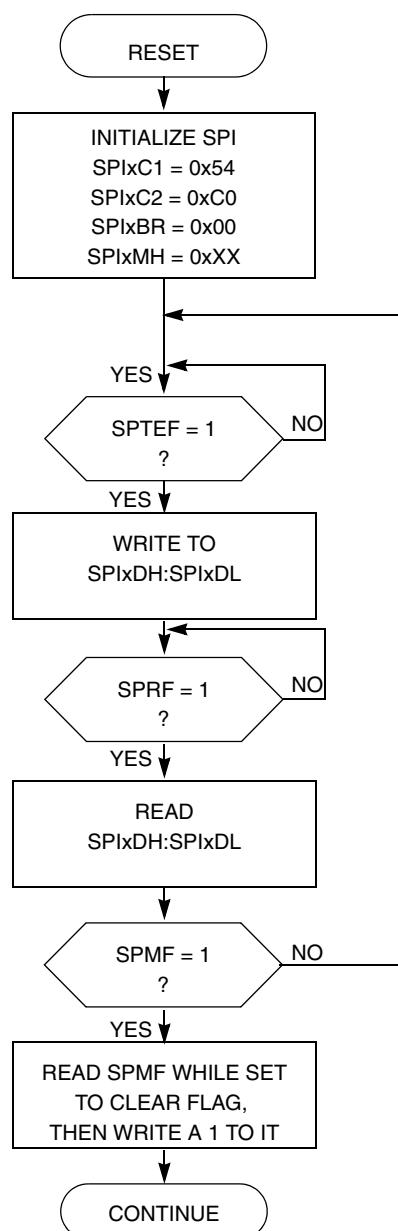
Holds bits 0–7 of the hardware match buffer.

**SPIxDH = 0xxx**

In 16-bit mode, this register holds bits 8–15 of the data to be transmitted by the transmit buffer and received by the receive buffer.

**SPIxDL = 0xxx**

Holds bits 0–7 of the data to be transmitted by the transmit buffer and received by the receive buffer.



**Figure 14-19. Initialization Flowchart Example for SPI Master Device in 16-bit Mode for FIFOMODE = 0**

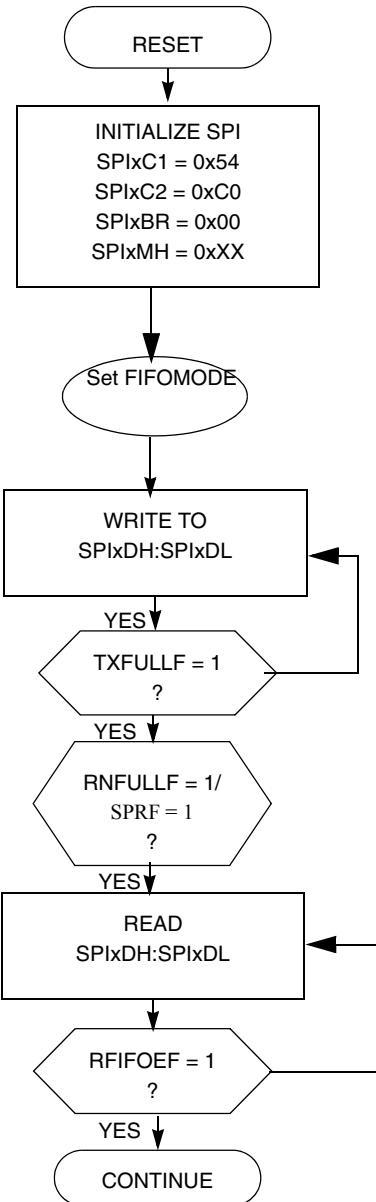


Figure 14-20. Initialization Flowchart Example for SPI Master Device in 16-bit Mode for FIFOMODE



# Chapter 15

## Carrier Modulator Timer (CMT)

### 15.1 Features

The CMT consists of a carrier generator, modulator, and transmitter that drives the infrared out (IRO) pin. The features of this module include:

- Four modes of operation
  - Time with independent control of high and low times
  - Baseband
  - Frequency shift key (FSK)
  - Direct software control of IRO pin
- Extended space operation in time, baseband, and FSK modes
- Selectable input clock divide: 1, 2, 4, or 8
- Interrupt on end of cycle
  - Ability to disable IRO pin and use as timer interrupt

### 15.2 CMT Block Diagram

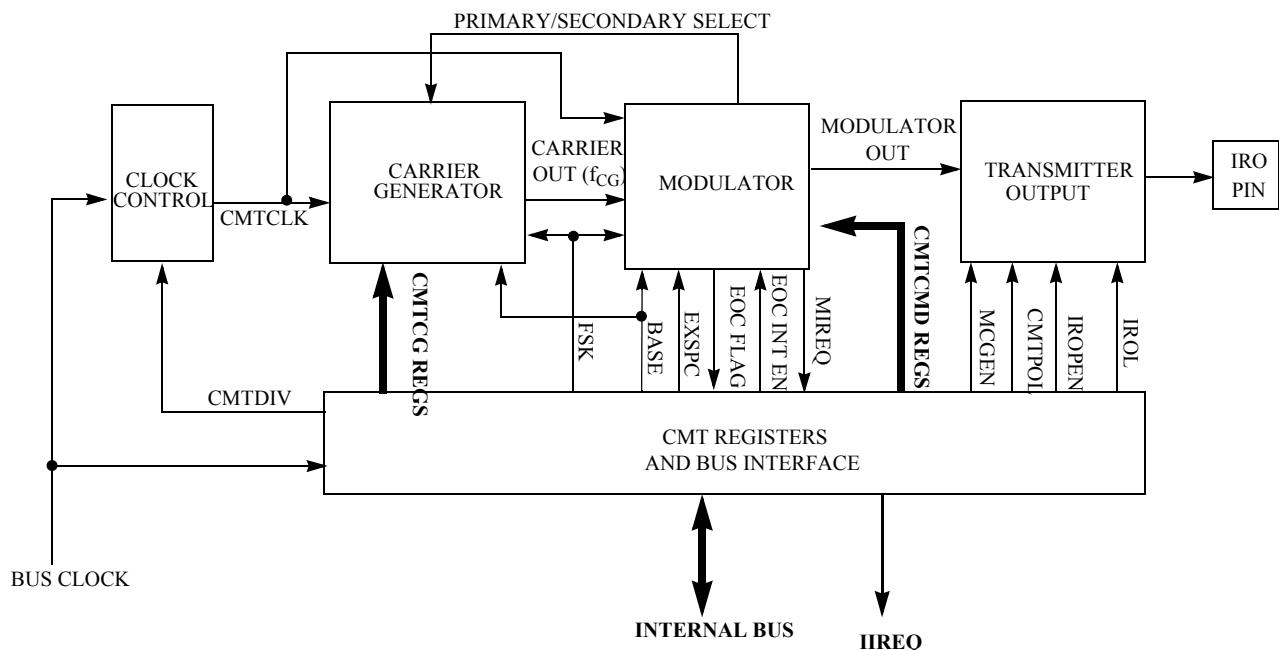


Figure 15-1. Carrier Modulator Transmitter Module Block Diagram

## 15.3 Pin Description

The IRO pin is the only pin associated with the CMT. The pin is driven by the transmitter output when the MCGEN bit in the CMTMSC register and the IROPEN bit in the CMTOC register are set. If the MCGEN bit is clear and the IROPEN bit is set, the pin is driven by the IROL bit in the CMTOC register. This enables user software to directly control the state of the IRO pin by writing to the IROL bit. If the IROPEN bit is clear, the pin is disabled and is not driven by the CMT module. This is so the CMT can be configured as a modulo timer for generating periodic interrupts without causing pin activity.

## 15.4 CMT Pad Configuration

The CMT pads of the PTC2 pin of GPIO are double bonded to enhance the current requirement for the external current-gain amplifier. To support the 20mA specification on this pin, use the following GPIO configuration for PTC2:

- Configure PTC2 with the highest drive option for CMT operation.
- Drive strength enabled (DSE=1) and slew rate disabled (SRE=0).

## 15.5 Functional Description

The CMT module consists of a carrier generator, a modulator, a transmitter output, and control registers. The block diagram is shown in [Figure 15-1](#). When operating in time mode, the user independently defines the high and low times of the carrier signal to determine period and duty cycle. The carrier generator resolution is 125 ns when operating with an 8 MHz internal bus frequency and the CMTDIV1 and CMTDIV0 bits in the CMTMSC register are both equal to 0. The carrier generator can generate signals with periods between 250 ns (4 MHz) and 127.5  $\mu$ s (7.84 kHz) in steps of 125 ns. See [Table 15-1](#).

**Table 15-1. Clock Divide**

Bus Clock (MHz)	CMTDIV1:CMTDIV 0	Carrier Generator Resolution ( $\mu$ s)	Min Carrier Generator Period ( $\mu$ s)	Min Modulator Period ( $\mu$ s)
8	0:0	0.125	0.25	1.0
8	0:1	0.25	0.5	2.0
8	1:0	0.5	1.0	4.0
8	1:1	1.0	2.0	8.0

The possible duty cycle options depend upon the number of counts required to complete the carrier period. For example, a 1.6 MHz signal has a period of 625 ns and requires  $5 \times 125$  ns counts to generate. These counts may be split between high and low times, so the duty cycles available are 20 percent (one high, four low), 40 percent (two high, three low), 60 percent (three high, two low) and 80 percent (four high, one low).

For lower frequency signals with larger periods, higher resolution (as a percentage of the total period) duty cycles are possible.

When the BASE bit in the CMT modulator status and control register (CMTMSC) is set, the carrier output ( $f_{CG}$ ) to the modulator is held high continuously to allow for the generation of baseband protocols.

A third mode allows the carrier generator to alternate between two sets of high and low times. When operating in FSK mode, the generator toggles between the two sets when instructed by the modulator, allowing the user to dynamically switch between two carrier frequencies without CPU intervention.

The modulator provides a simple method to control protocol timing. The modulator has a minimum resolution of 1.0  $\mu$ s with an 8 MHz internal bus clock. It can count bus clocks (to provide real-time control) or it can count carrier clocks (for self-clocked protocols). See [Section 15.5.2, “Modulator,”](#) for more details.

The transmitter output block controls the state of the infrared out pin (IRO). The modulator output is gated on to the IRO pin when the modulator/carrier generator is enabled.

A summary of the possible modes is shown in [Table 15-2](#).

**Table 15-2. CMT Modes of Operation**

Mode	MCGEN Bit <sup>1</sup>	BASE Bit <sup>2</sup>	FSK Bit <sup>(2)</sup>	EXSPC Bit	Comment
Time	1	0	0	0	$f_{CG}$ controlled by primary high and low registers. $f_{CG}$ transmitted to IRO pin when modulator gate is open.
Baseband	1	1	x	0	$f_{CG}$ is always high. IRO pin high when modulator gate is open.
FSK	1	0	1	0	$f_{CG}$ control alternates between primary high/low registers and secondary high/low registers. $f_{CG}$ transmitted to IRO pin when modulator gate is open.
Extended Space	1	x	x	1	Setting the EXSPC bit causes subsequent modulator cycles to be spaces (modulator out not asserted) for the duration of the modulator period (mark and space times).
IRO Latch	0	x	x	x	IROL bit controls state of IRO pin.

<sup>1</sup> To prevent spurious operation, initialize all data and control registers before beginning a transmission (MCGEN=1).

<sup>2</sup> These bits are not double buffered and should not be changed during a transmission (while MCGEN=1).

### 15.5.1 Carrier Generator

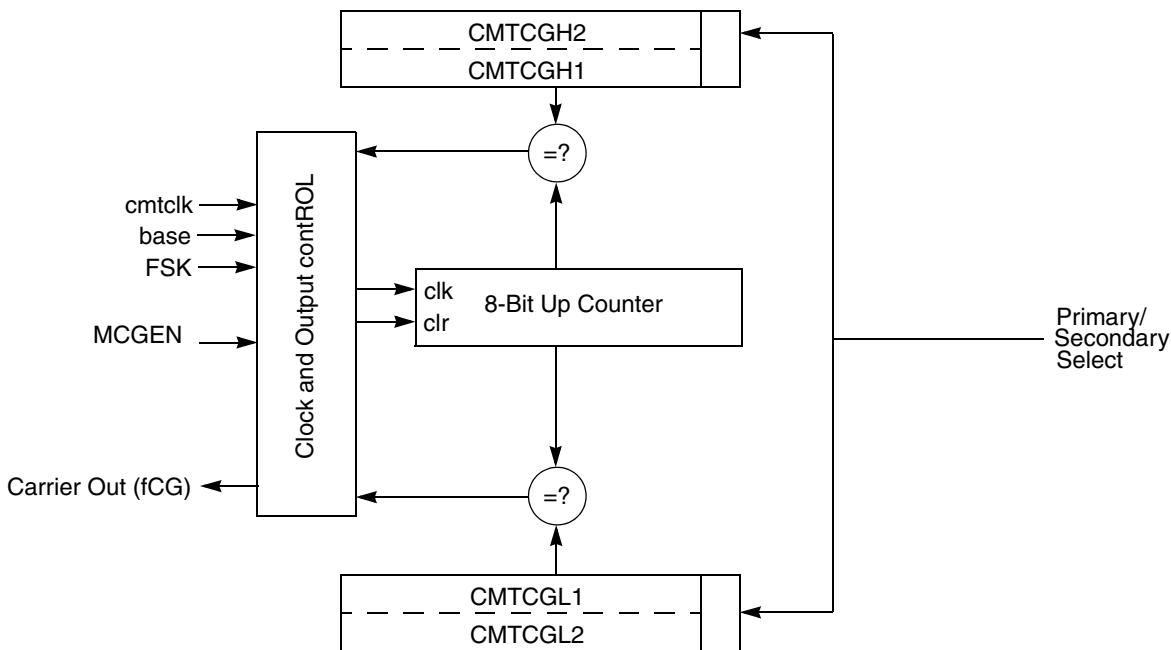
The carrier signal is generated by counting a register-selected number of input clocks (125 ns for an 8 MHz bus) for the carrier high time and the carrier low time. The period is determined by the total number of clocks counted. The duty cycle is determined by the ratio of high time clocks to total clocks counted. The high and low time values are user programmable and are held in two registers.

An alternate set of high/low count values is held in another set of registers to allow the generation of dual frequency FSK (frequency shift keying) protocols without CPU intervention.

**NOTE**

Only non-zero data values are allowed. The carrier generator does not work if any of the count values are equal to zero.

The MCGEN bit in the CMTMSC register must be set and the BASE bit must be cleared to enable carrier generator clocks. When the BASE bit is set, the carrier output to the modulator is held high continuously. The block diagram is shown in [Figure 15-2](#).



**Figure 15-2. Carrier Generator Block Diagram**

The high/low time counter is an 8-bit up counter. After each increment, the contents of the counter are compared with the appropriate high or low count value register. When the compare value is reached, the counter is reset to a value of 0x01, and the compare is redirected to the other count value register.

Assuming that the high time count compare register is currently active, a valid compare causes the carrier output to be driven low. The counter continues to increment (starting at reset value of 0x01). When the value stored in the selected low count value register is reached, the counter is reset again and the carrier output is driven high.

The cycle repeats, automatically generating a periodic signal directed to the modulator. The lowest frequency (maximum period) and highest frequency (minimum period) that can be generated are defined as:

$$f_{\max} = f_{\text{CMTCCLK}} \div (2 \times 1) \text{ Hz} \quad \text{Eqn. 15-1}$$

$$f_{\min} = f_{\text{CMTCCLK}} \div (2 \times (2^8 - 1)) \text{ Hz} \quad \text{Eqn. 15-2}$$

In the general case, the carrier generator output frequency is:

$$f_{\text{CG}} = f_{\text{CMTCCLK}} \div (\text{Highcount} + \text{Lowcount}) \text{ Hz} \quad \text{Eqn. 15-3}$$

Where:  $0 < \text{Highcount} < 256$  and  
 $0 < \text{Lowcount} < 256$

The duty cycle of the carrier signal is controlled by varying the ratio of high time to low + high time. As the input clock period is fixed, the duty cycle resolution is proportional to the number of counts required to generate the desired carrier period.

**Eqn. 15-4**

$$\text{Duty Cycle} = \frac{\text{Highcount}}{\text{Highcount} + \text{Lowcount}}$$

### 15.5.2 Modulator

The modulator has three main modes of operation:

- Gate the carrier onto the modulator output (time mode)
- Control the logic level of the modulator output (baseband mode)
- Count carrier periods and instruct the carrier generator to alternate between two carrier frequencies when a modulation period (mark + space counts) expires (FSK mode)

The modulator includes a 17-bit down counter with underflow detection. The counter is loaded from the 16-bit modulation mark period buffer registers, CMTCMD1 and CMTCMD2. The most significant bit is loaded with a logic zero and serves as a sign bit. When the counter holds a positive value, the modulator gate is open and the carrier signal is driven to the transmitter block.

When the counter underflows, the modulator gate is closed and a 16-bit comparator is enabled that compares the logical complement of the value of the down-counter with the contents of the modulation space period register (which has been loaded from the registers CMTCMD3 and CMTCMD4).

When a match is obtained the cycle repeats by opening the modulator gate, reloading the counter with the contents of CMTCMD1 and CMTCMD2, and reloading the modulation space period register with the contents of CMTCMD3 and CMTCMD4.

If the contents of the modulation space period register are all zeroes, the match is immediate and no space period is generated (for instance, for FSK protocols that require successive bursts of different frequencies).

The MCGEN bit in the CMTMSC register must be set to enable the modulator timer.

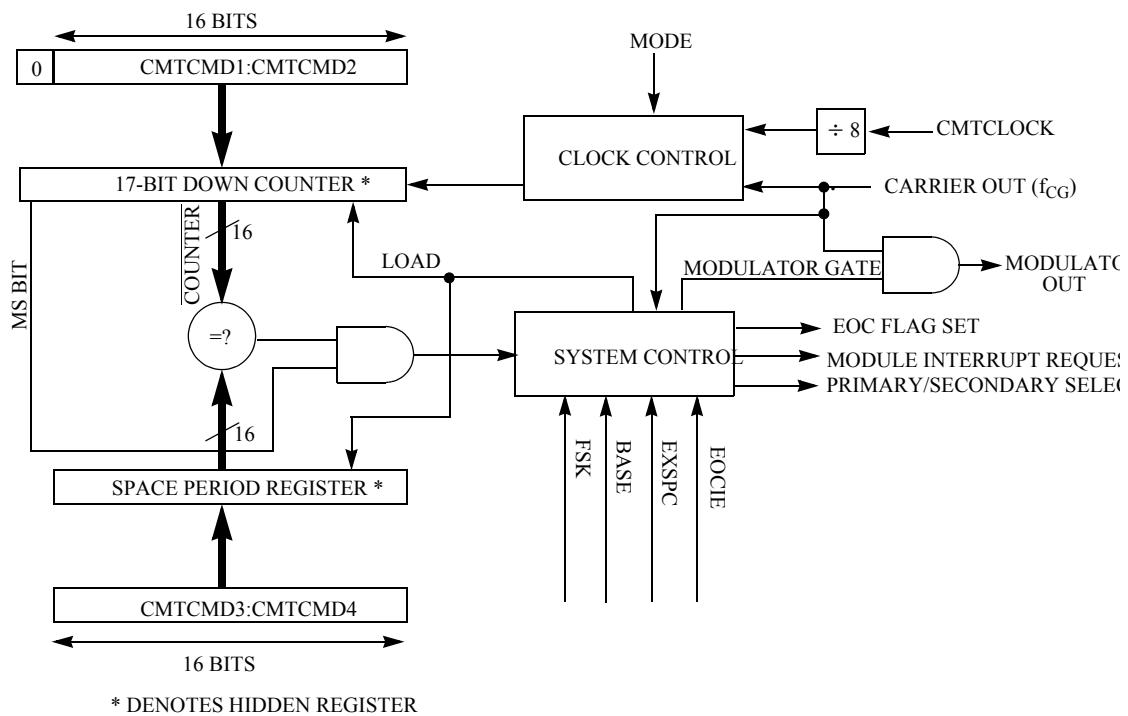


Figure 15-3. Modulator Block Diagram

### 15.5.2.1 Time Mode

When the modulator operates in time mode (MCGEN bit is set, BASE bit is clear, and FSK bit is clear), the modulation mark period consists of an integer number of  $CMTCLK \div 8$  clock periods. The modulation space period consists of zero or an integer number of  $CMTCLK \div 8$  clock periods. With an 8 MHz bus and  $CMTDIV1:CMTDIV0 = 00$ , the modulator resolution is 1  $\mu$ s and has a maximum mark and space period of about 65.535 ms each. See [Figure 15-4](#) for an example of the time mode and baseband mode outputs.

The mark and space time equations for time and baseband mode are:

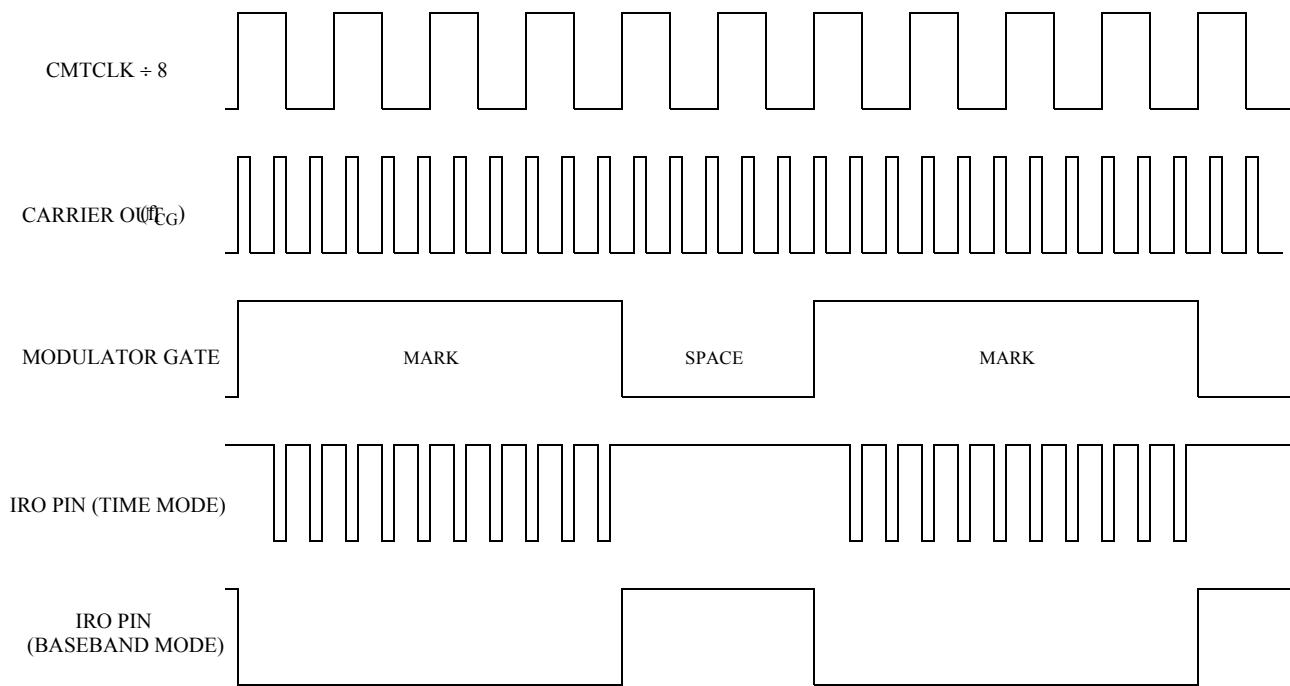
$$t_{mark} = (CMTCMD1:CMTCMD2 + 1) \div (f_{CMTCLK} \div 8) \quad Eqn. 15-5$$

$$t_{space} = CMTCMD3:CMTCMD4 \div (f_{CMTCLK} \div 8) \quad Eqn. 15-6$$

where CMTCMD1:CMTCMD2 and CMTCMD3:CMTCMD4 are the decimal values of the concatenated registers.

#### NOTE

If the modulator is disabled while the  $t_{mark}$  time is less than the programmed carrier high time ( $t_{mark} < CMTCGH1/f_{CMTCLK}$ ), the modulator can enter into an illegal state and end the current cycle before the programmed value. Make sure to program  $t_{mark}$  greater than the carrier high time to avoid this illegal state.



**Figure 15-4. Example CMT Output in Time and Baseband Modes**

### 15.5.2.2 Baseband Mode

Baseband mode (MCGEN bit is set and BASE bit is set) is a derivative of time mode, where the mark and space period is based on ( $\text{CMTCLK} \div 8$ ) counts. The mark and space calculations are the same as in time mode. In this mode the modulator output is at a logic 1 for the duration of the mark period and at a logic 0 for the duration of a space period. See [Figure 15-4](#) for an example of the output for baseband and time modes. In the example, the carrier out frequency ( $f_{CG}$ ) is generated with a high count of 0x01 and a low count of 0x02, which results in a divide of 3 of CMTCLK with a 33 percent duty cycle. The modulator down-counter was loaded with the value 0x0003 and the space period register with 0x0002.

#### NOTE

The waveforms in [Figure 15-4](#) and [Figure 15-5](#) are for the purpose of conceptual illustration and are not meant to represent precise timing relationships between the signals shown.

### 15.5.2.3 FSK Mode

When the modulator operates in FSK mode (MCGEN bit is set, FSK bit is set, and BASE bit is clear), the modulation mark and space periods consist of an integer number of carrier clocks (space period can be 0). When the mark period expires, the space period is transparently started (as in time mode). The carrier generator toggles between primary and secondary data register values when the modulator space period expires.

## Carrier Modulator Timer (CMT)

The space period provides an interpulse gap (no carrier). If CMTCMD3:CMTCMD4 = 0x0000, then the modulator and carrier generator switches between carrier frequencies without a gap or any carrier glitches (zero space).

Using timing data for carrier burst and interpulse gap length calculated by the CPU, FSK mode can automatically generate a phase-coherent, dual-frequency FSK signal with programmable burst and interburst gaps.

The mark and space time equations for FSK mode are:

$$t_{\text{mark}} = (\text{CMTCMD1:CMTCMD2} + 1) \div f_{\text{CG}} \quad \text{Eqn. 15-7}$$

$$t_{\text{space}} = \text{CMTCMD3:CMTCMD4} \div f_{\text{CG}} \quad \text{Eqn. 15-8}$$

Where  $f_{\text{CG}}$  is the frequency output from the carrier generator. The example in [Figure 15-5](#) shows what the IRO pin output looks like in FSK mode with the following values: CMTCMD1:CMTCMD2 = 0x0003, CMTCMD3:CMTCMD4 = 0x0002, primary carrier high count = 0x01, primary carrier low count = 0x02, secondary carrier high count = 0x03, and secondary carrier low count = 0x01.

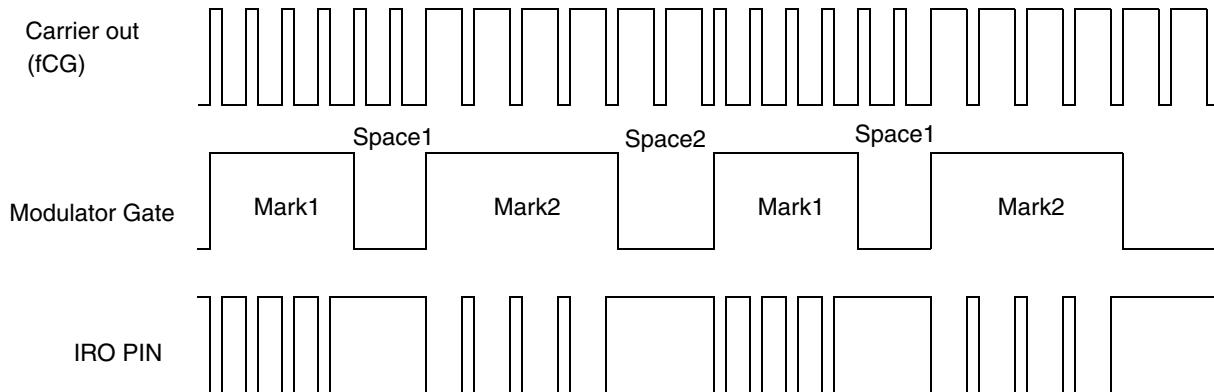


Figure 15-5. Example CMT Output in FSK Mode

### 15.5.3 Extended Space Operation

In time, baseband, or FSK mode, the space period can be made longer than the maximum possible value of the space period register. Setting the EXSPC bit in the CMTMSC register forces the modulator to treat the next modulation period (beginning with the next load of the counter and space period register) as a space period equal in length to the mark and space counts combined. Subsequent modulation periods consist entirely of these extended space periods with no mark periods. Clearing EXSPC returns the modulator to standard operation at the beginning of the next modulation period.

#### 15.5.3.1 EXSPC Operation in Time Mode

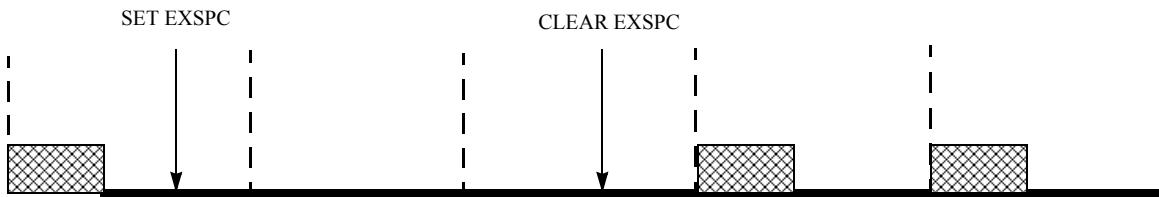
To calculate the length of an extended space in time or baseband modes, add the mark and space times and multiply by the number of modulation periods that EXSPC is set.

$$t_{\text{exspace}} = t_{\text{space}} + (t_{\text{mark}} + t_{\text{space}}) \times (\text{number of modulation periods}) \quad \text{Eqn. 15-9}$$

For an example of extended space operation, see [Figure 15-6](#).

#### NOTE

The EXSPC feature can be used to emulate a zero mark event.



**Figure 15-6. Extended Space Operation**

#### 15.5.3.2 EXSPC Operation in FSK Mode

In FSK mode, the modulator continues to count carrier out clocks, alternating between the primary and secondary registers at the end of each modulation period.

To calculate the length of an extended space in FSK mode, the user must know whether the EXSPC bit was set on a primary or secondary modulation period, as well as the total number of primary and secondary modulation periods completed while the EXSPC bit is high. A status bit for the current modulation is not accessible to the CPU. If necessary, software should maintain tracking of the current modulation cycle (primary or secondary). The extended space period ends at the completion of the space period time of the modulation period during which the EXSPC bit is cleared.

If the EXSPC bit was set during a primary modulation cycle, use the equation:

$$t_{exspace} = (t_{space})_p + (t_{mark} + t_{space})_s + (t_{mark} + t_{space})_p + \dots \quad \text{Eqn. 15-10}$$

Where the subscripts p and s refer to mark and space times for the primary and secondary modulation cycles.

If the EXSPC bit was set during a secondary modulation cycle, use the equation:

$$t_{exspace} = (t_{space})_s + (t_{mark} + t_{space})_p + (t_{mark} + t_{space})_s + \dots \quad \text{Eqn. 15-11}$$

#### 15.5.4 Transmitter

The transmitter output block controls the state of the infrared out pin (IRO). The modulator output is gated on to the IRO pin when the modulator/carrier generator is enabled. When the modulator/carrier generator is disabled, the IRO pin is controlled by the state of the IRO latch.

A polarity bit in the CMTOC register enables the IRO pin to be high true or low true.

#### 15.5.5 CMT Interrupts

The end of cycle flag (EOCF) is set when:

- The modulator is not currently active and the MCGEN bit is set to begin the initial CMT transmission

- At the end of each modulation cycle (when the counter is reloaded from CMTCMD1:CMTCMD2) while the MCGEN bit is set

In the case where the MCGEN bit is cleared and then set before the end of the modulation cycle, the EOCF bit is not set when the MCGEN is set, but becomes set at the end of the current modulation cycle.

When the MCGEN becomes disabled, the CMT module does not set the EOC flag at the end of the last modulation cycle.

The EOCF bit is cleared by reading the CMT modulator status and control register (CMTMSC) followed by an access of CMTCMD2 or CMTCMD4.

If the EOC interrupt enable (EOCIE) bit is high when the EOCF bit is set, the CMT module generates an interrupt request. The EOCF bit must be cleared within the interrupt service routine to prevent another interrupt from being generated after exiting the interrupt service routine.

The EOC interrupt is coincident with loading the down-counter with the contents of CMTCMD1:CMTCMD2 and loading the space period register with the contents of CMTCMD3:CMTCMD4. The EOC interrupt provides a means for the user to reload new mark/space values into the modulator data registers. Modulator data register updates takes effect at the end of the current modulation cycle. The down-counter and space period register are updated at the end of every modulation cycle, regardless of interrupt handling and the state of the EOCF flag.

### **15.5.6 Wait Mode Operation**

During wait mode, the CMT, if enabled, continues to operate normally. However, there are no new codes or changes of pattern mode while in wait mode, because the CPU is not operating.

### **15.5.7 Stop Mode Operation**

During all stop modes, clocks to the CMT module are halted.

In stop2 modes, all CMT register data is lost and must be re-initialized upon recovery from these two stop modes.

No CMT module registers are affected in stop3/4 mode.

Because the clocks are halted, the CMT resumes upon exit from stop (only in stop3/4 mode). Software should ensure stop2 or stop3/4 mode is not entered while the modulator is in operation to prevent the IRO pin from being asserted while in stop mode. This may require a time-out period from the time that the MCGEN bit is cleared to allow the last modulator cycle to complete.

### **15.5.8 Background Mode Operation**

When the microcontroller is in active background mode, the CMT temporarily suspends all counting until the microcontroller returns to normal user mode.

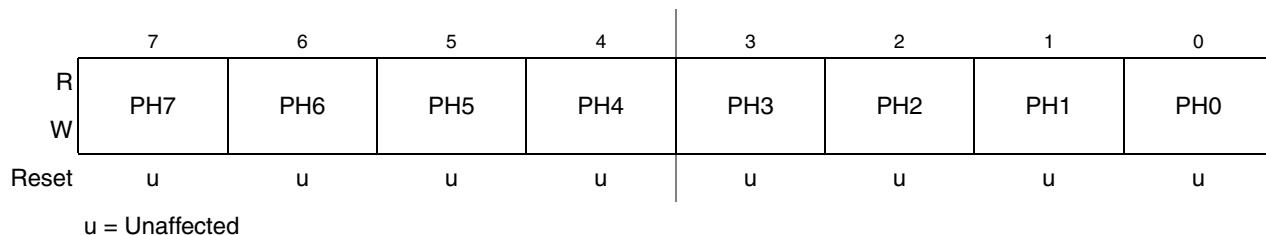
## 15.6 CMT Registers and Control Bits

The following registers control and monitor CMT operation:

- CMT carrier generator data registers (CMTCGH1, CMTCGL1, CMTCGH2, CMTCGL2)
- CMT output control register (CMTOC)
- CMT modulator status and control register (CMTMSC)
- CMT modulator period data registers (CMTCMD1, CMTCMD2, CMTCMD3, CMTCMD4)

### 15.6.1 Carrier Generator Data Registers (CMTCGH1, CMTCGL1, CMTCGH2, and CMTCGL2)

The carrier generator data registers contain the primary and secondary high and low values for generating the carrier output.



**Figure 15-7. Carrier Generator Data Register High 1(CMTCGH1)**

**Table 15-3. CMTCGH1 Field Descriptions**

Field	Description
7:0 PH[7:0]	<b>Primary Carrier High Time Data Values</b> — When selected, these bits contain the number of input clocks required to generate the carrier high and low time periods. When operating in time mode (see <a href="#">Section 15.5.2.1, “Time Mode”</a> ), this register pair is always selected. When operating in FSK mode (see <a href="#">Section 15.5.2.3, “FSK Mode”</a> ), this register pair and the secondary register pair are alternatively selected under control of the modulator. The primary carrier high and low time values are unaffected out of reset. These bits must be written to nonzero values before the carrier generator is enabled to avoid spurious results.

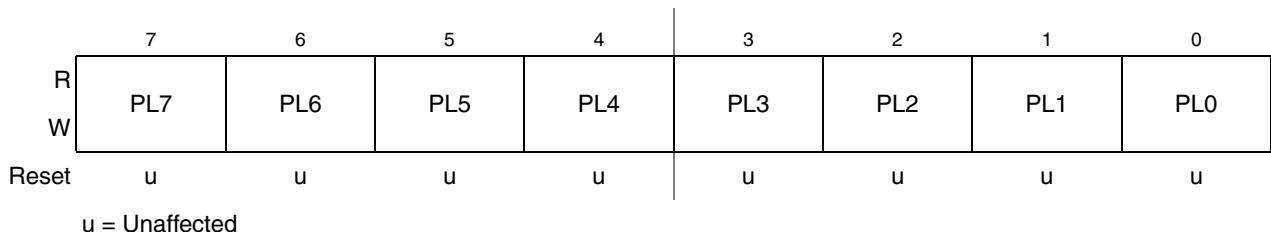


Figure 15-8. Carrier Generator Data Register Low 1 (CMTCGL1)

Table 15-4. CMTCGL1 Field Descriptions

Field	Description
7:0 PL[7:0]	<b>Primary Carrier Low Time Data Values</b> — When selected, these bits contain the number of input clocks required to generate the carrier high and low time periods. When operating in time mode (see <a href="#">Section 15.5.2.1, "Time Mode"</a> ), this register pair is always selected. When operating in FSK mode (see <a href="#">Section 15.5.2.3, "FSK Mode"</a> ), this register pair and the secondary register pair are alternatively selected under control of the modulator. The primary carrier high and low time values are unaffected out of reset. These bits must be written to nonzero values before the carrier generator is enabled to avoid spurious results.

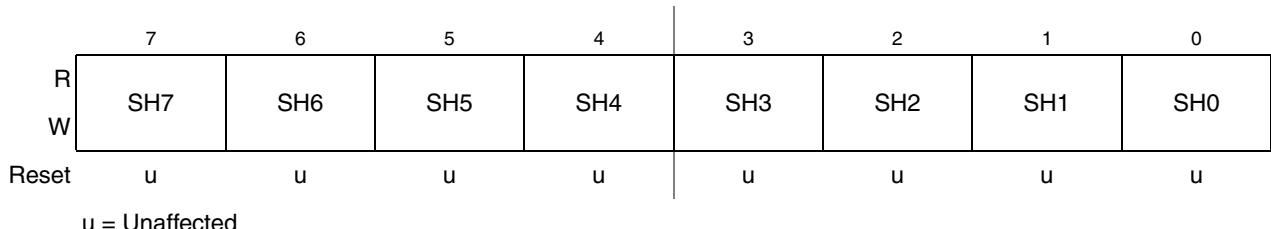


Figure 15-9. Carrier Generator Data Register High 2 (CMTCGH2)

Table 15-5. CMTCGH2 Field Descriptions

Field	Description
7:0 SH[7:0]	<b>Secondary Carrier High Time Data Values</b> — When selected, these bits contain the number of input clocks required to generate the carrier high and low time periods. When operating in time mode (see <a href="#">Section 15.5.2.1, "Time Mode"</a> ), this register pair is never selected. When operating in FSK mode (see <a href="#">Section 15.5.2.3, "FSK Mode"</a> ), this register pair and the primary register pair are alternatively selected under control of the modulator. The secondary carrier high and low time values are unaffected out of reset. These bits must be written to nonzero values before the carrier generator is enabled when operating in FSK mode.

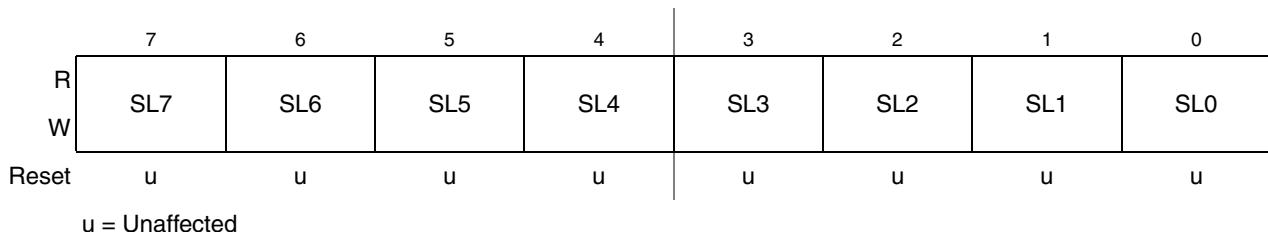


Figure 15-10. Carrier Generator Data Register Low 2 (CMTCGL2)

Table 15-6. CMTCGL2 Field Descriptions

Field	Description
7:0 SL[7:0]	<b>Secondary Carrier Low Time Data Values</b> — When selected, these bits contain the number of input clocks required to generate the carrier high and low time periods. When operating in time mode (see <a href="#">Section 15.5.2.1, "Time Mode"</a> ), this register pair is never selected. When operating in FSK mode (see <a href="#">Section 15.5.2.3, "FSK Mode"</a> ), this register pair and the primary register pair are alternatively selected under control of the modulator. The secondary carrier high and low time values are unaffected out of reset. These bits must be written to nonzero values before the carrier generator is enabled when operating in FSK mode.

## 15.6.2 CMT Output Control Register (CMTOC)

This register is used to control the IRO output of the CMT module.



Figure 15-11. CMT Output Control Register (CMTOC)

Table 15-7. CMTOC Field Descriptions

Field	Description
7 IROL	<b>IRO Latch Control</b> — Reading IROL reads the state of the IRO latch. Writing IROL changes the state of the IRO pin when the MCGEN bit is clear in the CMTMSC register and the IROOPEN bit is set.
6 CMTPOL	<b>CMT Output Polarity</b> — The CMTPOL bit controls the polarity of the IRO pin output of the CMT. 0 IRO pin is active low 1 IRO pin is active high
5 IROOPEN	<b>IRO Pin Enable</b> — The IROOPEN bit is used to enable and disable the IRO pin. When the pin is enabled, it is an output that drives out the CMT transmitter output or the state of the IROL bit depending on whether the MCGEN bit in the CMTMSC register is set. Also, the state of the output is inverted or not depending on the state of the CMTPOL bit. When the pin is disabled, it is in a high impedance state so it doesn't draw any current. The pin is disabled during reset. 0 IRO pin disabled 1 IRO pin enabled as output

### 15.6.3 CMT Modulator Status and Control Register (CMTMSC)

The CMT modulator status and control register (CMTMSC) contains the modulator and carrier generator enable (MCGEN), end of cycle interrupt enable (EOCIE), FSK mode select (FSK), baseband enable (BASE), extended space (EXSPC), prescaler (CMTDIV1:CMTDIV0) bits, and the end of cycle (EOCF) status bit.

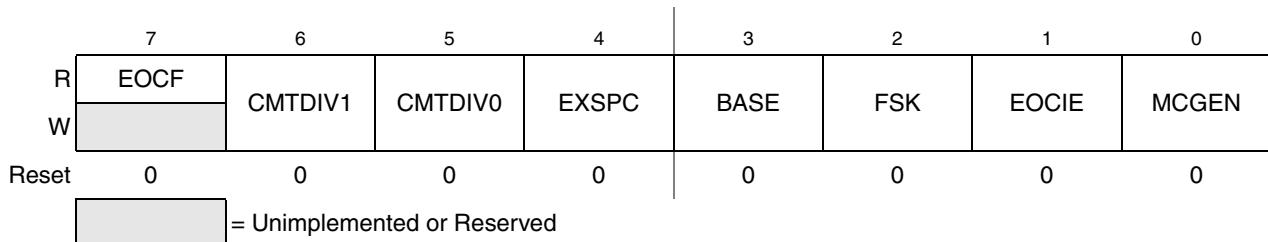


Figure 15-12. CMT Modulator Status and Control Register (CMTMSC)

Table 15-8. CMTMSC Field Descriptions

Field	Description
7 EOCF	<b>End of Cycle Status Flag</b> — The EOCF bit is set when: <ul style="list-style-type: none"> <li>The modulator is not currently active and the MCGEN bit is set to begin the initial CMT transmission.</li> <li>At the end of each modulation cycle while the MCGEN bit is set. This is recognized when a match occurs between the contents of the space period register and the down-counter. At this time, the counter is initialized with the (possibly new) contents of the mark period buffer, CMTCMD1 and CMTCMD2. The space period register is loaded with the (possibly new) contents of the space period buffer, CMTCMD3 and CMTCMD4.</li> </ul> This flag is cleared by a read of the CMTMSC register followed by an access of CMTCMD2 or CMTCMD4. In the case where the MCGEN bit is cleared and then set before the end of the modulation cycle, EOCF is not set when MCGEN is set, but is set at the end of the current modulation cycle. 0 No end of modulation cycle occurrence since flag last cleared 1 End of modulator cycle has occurred
6:5 CMTDIV[1:0]	<b>CMT Clock Divide Prescaler</b> — The CMT clock divide prescaler causes the CMT to be clocked at the BUS CLOCK frequency, or the BUS CLOCK frequency divided by 1, 2, 4, or 8. Because these bits are not double buffered, they should not be changed during a transmission. 00 Bus clock $\div 1$ 01 Bus clock $\div 2$ 10 Bus clock $\div 4$ 11 Bus clock $\div 8$
4 EXSPC	<b>Extended Space Enable</b> — The EXSPC bit enables extended space operation. 0 Extended space disabled 1 Extended space enabled
3 BASE	<b>Baseband Enable</b> — When set, the BASE bit disables the carrier generator and forces the carrier output high for generation of baseband protocols. When BASE is clear, the carrier generator is enabled and the carrier output toggles at the frequency determined by values stored in the carrier data registers. See <a href="#">Section 15.5.2.2, "Baseband Mode."</a> This bit is cleared by reset. This bit is not double buffered and should not be written to during a transmission. 0 Baseband mode disabled 1 Baseband mode enabled
2 FSK	<b>FSK Mode Select</b> — The FSK bit enables FSK operation. 0 CMT operates in time or baseband mode 1 CMT operates in FSK mode

**Table 15-8. CMTMSC Field Descriptions (continued)**

Field	Description
1 EOCIE	<b>End of Cycle Interrupt Enable</b> — A CPU interrupt is requested when EOCF is set if EOCIE is high. 0 CPU interrupt disabled 1 CPU interrupt enabled
0 MCGEN	<b>Modulator and Carrier Generator Enable</b> — Setting MCGEN initializes the carrier generator and modulator and enable all clocks. After it is enabled, the carrier generator and modulator function continuously. When MCGEN is cleared, the current modulator cycle is allowed to expire before all carrier and modulator clocks are disabled (to save power) and the modulator output is forced low. To prevent spurious operation, the user should initialize all data and control registers before enabling the system. 0 Modulator and carrier generator disabled 1 Modulator and carrier generator enabled

### 15.6.4 CMT Modulator Data Registers (CMTCMD1, CMTCMD2, CMTCMD3, and CMTCMD4)

The modulator data registers control the mark and space periods of the modulator for all modes. The contents of these registers are transferred to the modulator down counter and space period register upon the completion of a modulation period.

**Table 15-9. Sample Register Summary**

Name		7	6	5	4	3	2	1	0
CMTCMD1	R	MB15	MB14	MB13	MB12	MB11	MB10	MB9	MB8
	W								
CMTCMD2	R	MB7	MB6	MB5	MB4	MB3	MB2	MB1	MB0
	W								
CMTCMD3	R	SB15	SB14	SB13	SB12	SB11	SB10	SB9	SB8
	W								
CMTCMD4	R	SB7	SB6	SB5	SB4	SB3	SB2	SB1	SB0
	W								



# **Chapter 16**

## **Universal Serial Bus, OTG Capable Controller**

### **NOTE**

Portions of [Chapter 16, “Universal Serial Bus, OTG Capable Controller,”](#) relating to the EHCI specification are Copyright © Intel Corporation 1999-2001. The EHCI specification is provided as is with no warranties whatsoever, including any warranty of merchantability, non-infringement, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification or sample. Intel disclaims all liability, including liability for infringement of any proprietary rights, relating to use of information in the EHCI specification. Intel may make changes to the EHCI specifications at any time, without notice.

This chapter describes the USB On The Go dual role Controller (USB-FS), which implements many industry standards. However, it is beyond the scope of this document to document the intricacies of these standards. Instead, it is left to the reader to refer to the governing specifications.

The following documents are available from the USB Implementers Forum web page at <http://www.usb.org/developers/docs>:

- *Universal Serial Bus Specification, Revision 1.1*
- *On-The-Go Supplement to the USB 2.0 Specification, Revision 1.0a*

### **16.1 Introduction**

This section describes the USB On The Go dual role controller (USB-FS). The OTG implementation in this module provides limited host functionality as well as full-speed device solutions for implementing a USB 2.0 full-speed/low-speed compliant peripheral. The OTG implementation supports the On-The-Go (OTG) addendum to the USB 2.0 Specification. Only one protocol can be active at any time. A negotiation protocol must be used to switch to a USB host functionality from a USB device. This is known as the Master Negotiation Protocol (MNP).

#### **16.1.1 USB**

The USB is a cable bus that supports data exchange between a host computer and a wide range of simultaneously accessible peripherals. The attached peripherals share USB bandwidth through a host-scheduled, token-based protocol. The bus allows peripherals to be attached, configured, used, and detached while the host and other peripherals are in operation.

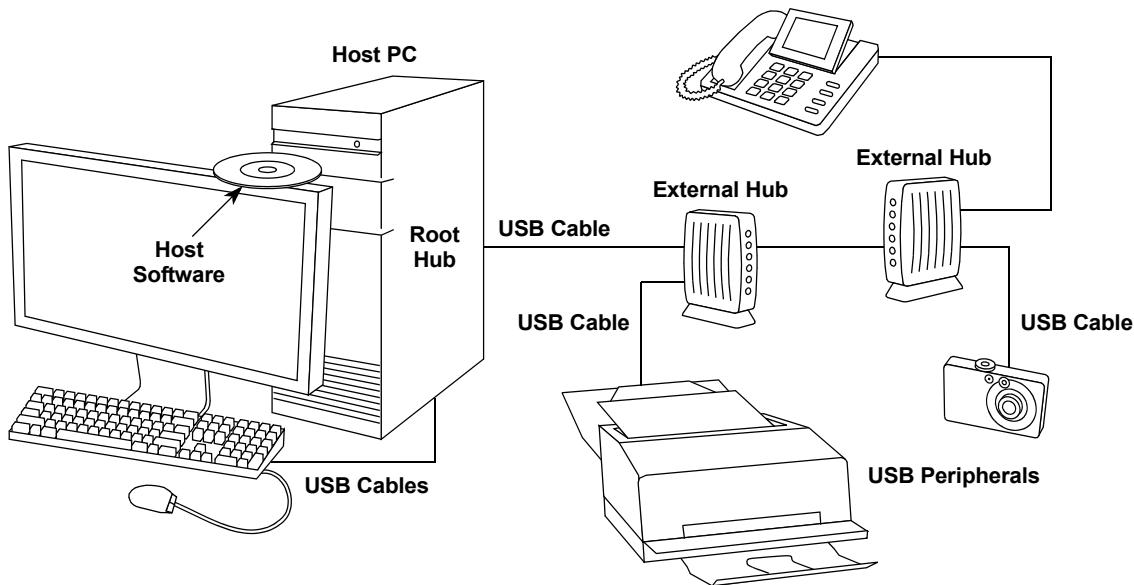
USB software provides a uniform view of the system for all application software, hiding implementation details making application software more portable. It manages the dynamic attach and detach of peripherals.

There is only one host in any USB system. The USB interface to the host computer system is referred to as the Host Controller.

There may be multiple USB devices in any system such as joysticks, speakers, printers, etc. USB devices present a standard USB interface in terms of comprehension, response, and standard capability.

The host initiates transactions to specific peripherals, while the device responds to control transactions. The device sends and receives data to and from the host using a standard USB data format. USB 2.0 full-speed /low-speed peripherals operate at 12Mb/s or 1.5 MB/s.

For additional information, refer to the USB2.0 specification [2].



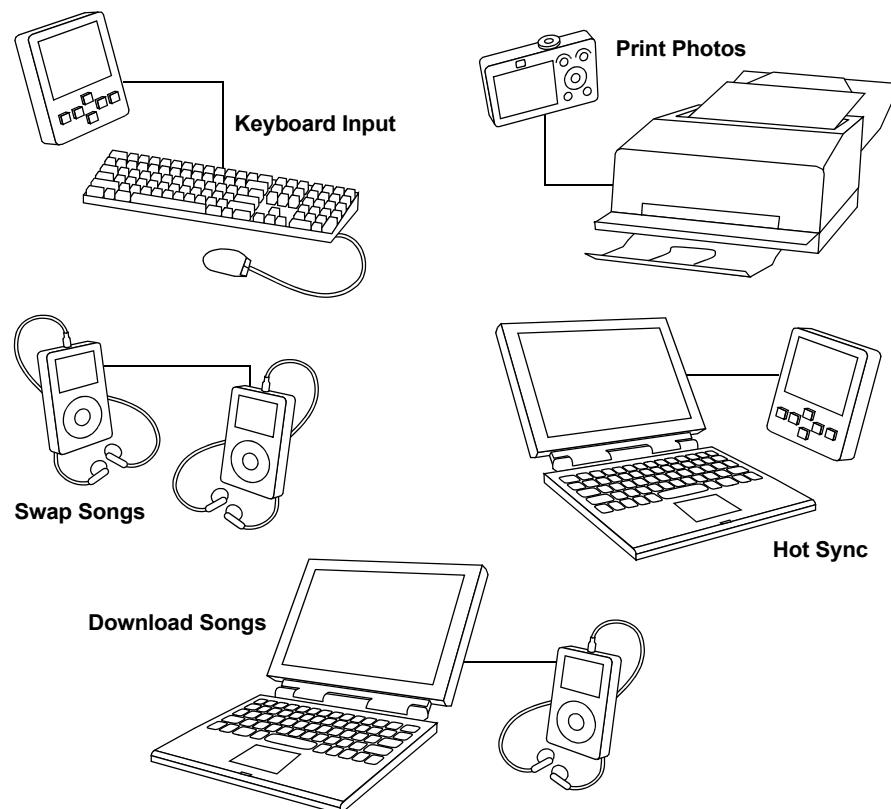
**Figure 16-1. Example USB 2.0 System Configuration**

### 16.1.2 USB On-The-Go

USB (Universal Serial Bus) is a popular standard for connecting peripherals and portable consumer electronic devices such as digital cameras and hand-held computers to host PCs. The On-The-Go (OTG) Supplement to the USB Specification extends USB to peer-to-peer application. Using USB OTG technology consumer electronics, peripherals and portable devices can connect to each other (for example, a digital camera can connect directly to a printer, or a keyboard can connect to a Personal Digital Assistant) to exchange data.

With the USB On-The-Go product, you can develop a fully USB-compliant peripheral device that can also assume the role of a USB host. Software determines the role of the device based on hardware signals, and then initializes the device in the appropriate mode of operation (host or peripheral) based on how it is connected. After connecting the devices can negotiate using the OTG protocols to assume the role of host or peripheral based on the task to be accomplished.

For additional information, refer to the *On-The-Go Supplement to the USB 2.0 Specification* [3].



**Figure 16-2. Example USB 2.0 On-The-Go Configurations**

### 16.1.3 USB-FS Features

- USB 1.1 and 2.0 compliant full-speed device controller
- 16-Bidirectional end points
- DMA or FIFO data stream interfaces
- Low-power consumption
- On-The-Go protocol logic

### 16.1.4 Modes of Operation

For details on low-power mode operation, refer to [Table 3-2](#).

USB functions in three modes: run, wait, and stop.

#### 16.1.4.1 Run Mode

This is the basic mode of operation.

#### 16.1.4.2 Wait Mode

The system wakes up from wait mode when USB asserts an interrupt.

#### 16.1.4.3 Stop Mode

When USB detects that there is no activity on the USB bus for more than 3 ms, the sleep bit in the INT\_STAT register gets set. This bit can cause an interrupt. After clearing the interrupt, you can choose whether to have the device go into STOP4/STOP3 mode.

Before having the device go into the STOP3/STOP4 mode, the following programming is recommended:

1. Program the USBRESMEN bit in the USBTRC0 register to 1 to wake up the system through an asynchronous interrupt because of activity on the USB bus.
2. Program the SUSP bit in the USB\_CTRL register so that the receive and transmit control signals are disabled on the transceiver for low-power consumption.

When the Sync Interrupt corresponding to this Async interrupt occurs (bit 0 of the USBTRC0 register), you must clear the SUSP bit in the USB\_CTRL register, so that the state machine in the USB controller can detect RESUME signaling on USB.

## 16.2 External Signal Description

### 16.2.1 USB Pull-up/Pull-down Connections

[Figure 16-3](#) depicts the USB Pull-up/Pull-down control signals and the DPlus and DMinus lines.

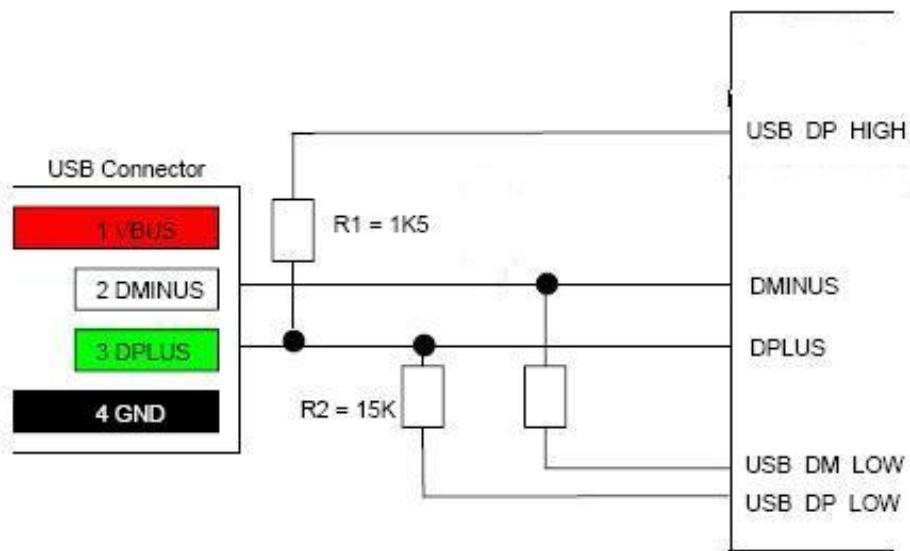
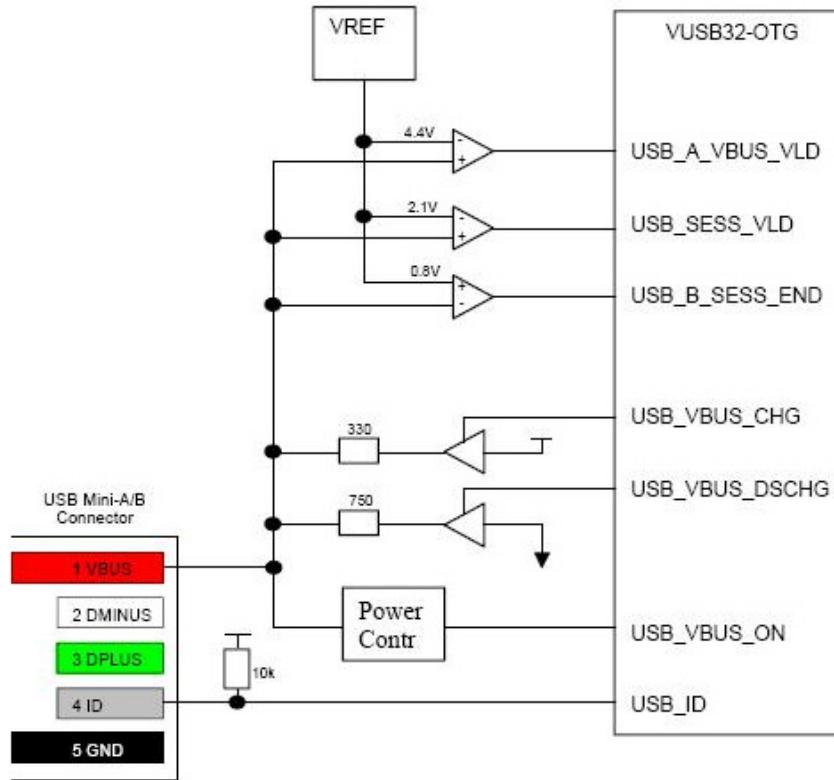


Figure 16-3. Pull-up/Pull-down Connections for USB

### 16.2.2 USB OTG Connections

Figure 16-4 depicts the USB OTG pin connections.



**Figure 16-4. OTG connections for USB**

As shown in [Figure 16-4](#), **USB\_VBUS\_CHG**, **USB\_VBUS\_DSCHG**, and **USB\_VBUS\_ON** are not controlled through any of the register bits within the USB controller. You should use the GPIOs to control these outputs.

## 16.3 Functional Description

The USB-FS 2.0 full-speed/low-speed module communicates with the ColdFire processor core through status and control registers, and data structures in memory.

### 16.3.1 Data Structures

The function of the device operation is to transfer a request in the memory image to and from the Universal Serial Bus. To efficiently manage USB endpoint communications the USB-FS implements a Buffer Descriptor Table (BDT) in system memory. See [Figure 16-5](#).

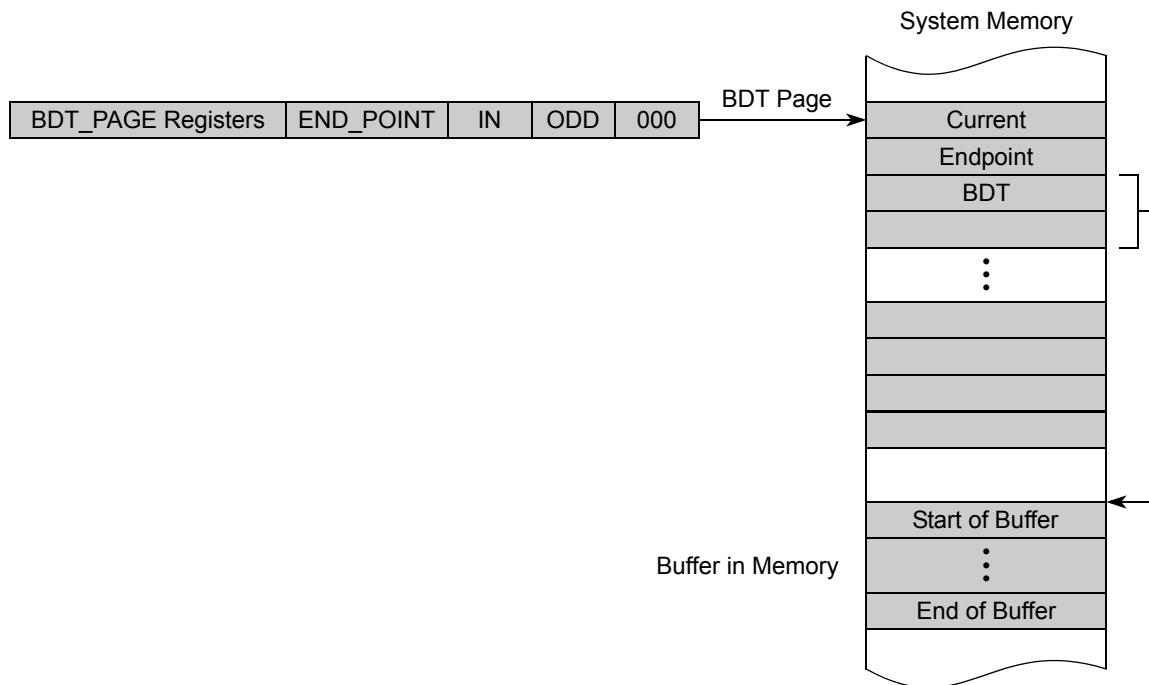
## 16.4 Programmers Interface

### 16.4.1 Buffer Descriptor Table

To efficiently manage USB endpoint communications the USB-FS implements a Buffer Descriptor Table (BDT) in system memory. The BDT resides on a 512 byte boundary in system memory and is pointed to

by the BDT Page Registers. Every endpoint direction requires two eight-byte Buffer Descriptor entries. Therefore, a system with 16 fully bidirectional endpoints would require 512 bytes of system memory to implement the BDT. The two Buffer Descriptor (BD) entries allows for an EVEN BD and ODD BD entry for each endpoint direction. This allows the microprocessor to process one BD while the USB-FS is processing the other BD. Double buffering BDs in this way allows the USB-FS to easily transfer data at the maximum throughput provided by USB.

The software API intelligently manages buffers for the USB-FS by updating the BDT when needed. This allows the USB-FS to efficiently manage data transmission and reception, while the microprocessor performs communication overhead processing and other function dependent applications. Because the buffers are shared between the microprocessor and the USB-FS a simple semaphore mechanism is used to distinguish who is allowed to update the BDT and buffers in system memory. A semaphore bit, the OWN bit, is cleared to 0 when the BD entry is owned by the microprocessor. The microprocessor is allowed read and write access to the BD entry and the buffer in system memory when the OWN bit is 0. When the OWN bit is set to 1, the BD entry and the buffer in system memory are owned by the USB-FS. The USB-FS now has full read and write access and the microprocessor should not modify the BD or its corresponding data buffer. The BD also contains indirect address pointers to where the actual buffer resides in system memory. This indirect address mechanism is shown in the following diagram.



**Figure 16-5. Buffer Descriptor Table**

#### 16.4.2 Rx vs. Tx as a USB Target Device or USB Host

The USB-FS core can function as a USB target device, or as a USB hosts, and may switch modes of operation between host and target device under software control. In either mode, USB host or USB target device, the same data paths and buffer descriptors are used for the transmission and reception of data. For this reason, a USB-FS core centric nomenclature is used to describe the direction of the data transfer

between the USB-FS core and the USB. Rx or receive is used to describe transfers that move data from the USB to memory, and Tx, or transmit is used to describe transfers that move data from memory to the USB. The following table shows how the data direction corresponds to the USB token type in host and target device applications.

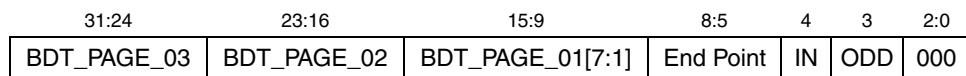
**Table 16-1. Data Direction for USB Host or USB Target**

	Rx	Tx
Device	OUT or Setup	IN
Host	IN	Out or Setup

### 16.4.3 Addressing Buffer Descriptor Table Entries

To access endpoint data via the USB-FS or microprocessor, the addressing mechanism of the Buffer Descriptor Table must be understood. As stated earlier, the Buffer Descriptor Table occupies up to 512 bytes of system memory. Sixteen bidirectional endpoints can be supported with a full BDT of 512 bytes. Sixteen bytes are needed for each USB endpoint direction. Applications with less than 16 End Points require less RAM to implement the BDT. The BDT Page Registers point to the starting location of the BDT. The BDT must be located on a 512-byte boundary in system memory. All enabled TX and RX endpoint BD entries are indexed into the BDT to allow easy access via the USB-FS or ColdFire Core.

When the USB-FS receives a USB token on an enabled endpoint it uses its integrated DMA controller to interrogate the BDT. The USB-FS must read the corresponding endpoint BD entry and determine if it owns the BD and corresponding buffer in system memory. To compute the entry point in to the BDT, the BDT\_PAGE registers is concatenated with the current endpoint and the TX and ODD fields to form a 32-bit address. This address mechanism is shown in the following diagrams:



**Figure 16-6. BDT Address Calculation**

**Table 16-2. BDT Address Calculation Fields**

Field	Description
BDT_PAGE	BDT_PAGE registers in the Control Register Block
END_POINT	END POINT field from the USB TOKEN
TX	1 for an TX transmit transfers and 0 for an RX receive transfers
ODD	This bit is maintained within the USB-FS SIE. It corresponds to the buffer currently in use. The buffers are used in a ping-pong fashion.

### 16.4.4 Buffer Descriptor Formats

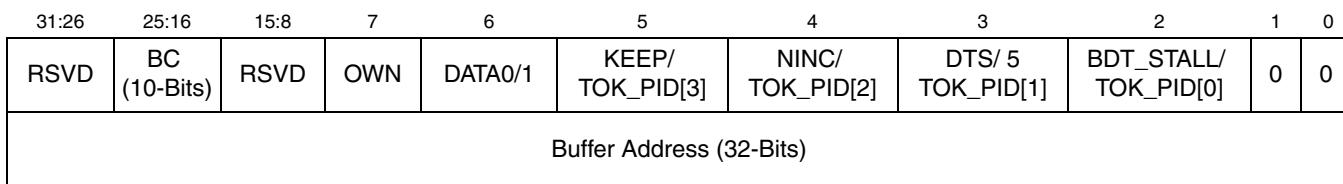
The Buffer Descriptors (BD) provide endpoint buffer control information for the USB-FS and microprocessor. The Buffer Descriptors have different meaning based on who is reading the BD in memory. The USB-FS Controller uses the data stored in the BDs to determine:

- Who owns the buffer in system memory
- Data0 or Data1 PID
- Release Own upon packet completion
- No address increment (FIFO Mode)
- Data toggle synchronization enable
- How much data is to be transmitted or received
- Where the buffer resides in system memory

While the microprocessor uses the data stored in the BDs to determine:

- Who owns the buffer in system memory
- Data0 or Data1 PID
- The received TOKEN PID
- How much data was transmitted or received
- Where the buffer resides in system memory

The format for the BD is shown in the following figure.



**Figure 16-7. Buffer Descriptor Byte Format**

**Table 16-3. Buffer Descriptor Byte Fields**

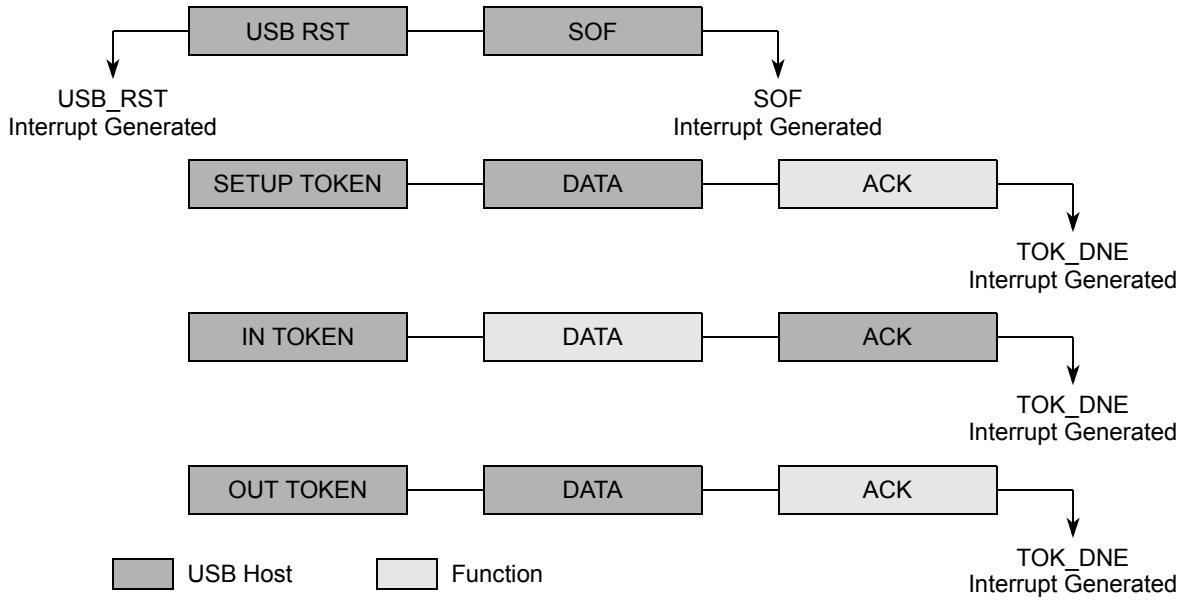
Field	Description
31 – 26 RSVD	Reserved
25 – 16 BC[9:0]	The Byte Count bits represent the 10-bit Byte Count. The USB-FS SIE changes this field upon the completion of a RX transfer with the byte count of the data received.
15 – 8 RSVD	Reserved
7 OWN	If OWN=1 USB-FS has exclusive access to the BD. If OWN=0 the microprocessor has exclusive access to the BD. This OWN bit determines who currently owns the buffer. The SIE generally writes a 0 to this bit when it has completed a token, except when KEEP=1. The USB-FS ignores all other fields in the BD when OWN=0. The microprocessor has access to the entire BD when OWN=0. This byte of the BD should always be the last byte the microprocessor updates when it initializes a BD. After the BD has been assigned to the USB-FS, the microprocessor should not change it in any way.

**Table 16-3. Buffer Descriptor Byte Fields (continued)**

6 DATA0/1	This bit defines if a DATA0 field (DATA0/1=0) or a DATA1 (DATA0/1=1) field was transmitted or received. It is unchanged by the USB-FS.
5 KEEP/ TOK_PID[3]	If KEEP equals 1, after the OWN bit is set it remains owned by the USB-FS forever. KEEP must equal 0 to allow the USB-FS to release the BD when a token has been processed. Typically this bit is set to 1 with ISO endpoints that are feeding a FIFO. The microprocessor is not informed that a token has been processed, the data is simply transferred to or from the FIFO. The NINC bit is normally also set when KEEP=1 to prevent address increment. If KEEP=1 this bit is unchanged by the USB-FS, otherwise bit 3 of the current token PID is written back in to the BD by the USB-FS.
4 NINC/ TOK_PID[2]	The No INCrement bit disables the DMA engine address increment. This forces the DMA engine to read or write from the same address. This is useful for endpoints when data needs to be read from or written to a single location such as a FIFO. Typically this bit is set with the KEEP bit for ISO endpoints that are interfacing to a FIFO. If KEEP=1 this bit is unchanged by the USB-FS, otherwise bit 2 of the current token PID is written back in to the BD by the USB-FS.
3 DTS/ TOK_PID[1]	Setting this bit enables the USB-FS to perform Data Toggle Synchronization. When this bit is 0 no Data Toggle Synchronization is performed. If KEEP=1 this bit is unchanged by the USB-FS, otherwise bit 1 of the current token PID is written back in to the BD by the USB-FS.
2 BDT_STALL TOK_PID[0]	Setting this bit causes the USB-FS to issue a STALL handshake if a token is received by the SIE that would use the BDT in this location. The BDT is not consumed by the SIE (the owns bit remains and the rest of the BDT are unchanged) when a BDT-STALL bit is set. If KEEP=1 this bit is unchanged by the USB-FS, otherwise bit 0 of the current token PID is written back in to the BD by the USB-FS
TOK_PID[n]	Bits [5:2] can also represent the current token PID. The current token PID is written back in to the BD by the USB-FS when a transfer completes. The values written back are the token PID values from the USB specification: 0x1 for an OUT token, 0x9 for an IN token or 0xd for a SETUP token. In host mode this field is used to report the last returned PID or a transfer status indication. The possible values returned are: 0x3 DATA0, 0xb DATA1, 0x2 ACK, 0xe STALL, 0xa NAK, 0x0 Bus Timeout, 0xf Data Error.
1–0 Reserved	Reserved, should read as zeroes
ADDR[31:0]	The Address bits represent the 32-bit buffer address in system memory. These bits are unchanged by the USB-FS.

### 16.4.5 USB Transaction

When the USB-FS transmits or receives data, it computes the BDT address using the address generation shown in Table 2. After the BDT has been read and if the OWN bit equals 1, the SIE transfers the data via the DMA to or from the buffer pointed to by the ADDR field of the BD. When the TOKEN is complete, the USB-FS updates the BDT and change the OWN bit to 0 if KEEP is 0. The STAT register is updated and the TOK\_DNE interrupt is set. When the microprocessor processes the TOK\_DNE interrupt it reads the status register, this gives the microprocessor all the information it needs to process the endpoint. At this point, the microprocessor allocates a new BD so additional USB data can be transmitted or received for that endpoint, and process then the last BD. The following figure shows a time line how a typical USB token would be processed.

**Figure 16-8. USB Token Transaction**

The USB has two sources for the DMA overrun error. First, the memory latency on the BVCI initiator interface may be too high and cause the receive FIFO to overflow. This is predominantly a hardware performance issue, usually caused by transient memory access issues. Second, the packet received may be larger than the negotiated *MaxPacket* size. This would be caused by a software bug.

In the first case, the USB responds with a NAK or Bus Timeout (BTO - See bit 4 in [Section 16.5.1.11, “Error Interrupt Status Register \(ERR\\_STAT\)”](#)) as appropriate for the class of transaction. The DMA\_ERR bit is set in the ERR\_STAT register of the core for host and device modes of operation. Depending on the values of the INT\_ENB and ERR\_ENB register, the core may assert an interrupt to notify the processor of the DMA error. In device mode, the BDT is not written back nor is the TOK\_DNE interrupt triggered because it is assumed that a second attempt is queued and succeed in the future. For host mode, the TOK\_DNE interrupt fires and the TOK\_PID field of the BDT is 1111 to indicate the DMA latency error. Host mode software can decide to retry or move to another item in its schedule.

In the second case of oversized data packets the USB specification is ambiguous. It assumes correct software drivers on both sides. The overrun is not due to memory latency but due to a lack of space to put the excess data. NAKing the packet may cause another retransmission of the already oversized packet data. In response to oversized packets, the USB core continues ACKing the packet for non-isochronous transfers. The data written to memory is clipped to the MaxPacket size so as not to corrupt system memory. The core asserts the DMA\_ERR bit of the ERR\_STAT register (which could trigger an interrupt, as above) and a TOK\_DNE interrupt fires. The TOK\_PID field of the BDT is not 1111 because the DMA\_ERR is not due to latency. The packet length field written back to the BDT is the MaxPacket value that represents the length of the clipped data actually written to memory. The software can decide an appropriate course of action from here for future transactions such as stalling the endpoint, canceling the transfer, disabling the endpoint, etc.

## 16.5 Memory Map/Register Definitions

This section provides the memory map and detailed descriptions of all USB interface registers. The memory map of the USB interface is shown in [Table 16-4](#).

**Table 16-4. USB Interface Memory Map**

Offset Address	Register	Acronym	Bits
<b>USB OTG Registers</b>			
0x0000	Peripheral ID Register	PER_ID	8
0x0004	Peripheral ID Complement Register	ID_COMP	8
0x0008	Peripheral Revision Register	REV	8
0x000C	Peripheral Additional Info Register	ADD_INFO	8
0x0010	OTG Interrupt Status Register	OTG_INT_STAT	8
0x0014	OTG Interrupt Control Register	OTG_INT_EN	8
0x0018	OTG Status Register	OTG_STATUS	8
0x001C	OTG Control Register	OTG_CTRL	8
0x0080	Interrupt Status Register	INT_STAT	8
0x0084	Interrupt Enable Register	INT_ENB	8
0x0088	Error Interrupt Status Register	ERR_STAT	8
0x008C	Error Interrupt Enable Register	ERR_ENG	8
0x0090	Status Register	STAT	8
0x0094	Control Register	CTL	8
0x0098	Address Register	ADDR	8
0x009C	BDT Page Register 1	BDT_PAGE_01	8
0x00A0	Frame Number Register Low	FRM_NUML	8
0x00A4	Frame Number Register High	FRM_NUMH	8
0x00A8	Token Register	TOKEN	8
0x00AC	SOF Threshold Register	SOF_THLD	8
0x00B0	BDT Page Register 2	BDT_PAGE_02	8
0x00B4	BDT Page Register 3	BDT_PAGE_03	8
0x00C0	Endpoint Control Register 0	ENDPT0	8
0x00C4	Endpoint Control Register 1	ENDPT1	8
0x00C8	Endpoint Control Register 2	ENDPT2	8
0x00CC	Endpoint Control Register 3	ENDPT3	8
0x00D0	Endpoint Control Register 4	ENDPT4	8
0x00D4	Endpoint Control Register 5	ENDPT5	8
0x00D8	Endpoint Control Register 6	ENDPT6	8
0x00DC	Endpoint Control Register 7	ENDPT7	8

**Table 16-4. USB Interface Memory Map (continued)**

<b>Offset Address</b>	<b>Register</b>	<b>Acronym</b>	<b>Bits</b>
0x00E0	Endpoint Control Register 8	ENDPT8	8
0x00E4	Endpoint Control Register 9	ENDPT9	8
0x00E8	Endpoint Control Register 10	ENDPT10	8
0x00EC	Endpoint Control Register 11	ENDPT11	8
0x00F0	Endpoint Control Register 12	ENDPT12	8
0x00F4	Endpoint Control Register 13	ENDPT13	8
0x00F8	Endpoint Control Register 14	ENDPT14	8
0x00FC	Endpoint Control Register 15	ENDPT15	8
0x0100	USB Control Register	USB_CTRL	8
0x0104	USB OTG Observe Register	USB_OTG_OBSERVE	8
0x0108	USB OTG Control Register	USB_OTG_CONTROL	8
0x010C	USB Transceiver and Regulator Control Register 0	USBTRC0	8
0x0110	OTG Pin Control Register	OTGPIN	8

The following sections provide details about the registers in the USB OTG memory map.

### 16.5.1 Capability Registers

The capability registers specify the software limits, restrictions, and capabilities of the host/device controller implementation. Most of these registers are defined by the EHCI specification. Registers that are not defined by the EHCI specification are noted in their descriptions.

### 16.5.1.1 Peripheral ID Register (PER\_ID)

The Peripheral ID Register reads back the value of 0x04. This value is defined for the USB Peripheral. Figure 16-9 shows the PER\_ID register.

								Access: User read-only								
R		7	6	5	4	3		2	1	0	W					
Reset:	0	0	0	0	0	0	0	1	0	0	W					

Figure 16-9. Peripheral ID Register (PER\_ID)

Table 16-11. PER\_ID Field Descriptions

Field	Description
7–6	These bits always read zeros
5–0 ID <sub>x</sub>	Peripheral identification bits. These bits always read 0x04 (00_0100).

### 16.5.1.2 Peripheral ID Complement Register (ID\_COMP)

The Peripheral ID Complement Register reads back the complement of the Peripheral ID Register. For the USB Peripheral, this is the value 0xFB. Figure 16-10 shows the ID\_COMP register.

								Access: User read-only								
R		7	6	5	4	3		2	1	0	W					
Reset:	1	1	1	1	1	1	1	0	1	1	W					

Figure 16-10. Peripheral ID Complement Register

Table 16-12. ID\_COMP Field Descriptions

Field	Description
7–6	These bits always read ones
5–0 NID <sub>x</sub>	Ones complement of peripheral identification bits.

### 16.5.1.3 Peripheral Revision Register (REV)

This register contains the revision number of the USB Module. [Figure 16-11](#) shows the REV register.

	Access: User read-only							
	7	6	5	4	3	2	1	0
R	REV7	REV6	REV5	REV4	REV3	REV2	REV1	REV0
W								
Reset:	0	0	1	1	0	0	1	1

**Figure 16-11. Peripheral Revision Register**

**Table 16-13. REV Field Descriptions**

Field	Description
7–0 REVx	REV[7:0] indicate the revision number of the USB Core.

### 16.5.1.4 Peripheral Additional Info Register (ADD\_INFO)

The Peripheral Additional info Register reads back the value of the fixed Interrupt Request Level (IRQ\_NUM) along with the Host Enable bit. If set to 1, the Host Enable bit indicates the USB peripheral is operating in host mode. [Figure 16-12](#) shows the ADD\_INFO register.

	Access: User read-only							
	7	6	5	4	3	2	1	0
R	IRQ_NUM					0	0	IEHOST
W								
Reset:	0	0	0	0	0	0	0	1

**Figure 16-12. Peripheral Additional Info Register**

**Table 16-14. ADD\_INFO Field Descriptions**

Field	Description
7–3 IRQ_NUM	Assigned Interrupt Request Number.
2–1 Reserved	RESERVED. These bits read back zeros.
0 IEHOST	This bit is set if host mode is enabled.

### 16.5.1.5 OTG Interrupt Status Register (OTG\_INT\_STAT)

The OTG Interrupt Status Register records changes of the ID sense and VBUS signals. Software can read this register to determine which event has caused an interrupt. Only bits that have changed since the last software read are set. Writing a one to a bit clears the associated interrupt. Figure 16-13 shows the OTG\_INT\_STAT register.

									Access: User read/write
	7	6	5	4	3	2	1	0	
R	ID_CHG	1_MSEC	LINE_STATE _CHG	Reserved	SESS_VLD _CHG	B_SESS _CHG	Reserved	A_VBUS _CHG	
W	0	1	1	X	0	0	X	0	

Reset: 0 1 1 X 0 0 X 0

Figure 16-13. OTG Interrupt Status Register

Table 16-15. OTG\_INT\_STAT Field Descriptions

Field	Description
7 ID_CHG	This bit is set when a change in the ID Signal from the USB connector is sensed.
6 1_MSEC	This bit is set when the 1 millisecond timer expires. This bit stays asserted until cleared by software. The interrupt must be serviced every millisecond to avoid losing 1msec counts.
5 LINE_STAT _CHG	This bit is set when the USB line state changes. The interrupt associated with this bit can be used to detect Reset, Resume, Connect, and Data Line Pulse signals.
4 Reserved	Reserved
3 SESS_VLD _CHG	This bit is set when a change in VBUS is detected indicating a session valid or a session no longer valid
2 B_SESS _CHG	This bit is set when a change in VBUS is detected on a B device.
1 Reserved	Reserved
0 A_VBUS _CHG	This bit is set when a change in VBUS is detected on an A device.

### 16.5.1.6 OTG Interrupt Control Register (OTG\_INT\_EN)

The OTG Interrupt Control Register enables the corresponding interrupt status bits defined in the OTG Interrupt Status Register. [Figure 16-14](#) shows the OTG\_INT\_EN register.

								Access: User read/write			
	7	6	5	4		3	2	1	0		
R	ID_EN	1_MSEC_EN	LINE_STATE_EN	Reserved	—	SESS_VLD_EN	B_SESS_EN	Reserved	A_VBUS_EN		
W	0	0	0	X		0	0	X	0		
Reset:	0	0	0	X		0	0	X	0		

**Figure 16-14. OTG Interrupt Control Register**

**Table 16-16. OTG\_INT\_EN Field Descriptions**

Field	Description
7 ID_EN	ID interrupt enable 0 The ID interrupt is disabled 1 The ID interrupt is enabled
6 1_MSEC_EN	1 millisecond interrupt enable 0 The 1msec timer interrupt is disabled 1 The 1msec timer interrupt is enabled
5 LINE_STATE_EN	Line State change interrupt enable 0 The LINE_STAT_CHG interrupt is disabled 1 The LINE_STAT_CHG interrupt is enabled
4	Reserved.
3 SESS_VLD_EN	Session valid interrupt enable 0 The SESS_VLD_CHG interrupt is disabled 1 The SESS_VLD_CHG interrupt is enabled
2 B_SESS_EN	B Session END interrupt enable 0 The B_SESS_CHG interrupt is disabled 1 The B_SESS_CHG interrupt is enabled
1	Reserved.
0 A_VBUS_EN	A VBUS Valid interrupt enable 0 The A_VBUS_CHG interrupt is disabled 1 The A_VBUS_CHG interrupt is enabled

### 16.5.1.7 Interrupt Status Register (OTG\_STAT)

The Interrupt Status Register displays the actual value from the external comparator outputs of the ID pin and VBUS. [Figure 16-15](#) shows the OTG\_STAT register.

									Access: User read/write	
R	7	6	5	4		3	2	1	0	
W	ID	1_MSEC_EN	LINE_STATE_STABLE	Reserved	—	SESS_VLD	B_SESS_END	Reserved	A_VBUS_VLD	
Reset:	0	0	1	X	—	0	0	X	0	

**Figure 16-15. Interrupt Status Register**

**Table 16-17. OTG\_STAT Field Descriptions**

Field	Description
7 ID	Indicates the current state of the ID pin on the USB connector 0 Indicates a Type A cable has been plugged into the USB connector 1 Indicates no cable is attached or a Type B cable has been plugged into the USB connector
6 1_MSEC_EN	This bit is reserved for the 1msec count, but it is not useful to software.
5 LINE_STATE_STABLE	This bit indicates that the internal signals that control the LINE_STATE_CHG bit (bit 5) of the OTG_INT_STAT register have been stable for at least 1 millisecond. First read the LINE_STATE_CHG bit, and then read this bit. If this bit reads as 1, then the value of LINE_STATE_CHG can be considered stable. 0 The LINE_STAT_CHG bit is not yet stable 1 The LINE_STAT_CHG bit has been debounced and is stable
4	Reserved.
3 SESS_VLD	Session Valid 0 The VBUS voltage is below the B session Valid threshold 1 The VBUS voltage is above the B session Valid threshold
2 B_SESS_END	B Session END 0 The VBUS voltage is above the B session End threshold 1 The VBUS voltage is below the B session End threshold
1	Reserved.
0 A_VBUS_VLD	A VBUS Valid 0 The VBUS voltage is below the A VBUS Valid threshold 1 The VBUS voltage is above the A VBUS Valid threshold

### 16.5.1.8 OTG Control Register (OTG\_CTRL)

The OTG Control Register controls the operation of VBUS and Data Line termination resistors. Figure 16-16 shows the OTG\_CTRL register.

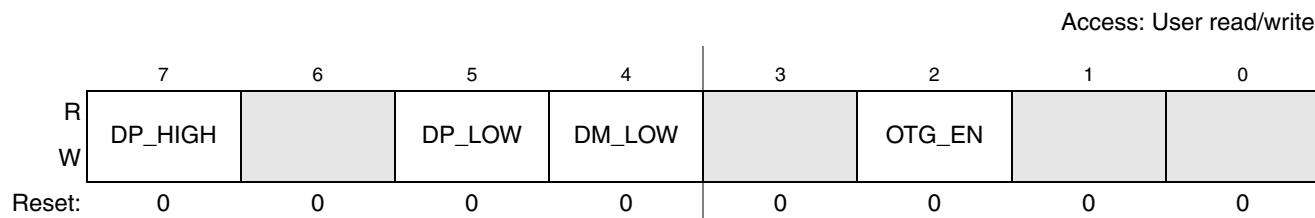


Figure 16-16. OTG Control Register

Table 16-18. OTG\_CTRL Field Descriptions

Field	Description
7 DP_HIGH	D+ Data Line pull-up resistor enable 0 D+ pull-up resistor is not enabled 1 D+ pull-up resistor is enabled
6 Reserved	Reserved.
5 DP_LOW	D+ Data Line pull-down resistor enable 0 D+ pull-down resistor is not enabled 1 D+ pull-down resistor is enabled This bit should always be enabled together with bit 4 (DM_LOW)
4 DM_LOW	D- Data Line pull-down resistor enable 0 D- pull-down resistor is not enabled 1 D- pull-down resistor is enabled This bit should always be enabled together with bit 5 (DP_LOW)
3 Reserved	Reserved.
2 OTG_EN	On-The-Go pull-up/pull-down resistor enable 0 If USB_EN is set and HOST_MODE is clear in the Control Register (CTL), then the D+ Data Line pull-up resistors are enabled. If HOST_MODE is set the D+ and D- Data Line pull-down resistors are engaged. 1 The pull-up and pull-down controls in this register are used
1 Reserved	Reserved.
0 Reserved	Reserved.

### 16.5.1.9 Interrupt Status Register (INT\_STAT)

The Interrupt Status Register contains bits for each of the interrupt sources within the USB Module. Each of these bits are qualified with their respective interrupt enable bits (see [Section 16.5.1.10, “Interrupt Enable Register \(INT\\_ENB\)”](#)). All bits of this register are logically OR’d together along with the OTG Interrupt Status Register (OTG\_STAT) to form a single interrupt source for the ColdFire core. After an interrupt bit has been set it may only be cleared by writing a one to the respective interrupt bit. This register contains the value of 0x00 after a reset. [Figure 16-17](#) shows the INT\_STAT register.

								Access: User read/write	
	7	6	5	4	3	2	1	0	
R	STALL	ATTACH	RESUME	SLEEP	TOK_DNE	SOF_TOK	ERROR	USB_RST	
Reset:	0	0	0	1	0	0	0	0	

**Figure 16-17. Interrupt Status Register**

**Table 16-19. INT\_STAT Field Descriptions**

Field	Description
7 STALL	Stall Interrupt In Target mode this bit is asserted when a STALL handshake is sent by the SIE. In Host mode this bit is set when the USB Module detects a STALL acknowledge during the handshake phase of a USB transaction. This interrupt can be used to determine if the last USB transaction was completed successfully or if it stalled.
6 ATTACH	Attach Interrupt This bit is set when the USB Module detects an attach of a USB device. This signal is only valid if HOST_MODE_EN is true. This interrupt signifies that a peripheral is now present and must be configured.
5 RESUME	This bit is set depending upon the DP/DM signals, and can be used to signal remote wake-up signaling on the USB bus. When not in suspend mode this interrupt should be disabled.
4 SLEEP	This bit is set when the USB Module detects a constant idle on the USB bus for 3 milliseconds. The sleep timer is reset by activity on the USB bus.
3 TOK_DNE	This bit is set when the current token being processed has completed. The ColdFire core should immediately read the STAT register to determine the EndPoint and BD used for this token. Clearing this bit (by writing a one) causes the STAT register to be cleared or the STAT holding register to be loaded into the STAT register.
2 SOF_TOK	This bit is set when the USB Module receives a Start Of Frame (SOF) token. In Host mode this bit is set when the SOF threshold is reached, so that software can prepare for the next SOF.
1 ERROR	This bit is set when any of the error conditions within the ERR_STAT register occur. The ColdFire core must then read the ERR_STAT register to determine the source of the error.
0 USB_RST	This bit is set when the USB Module has decoded a valid USB reset. This informs the Microprocessor that it should write 0x00 into the address register and enable endpoint 0. USB_RST is set after a USB reset has been detected for 2.5 microseconds. It is not asserted again until the USB reset condition has been removed and then reasserted.

### 16.5.1.10 Interrupt Enable Register (INT\_ENB)

The Interrupt Enable Register contains enable bits for each of the interrupt sources within the USB Module. Setting any of these bits enables the respective interrupt source in the INT\_STAT register. This register contains the value of 0x00 after a reset. [Figure 16-18](#) shows the INT\_ENB register.

								Access: User read/write	
	7	6	5	4	3	2	1	0	
R	STALL_EN	ATTACH_EN	RESUME_EN	SLEEP_EN	TOK_DNE_EN	SOF_TOK_EN	ERROR_EN	USB_RST_EN	
W	0	0	0	0	0	0	0	0	

Reset: 0 0 0 0 0 0 0 0 0

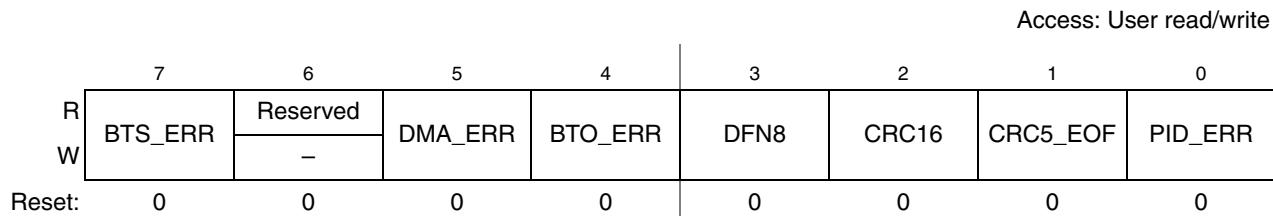
**Figure 16-18. Interrupt Enable Register**

**Table 16-20. INT\_ENB Field Descriptions**

Field	Description
7 STALL_EN	STALL Interrupt Enable 0 The STALL interrupt is not enabled 1 The STALL interrupt is enabled
6 ATTACH_EN	ATTACH Interrupt Enable 0 The ATTACH interrupt is not enabled 1 The ATTACH interrupt is enabled
5 RESUME_EN	RESUME Interrupt Enable 0 The RESUME interrupt is not enabled 1 The RESUME interrupt is enabled
4 SLEEP_EN	SLEEP Interrupt Enable 0 The SLEEP interrupt is not enabled 1 The SLEEP interrupt is enabled
3 TOK_DNE_EN	TOK_DNE Interrupt Enable 0 The TOK_DNE interrupt is not enabled 1 The TOK_DNE interrupt is enabled
2 SOF_TOK_EN	SOF_TOK Interrupt Enable 0 The SOF_TOK interrupt is not enabled 1 The SOF_TOK interrupt is enabled
1 ERROR_EN	ERROR Interrupt Enable 0 The ERROR interrupt is not enabled 1 The ERROR interrupt is enabled
0 USB_RST_EN	USB_RST Interrupt Enable 0 The USB_RST interrupt is not enabled 1 The USB_RST interrupt is enabled

### 16.5.1.11 Error Interrupt Status Register (ERR\_STAT)

The Error Interrupt Status Register contains enable bits for each of the error sources within the USB Module. Each of these bits are qualified with their respective error enable bits (see [Section 16.5.1.12, “Error Interrupt Enable Register \(ERR\\_ENB\)”\). All bits of this Register are logically OR’d together and the result placed in the ERROR bit of the INT\\_STAT register. After an interrupt bit has been set it may only be cleared by writing a one to the respective interrupt bit. Each bit is set as soon as the error conditions is detected. Therefore, the interrupt does not typically correspond with the end of a token being processed. This register contains the value of 0x00 after a reset. \[Figure 16-19\]\(#\) shows the ERR\\_STAT register.](#)



**Figure 16-19. Error Interrupt Status Register**

**Table 16-21. ERR\_STAT Field Descriptions**

Field	Description
7 BTS_ERR	This bit is set when a bit stuff error is detected. If set, the corresponding packet is rejected due to the error.
6	Reserved
5 DMA_ERR	This bit is set if the USB Module has requested a DMA access to read a new BDT but has not been given the bus before it needs to receive or transmit data. If processing a TX transfer this would cause a transmit data underflow condition. If processing a RX transfer this would cause a receive data overflow condition. This interrupt is useful when developing device arbitration hardware for the microprocessor and the USB Module to minimize bus request and bus grant latency. This bit is also set if a data packet to or from the host is larger than the buffer size allocated in the BDT. In this case the data packet is truncated as it is put into buffer memory.
4 BTO_ERR	This bit is set when a bus turnaround timeout error occurs. The USB Module contains a bus turnaround timer that keeps track of the amount of time elapsed between the token and data phases of a SETUP or OUT TOKEN or the data and handshake phases of a IN TOKEN. If more than 16 bit times are counted from the previous EOP before a transition from IDLE, a bus turnaround timeout error occurs.
3 DFN8	This bit is set if the data field received was not 8 bits in length. USB Specification 1.0 requires that data fields be an integral number of bytes. If the data field was not an integral number of bytes, this bit is set.
2 CRC16	This bit is set when a data packet is rejected due to a CRC16 error.
1 CRC5_EOF	This error interrupt has two functions. When the USB Module is operating in peripheral mode (HOST_MODE_EN=0), this interrupt detects CRC5 errors in the token packets generated by the host. If set the token packet was rejected due to a CRC5 error. When the USB Module is operating in host mode (HOST_MODE_EN=1), this interrupt detects End Of Frame (EOF) error conditions. This occurs when the USB Module is transmitting or receiving data and the SOF counter reaches zero. This interrupt is useful when developing USB packet scheduling software to ensure that no USB transactions cross the start of the next frame.
0 PID_ERR	This bit is set when the PID check field fails.

### 16.5.1.12 Error Interrupt Enable Register (ERR\_ENB)

The Error Interrupt Enable Register contains enable bits for each of the error interrupt sources within the USB Module. Setting any of these bits enables the respective interrupt source in the ERR\_STAT register. Each bit is set as soon as the error conditions is detected. Therefore, the interrupt does not typically correspond with the end of a token being processed. This register contains the value of 0x00 after a reset. Figure 16-20 shows the ERR\_ENB register.

									Access: User read/write
	7	6	5	4	3	2	1	0	
R	BTS_ERR _EN	Reserved	DMA_ERR _EN	BTO_ERR _EN	DFN8_EN	CRC16_EN	CRC5_EOF _EN	PID_ERR _EN	
W	–	–	0	0	0	0	0	0	

Reset: 0 0 0 0 0 0 0 0 0

Figure 16-20. Error Interrupt Enable Register

Table 16-22. ERR\_ENB Field Descriptions

Field	Description
7 BTS_ERR _EN	BTS_ERR Interrupt Enable 0 The BTS_ERR interrupt is not enabled 1 The BTS_ERR interrupt is enabled
6	Reserved
5 DMA_ERR _EN	DMA_ERR Interrupt Enable 0 The DMA_ERR interrupt is not enabled 1 The DMA_ERR interrupt is enabled
4 BTO_ERR _EN	BTO_ERR Interrupt Enable 0 The BTO_ERR interrupt is not enabled 1 The BTO_ERR interrupt is enabled
3 DFN8_EN	DFN8 Interrupt Enable 0 The DFN8 interrupt is not enabled 1 The DFN8 interrupt is enabled
2 CRC16_EN	CRC16 Interrupt Enable 0 The CRC16 interrupt is not enabled 1 The CRC16 interrupt is enabled
1 CRC5_EOF _EN	CRC5/EOF Interrupt Enable 0 The CRC5/EOF interrupt is not enabled 1 The CRC5/EOF interrupt is enabled
0 PID_ERR _EN	PID_ERR Interrupt Enable 0 The PID_ERR interrupt is not enabled 1 The PID_ERR interrupt is enabled

### 16.5.1.13 Status Register (STAT)

The Status Register reports the transaction status within the USB Module. When the ColdFire core has received a TOK\_DNE interrupt the Status Register should be read to determine the status of the previous endpoint communication. The data in the status register is valid when the TOK\_DNE interrupt bit is asserted. The STAT register is actually a read window into a status FIFO maintained by the USB Module. When the USB Module uses a BD, it updates the Status Register. If another USB transaction is performed before the TOK\_DNE interrupt is serviced, the USB Module stores the status of the next transaction in the STAT FIFO. Thus the STAT register is actually a four byte FIFO that allows the ColdFire core to process one transaction while the SIE is processing the next transaction. Clearing the TOK\_DNE bit in the INT\_STAT register causes the SIE to update the STAT register with the contents of the next STAT value. If the data in the STAT holding register is valid, the SIE immediately reasserts to TOK\_DNE interrupt. Figure 16-21 shows the STAT register.

								Access: User read-only			
	7	6	5	4		3	2	1	0		
R	ENDP				TX	ODD	Reserved	Reserved			
W											
Reset:	0	0	0	0	0	0	0	0	0		

Figure 16-21. Status Register

Table 16-23. STAT Field Descriptions

Field	Description
7 - 5 ENDP[3:0]	This four-bit field encodes the endpoint address that received or transmitted the previous token. This allows the ColdFire core to determine which BDT entry was updated by the last USB transaction.
3 TX	Transmit Indicator 0 The most recent transaction was a Receive operation 1 The most recent transaction was a Transmit operation
2 ODD	this bit is set if the last Buffer Descriptor updated was in the odd bank of the BDT.
1 - 0	Reserved

### 16.5.1.14 Control Register (CTL)

The Control Register provides various control and configuration information for the USB Module. Figure 16-22 shows the CTL register.

									Access: User read/write
R W	7 JSTATE	6 SE0	5 TXSUSPEND/ TOKENBUSY	4 RESET	3 HOST_ MODE_EN	2 RESUME	1 ODD_RST	0 USB_EN/ SOF_EN	
Reset:	1	1	0	0	0	0	0	0	

Figure 16-22. Control Register

Table 16-24. CTL Field Descriptions

Field	Description
7 JSTATE	Live USB differential receiver JSTATE signal. The polarity of this signal is affected by the current state of LS_EN (See)
6 SE0	Live USB Single Ended Zero signal
5 TXSUSPEND/ TOKENBUSY	When the USB Module is in Host mode TOKEN_BUSY is set when the USB Module is busy executing a USB token and no more token commands should be written to the Token Register. Software should check this bit before writing any tokens to the Token Register to ensure that token commands are not lost. In Target mode TXD_SUSPEND is set when the SIE has disabled packet transmission and reception. Clearing this bit allows the SIE to continue token processing. This bit is set by the SIE when a Setup Token is received allowing software to dequeue any pending packet transactions in the BDT before resuming token processing.
4 RESET	Setting this bit enables the USB Module to generate USB reset signaling. This allows the USB Module to reset USB peripherals. This control signal is only valid in Host mode (HOST_MODE_EN=1). Software must set RESET to 1 for the required amount of time and then clear it to 0 to end reset signaling. For more information on RESET signaling see Section 7.1.4.3 of the USB specification version 1.0.
3 HOST_ MODE_EN	When set to 1, this bit enables the USB Module to operate in Host mode. In host mode, the USB module performs USB transactions under the programmed control of the host processor.
2 RESUME	When set to 1 this bit enables the USB Module to execute resume signaling. This allows the USB Module to perform remote wake-up. Software must set RESUME to 1 for the required amount of time and then clear it to 0. If the HOST_MODE_EN bit is set, the USB module appends a Low Speed End of Packet to the Resume signaling when the RESUME bit is cleared. For more information on RESUME signaling see Section 7.1.4.5 of the USB specification version 1.0.
1 ODD_RST	Setting this bit to 1 resets all the BDT ODD ping/pong bits to 0, which then specifies the EVEN BDT bank.
0 USB_EN/ SOF_EN	USB Enable 0 The USB Module is disabled 1 The USB Module is enabled. Setting this bit causes the SIE to reset all of its ODD bits to the BDTs. Therefore, setting this bit resets much of the logic in the SIE. When host mode is enabled, clearing this bit causes the SIE to stop sending SOF tokens.

### 16.5.1.15 Address Register (ADDR)

The Address Register holds the unique USB address that the USB Module decodes when in Peripheral mode (HOST\_MODE\_EN=0). When operating in Host mode (HOST\_MODE\_EN=1) the USB Module transmits this address with a TOKEN packet. This enables the USB Module to uniquely address an USB peripheral. In either mode, the USB\_EN bit within the control register must be set. The Address Register is reset to 0x00 after the reset input becomes active or the USB Module decodes a USB reset signal. This action initializes the Address Register to decode address 0x00 as required by the USB specification.

Figure 16-23 shows the ADDR register.



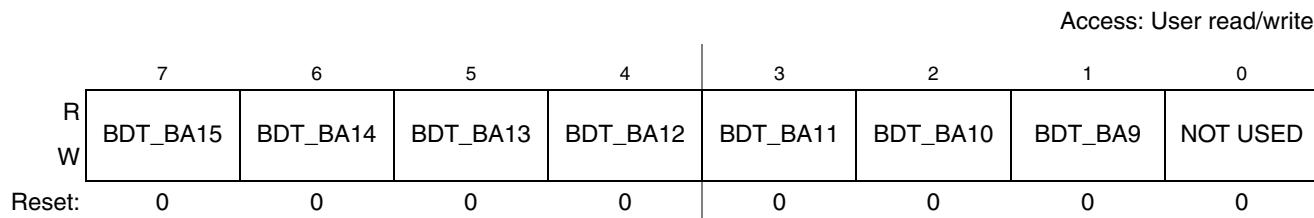
Figure 16-23. ADDR Register

Table 16-25. ADDR Field Descriptions

Field	Description
7 LS_EN	Low Speed Enable bit. This bit informs the USB Module that the next token command written to the token register must be performed at low speed. This enables the USB Module to perform the necessary preamble required for low-speed data transmissions.
6–0 ADDR	USB address. This 7-bit value defines the USB address that the USB Module decodes in peripheral mode, or transmit when in host mode.

### 16.5.1.16 BDT Page Register 1 (BDT\_PAGE\_01)

The Buffer Descriptor Table Page Register 1 contains an 8-bit value used to compute the address where the current Buffer Descriptor Table (BDT) resides in system memory. [Figure 16-24](#) shows the BDT Page Register 1.



**Figure 16-24. BDT\_PAGE\_01 Register**

**Table 16-26. BDT\_PAGE\_01 Field Descriptions**

Field	Description
7 – 1 BDT_BA[15:8]	This 7 bit field provides address bits 15 through 9 of the BDT base address, which defines where the Buffer Descriptor Table resides in system memory.
0 NOT USED	This bit is always zero. The 32-bit BDT Base Address is always aligned on 512 byte boundaries in memory.

### 16.5.1.17 Frame Number Register Low/High (FRM\_NUML, FRM\_NUMH)

The Frame Number Register contains an 11-bit value used to compute the address where the current Buffer Descriptor Table (BDT) resides in system memory. [Figure 16-25](#) shows the FRM\_NUML Register.

								Access: User read-only
	7	6	5	4	3	2	1	0
R	FRM7	FRM6	FRM5	FRM4	FRM3	FRM2	FRM1	FRM0
W								
Reset:	0	0	0	0	0	0	0	0

**Figure 16-25. FRM\_NUML Register**

**Table 16-27. FRM\_NUML Field Descriptions**

Field	Description
7–0 FRM[7:0]	These 8 bits represent the low-order bits of the 11-bit Frame Number

								Access: User read-only
	7	6	5	4	3	2	1	0
R	0	0	0	0	0	FRM10	FRM9	FRM8
W								
Reset:	0	0	0	0	0	0	0	0

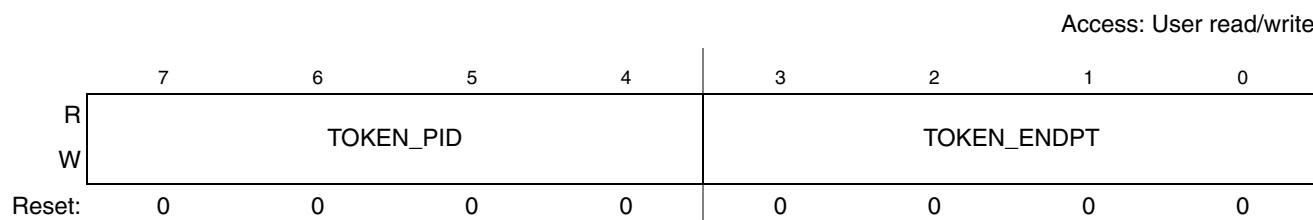
**Figure 16-26. FRM\_NUMH Register**

**Table 16-28. FRM\_NUMH Field Descriptions**

Field	Description
2–0 FRM[10:8]	These 3 bits represent the high-order bits of the 11-bit Frame Number
7–3 NOT USED	This bits always read zero.

### 16.5.1.18 Token Register (TOKEN)

The Token Register is used to perform USB transactions when in host mode (HOST\_MODE\_EN=1). When the ColdFire core processor wishes to execute a USB transaction to a peripheral, it writes the TOKEN type and endpoint to this register. After this register has been written, the USB module begins the specified USB transaction to the address contained in the address register. The ColdFire core should always check that the TOKEN\_BUSY bit in the control register is not set before performing a write to the Token Register. This ensures token commands are not overwritten before they can be executed. The address register and endpoint control register 0 are also used when performing a token command and therefore must also be written before the Token Register. The address register is used to correctly select the USB peripheral address transmitted by the token command. The endpoint control register determines the handshake and retry policies used during the transfer. [Figure 16-27](#) shows the TOKEN Register.



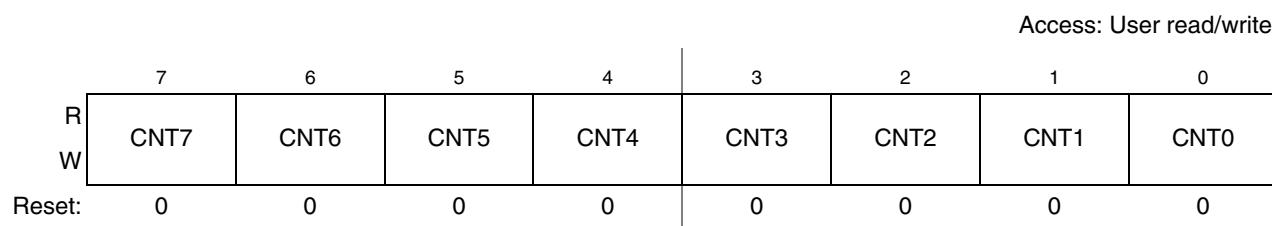
**Figure 16-27. TOKEN Register**

**Table 16-29. TOKEN Field Descriptions**

Field	Description
7 – 4 TOKEN _ENDPT	This 4 bit field holds the Endpoint address for the token command. The four bit value written must be a valid endpoint.
3 – 0 TOKEN_PID	This 4-bit field contains the token type executed by the USB Module. Valid tokens are: TOKEN_PID=0001      OUT Token      USB Module performs an OUT (TX) transaction TOKEN_PID=1001      IN Token      USB Module performs an In (RX) transaction TOKEN_PID=1101      SETUP Token      USB Module performs a SETUP (TX) transaction

### 16.5.1.19 SOF Threshold Register (SOF\_THLD)

The SOF Threshold Register is used only in Hosts mode (HOST\_MODE\_EN=1). When in Host mode, the 14-bit SOF counter counts the interval between SOF frames. The SOF must be transmitted every 1msec so the SOF counter is loaded with a value of 12000. When the SOF counter reaches zero, a Start Of Frame (SOF) token is transmitted. The SOF threshold register is used to program the number of USB byte times *before* the SOF to stop initiating token packet transactions. This register must be set to a value that ensures that other packets are not actively being transmitted when the SOF time counts to zero. When the SOF counter reaches the threshold value, no more tokens are transmitted until after the SOF has been transmitted. The value programmed into the threshold register must reserve enough time to ensure the worst case transaction completes. In general the worst case transaction is a IN token followed by a data packet from the target followed by the response from the host. The actual time required is a function of the maximum packet size on the bus. Typical values for the SOF threshold are: 64-byte packets=74; 32-byte packets=42; 16-byte packets=26; 8-byte packets=18. [Figure 16-28](#) shows the SOF\_THLD Register.



**Figure 16-28. SOF\_THLD Register**

**Table 16-30. SOF\_THLD Field Descriptions**

Field	Description
7 – 0 CNT[7:0]	This 8 bit field represents the SOF count threshold in byte times.

### 16.5.1.20 BDT Page Register 2 (BDT\_PAGE\_02)

The Buffer Descriptor Table Page Register 2 contains an 8-bit value used to compute the address where the current Buffer Descriptor Table (BDT) resides in system memory. See [Section 16.5.1.16, “BDT Page Register 1 \(BDT\\_PAGE\\_01\)”](#) for more information on the format of the Buffer Descriptor Table.

[Figure 16-29](#) shows the BDT Page Register 2.

	Access: User read/write							
	7	6	5	4	3	2	1	0
R	BDT_BA23	BDT_BA22	BDT_BA21	BDT_BA20	BDT_BA19	BDT_BA18	BDT_BA17	BDT_BA16
W	0	0	0	0	0	0	0	0

Reset: 0 0 0 0 0 0 0 0 0

**Figure 16-29. BDT\_PAGE\_02 Register**

**Table 16-31. BDT\_PAGE\_02 Field Descriptions**

Field	Description
7 – 0 BDT_BA[23:16]	This 8 bit field provides address bits 23 through 16 of the BDT base address, which defines where the Buffer Descriptor Table resides in system memory.

### 16.5.1.21 BDT Page Register 3 (BDT\_PAGE\_03)

The Buffer Descriptor Table Page Register 3 contains an 8-bit value used to compute the address where the current Buffer Descriptor Table (BDT) resides in system memory. See [Section 16.5.1.16, “BDT Page Register 1 \(BDT\\_PAGE\\_01\)”](#) for more information on the format of the Buffer Descriptor Table.

[Figure 16-30](#) shows the BDT Page Register 3.

	Access: User read/write							
	7	6	5	4	3	2	1	0
R	BDT_BA31	BDT_BA30	BDT_BA29	BDT_BA28	BDT_BA27	BDT_BA26	BDT_BA25	BDT_BA24
W	0	0	0	0	0	0	0	0

Reset: 0 0 0 0 0 0 0 0 0

**Figure 16-30. BDT\_PAGE\_03 Register**

**Table 16-32. BDT\_PAGE\_03 Field Descriptions**

Field	Description
7 – 0 BDT_BA[31:24]	This 8 bit field provides address bits 31through 24 of the BDT base address, which defines where the Buffer Descriptor Table resides in system memory.

### 16.5.1.22 Endpoint Control Registers 0 – 15 (ENDPT0–15)

The Endpoint Control Registers contain the endpoint control bits for each of the 16 endpoints available within the USB Module for a decoded address. The format for these registers is shown in the following figure. Endpoint 0 (ENDPT0) is associated with control pipe 0, which is required for all USB functions. Therefore, after a USB\_RST interrupt occurs the ColdFire core should set the ENDPT0 register to contain 0x0D.

In Host mode ENDPT0 is used to determine the handshake, retry and low speed characteristics of the host transfer. For Host mode control, bulk and interrupt transfers the EP\_HSHK bit should be set to 1. For Isochronous transfers it should be set to 0. Common values to use for ENDPT0 in host mode are 0x4D for Control, Bulk, and Interrupt transfers, and 0x4C for Isochronous transfers.

Figure 16-31 shows the Endpoint Control Registers.

								Access: User read/write	
	7	6	5	4		3	2	1	0
R W	HOST_WO_HUB	RETRY_DIS	0	EP_CTL_DIS	EP_RX_EN	EP_TX_EN	EP_STALL	EP_HSHK	
Reset:	0	0	0	0	0	0	0	0	0

Figure 16-31. Endpoint Control Registers

Table 16-33. Endpoint Control Registers Field Descriptions

Field	Description
7 HOST_WO_HUB	This is a Host mode only bit and is only present in the control register for endpoint 0 (ENDPT0). When set this bit allows the host to communicate to a directly connected low speed device. When cleared, the host produces the PRE_PID then switch to low speed signaling when sending a token to a low speed device as required to communicate with a low speed device through a hub.
6 RETRY_DIS	This is a Host mode only bit and is only present in the control register for endpoint 0 (ENDPT0). When set this bit causes the host to not retry NAK'ed (Negative Acknowledgement) transactions. When a transaction is NAKed, the BDT PID field is updated with the NAK PID, and the TOKEN_DNE interrupt is set. When this bit is cleared NAKed transactions is retried in hardware. This bit must be set when the host is attempting to poll an interrupt endpoint.
5	Reserved
4 EP_CTL_DIS	This bit, when set, disables control (SETUP) transfers. When cleared, control transfers are enabled. This applies if and only if the EP_RX_EN and EP_TX_EN bits are also set. See <a href="#">Table 16-34</a>
3 EP_RX_EN	This bit, when set, enables the endpoint for RX transfers. See <a href="#">Table 16-34</a>
2 EP_TX_EN	This bit, when set, enables the endpoint for TX transfers. See <a href="#">Table 16-34</a>
1 EP_STALL	When set this bit indicates that the endpoint is stalled. This bit has priority over all other control bits in the EndPoint Enable Register, but it is only valid if EP_TX_EN=1 or EP_RX_EN=1. Any access to this endpoint causes the USB Module to return a STALL handshake. After an endpoint is stalled it requires intervention from the Host Controller.
0 EP_HSHK	When set this bit enables an endpoint to perform handshaking during a transaction to this endpoint. This bit is generally set unless the endpoint is Isochronous.

**Table 16-34. Endpoint Direction and Control**

<b>EPL_CTL_DIS</b>	<b>EP_RX_EN</b>	<b>EP_TX_EN</b>	<b>Endpoint Enable / Direction Control</b>
X	0	0	Disable Endpoint
X	0	1	Enable Endpoint for TX transfers only
X	1	0	Enable Endpoint for RX transfers only
1	1	1	Enable Endpoint for RX and TX transfers
0	1	1	Enable Endpoint for RX and TX as well as control (SETUP) transfers

### 16.5.1.23 USB Control Register (USB\_CTRL)

Access: User read/write

	7	6	5	4	3	2	1	0
R	SUSP	PDE	—	—	—	—	CLK_SRC	
W								
Reset:	1	1	0	0	0	0	1	1

Figure 16-32. USB Control Register

Table 16-35. USB\_CTRL Field Descriptions

Field	Description
7 SUSP	Places the USB transceiver into the suspend state. 0 USB transceiver is not in suspend state. 1 USB transceiver is in suspend state.
6 PDE	Enables the non-functional weak pulldowns on the USB transceiver 0 Weak pulldowns are disabled on D+ and D– 1 Weak pulldowns are enabled on D+ and D–
5 –2	Reserved
1-0 CLK_SRC	Determines the clock source for the USB 48 MHz clock 00 USB_ALT_CLK pin (External clock that can feed in from PTG0) 01 External OSC on EXTAL pin 10 Reserved 11 System clock source (MCGPLLCLK)

### 16.5.1.24 USB OTG Observe Register (USB\_OTG\_OBSERVE)

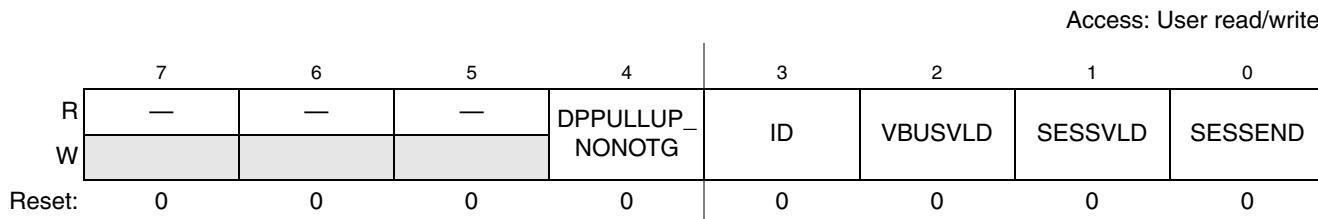
Access: User read/write

	7	6	5	4	3	2	1	0
R	DP_PU	DP_PD	0	DM_PD	—	—	—	1
W			0		0	0	0	—
Reset:	0	0	0	0	0	0	0	0

Figure 16-33. USB OTG Observe Register

**Table 16-36. USB\_OTG\_OBSERVE Field Descriptions**

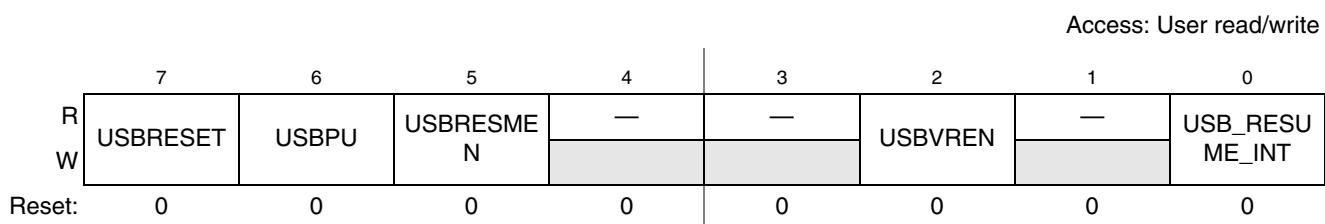
<b>Field</b>	<b>Description</b>
7 DP_PU	Provides observability of the D+ Pull Up signal output from the USB OTG module. This bit is useful when interfacing to an external OTG control module via a serial interface. 0 D+ pullup disabled. 1 D+ pullup enabled.
6 DP_PD	Provides observability of the D+ Pull Down signal output from the USB OTG module. Useful when interfacing to an external OTG control module via a serial interface. 0 D+ pulldown disabled. 1 D+ pulldown enabled.
5 Reserved	Reserved. Should always read zero.
4 DM_PD	Provides observability of the D+ Pull Down signal output from the USB OTG module. Useful when interfacing to an external OTG control module via a serial interface. 0 D+ pulldown disabled. 1 D+ pulldown enabled.
3-0 Reserved	Reserved

**16.5.1.25 USB OTG Control Register (USB\_OTG\_CONTROL)****Figure 16-34. USB OTG Control Register****Table 16-37. USB\_OTG\_CONTROL Field Descriptions**

<b>Field</b>	<b>Description</b>
7 — 5	Reserved
4 DPPULLUP_ NONOTG	Provides control of the DP PULLUP in the USB OTG module, if USB is configured in non-OTG device mode. 0 DP Pull up in non-OTG device mode is not enabled. 1 DP Pull up in non-OTG device mode is enabled.
3 ID	Provides control of the USB ID signal into the USB OTG module if a pin has not been configured for this function. Useful when interfacing to an external OTG control module via a serial interface. 0 USB ID input is negated. 1 USB ID input is asserted.
2 VBUSVLD	Provides control of the VBUS Valid signal into the USB OTG module if a pin has not been configured for this function. Useful when interfacing to an external OTG control module via a serial interface. 0 VBUS Valid input is negated. 1 VBUS Valid input is asserted.

**Table 16-37. USB\_OTG\_CONTROL Field Descriptions**

Field	Description
1 SESSVLD	Provides observability of the Session Valid signal output from the USB OTG module. Useful when interfacing to an external OTG control module via a serial interface. 0 Session Valid input is negated. 1 Session Valid input is asserted.
0 SESSEND	Provides observability of the Session End signal output from the USB OTG module. Useful when interfacing to an external OTG control module via a serial interface. 0 Session End input is negated. 1 Session End input is asserted.

**16.5.1.26 USB Transceiver and Regulator Control Register 0 (USBTRC0)****Figure 16-35. USB Transceiver and Regulator Control Register 0 (USBTRC0)****Table 16-38. USBTRC0 Field Descriptions**

Field	Description
7 USBRESET	USB reset bit. This bit generates a hard reset to the USB module, including USB PHY and USB VREG enables. After this bit is set and the reset occurs, this bit is automatically cleared. 0 Normal USB module operation 1 Returns the USB module to its reset state
6 USBUPU	Pull-up source bit. This bit determines the source of the pull-up resistor on the USBDP line. 0 Internal USBDP pull-up resistor is disabled; the application can use an external pull-up resistor 1 Internal USBDP pull-up resistor is enabled
5 USBRESME N	USB resume event enable bit. This bit, when set, allows the USB module to send an asynchronous wakeup event to the MCU upon detection of resume signaling on the USB bus. The MCU then re-enables clocks to the USB module. It is used for low-power suspend mode when USB module clocks are stopped. This bit should be set only after SLEEPF=1. Async wakeup only works in device mode. 0 USB asynchronous wakeup from suspend mode disabled 1 USB asynchronous wakeup from suspend mode enabled
4–3	Reserved, should be cleared.
2 USBVREN	USB voltage regulator enable bit. This bit enables the on-chip 3.3V USB voltage regulator. 0 On-chip USB voltage regulator is disabled (OFF MODE) 1 On-chip USB voltage regulator is enabled for Active or Standby mode
1	Reserved
0 USB_RESUM E_INT	USB Asynchronous Interrupt — This bit's value is interpreted as follows: 0 No interrupt was generated 1 Interrupt was generated because of the USB asynchronous interrupt

### 16.5.1.27 OTG Pin Control Register (OTGPIN)

Access: User read/write

	7	6	5	4	3	2	1	0
R	—	USBID	DMDOWN	DPDOWN	PULLUP	VBUSVLD	SESEND	SESVLD
W								
Reset:	0	0	0	0	0	0	0	0

Figure 16-36. OTG Pin Control Register (OTGPIN)

Table 16-39. OTGPIN Field Descriptions

Field	Description
7	Reserved, should be cleared.
6 USBID	USB_ID pin enable bit. This bit allows the USB_ID input pin to be provided externally. The USB_ID pin directly affects the ID bit in the OTG_STAT register. 0 USB_ID pin is not connected externally 1 USB_ID is connected to PortA bit 6. The normal port logic and GPIO functions are disabled.
5 DMDOWN	DMDOWN pin enable bit. This bit allows the DM_DOWN output pin to be provided externally, allowing users to connect a pull-down resistor externally to the D- signal. The DM_DOWN pin works in conjunction with the DM_LOW bit of the OTG_CTRL register. Read <a href="#">Section 16.6.1, “Configuration of External Pull-up/Pull-down for USB”</a> to configure the external pull-up/pull-down. 0 DM_DOWN pin is not connected externally 1 DM_DOWN is connected to PortA bit 4. The normal port logic and GPIO functions are disabled.
4 DPDOWN	DPDOWN pin enable bit. This bit allows the DP_DOWN output pin to be provided externally, allowing users to connect a pull-down resistor externally to the D- signal. The DP_DOWN pin works in conjunction with the DP_LOW bit of the OTG_CTRL register. Read <a href="#">Section 16.6.1, “Configuration of External Pull-up/Pull-down for USB”</a> to configure the external pull-up/pull-down. 0 DP_DOWN pin is not connected externally 1 DP_DOWN is connected to PortA bit 5. The normal port logic and GPIO functions are disabled.
3 PULLUP	PULLUP pin enable bit. The USB_PULLUP pin is an output pin from the USB module. It works in conjunction with the USBPU bit in the USBCTRLC0 register. If USBPU is 0 then no internal pull-up resistor is deployed on the D+ signal line. Setting the DP_HI bit in the OTGCTRL register outputs the 3.3V VREG voltage(if enabled) to the USB_PULLUP pin. If DP_HI bit is 0 then USB_PULLUP is tri-state/high impedance. Read <a href="#">Section 16.6.1, “Configuration of External Pull-up/Pull-down for USB”</a> to configure the external pull-up/pull-down. 0 USB_PULLUP output is connected externally 1 USB_PULLUP output is connected to PortA bit 3. This allows the DP_HI bit to control the removal/attaching of an external pull-up resistor. The normal port logic and GPIO functions are disabled.
2 VBUSVLD	VBUS valid pin enable bit. The VBUS_VLD pin is an input pin to the USB module that works in conjunction with the VBUD_VLD bit in the OTG_STAT register. 0 VBUS_VLD pin is not connected externally 1 VBUS_VLD pin is connected to PortA bit 2. The normal port logic and GPIO functions are disabled.

**Table 16-39. OTGPIN Field Descriptions (continued)**

<b>Field</b>	<b>Description</b>
1 SESSEND	Session end pin enable bit. The SESS_END pin is an input pin to the USB module that works in conjunction with the SESS_END bit in the OTG_STAT register. 0 SESS_END pin is not connected externally) 1 SESS_END pin is connected to PortA bit 1. The normal port logic and GPIO functions are disabled.
0 SESSVLD	Session valid pin enable bit. The SESS_VLD pin is an input pin to the USB module that works in conjunction with the SESS_VLD bit in the OTG_STAT register. 0 SESS_VLD pin is not connected externally 1 SESS_VLD pin is connected to PortA bit 0. The normal port logic and GPIO functions are disabled.

## 16.6 OTG and Host Mode Operation

The Host Mode logic allows devices such as digital cameras and palmtop computers to function as a USB Host Controller. The OTG logic adds an interface to allow the OTG Host Negotiation and Session Request Protocols (HNP and SRP) to be implemented in software. Host Mode allows a peripheral such as a digital camera to be connected directly to a USB compliant printer. Digital photos can then be easily printed without having to upload them to a PC. In the palmtop computer application, a USB compliant keyboard/mouse can be connected to the palmtop computer with the obvious advantages of easier interaction.

Host mode is intended for use in handheld-portable devices to allow easy connection to simple HID class devices such as printers and keyboards. It is NOT intended to perform the functions of a full OHCI or UHCI compatible host controller found on PC motherboards. The USB-FS is not supported by Windows 98 as a USB host controller. Host mode allows bulk, Isochronous, interrupt and control transfers. Bulk data transfers are performed at nearly the full USB bus bandwidth. Support is provided for ISO transfers, but the number of ISO streams that can be practically supported is affected by the interrupt latency of the processor servicing the token during interrupts from the SIE. Custom drivers must be written to support Host mode operation.

Setting the HOST\_MODE\_EN bit in the CTL register enables host Mode. The USB-FS core can only operate as a peripheral device or in Host Mode. It cannot operate in both modes simultaneously. When HOST\_MODE is enabled, only endpoint zero is used. All other endpoints should be disabled by software.

### 16.6.1 Configuration of External Pull-up/Pull-down for USB

The OTG\_CTL, CTL, and USB\_OTG\_CONTROL register bit values control the external pull-up and pull-down for the USB controller.

To configure pull-down on the DM line from the USB controller in OTG mode, set these bits:

- OTG\_CTL [OTG\_EN] = 1
- OTG\_CTL [DM\_LOW] = 1

To configure pull-down on the DM line from the USB controller in host mode, set these bits:

- OTG\_CTL [OTG\_EN] = 0

- CTL [Host\_Mode\_en] = 1

After the USB controller generates the pull-down on the DM line, use the configuration explained in [Table 16-40](#) to route the pull-down to the external pin.

**Table 16-40. DM Line Pull-down Configuration**

DMDOWN (OTGPIN)	Pull Down on DM from USB Controller	Description
0	x	PTA4 stays as normal I/O. The DM line has no internal pull-down connected. Additionally, you can use this configuration to provide a tri-state on the pin when the DM line uses an external pull-down resistor. Configure the pin as input with pull-up disabled in order to get a tri-state on the pin.
1	0	PTA4 becomes an external pull-down control (PTA4 should output Vdd). The DM line has no internal pull-down connected.
1	1	PTA4 becomes an external pull-down control (PTA4 should output Vss). The DM line has no internal pull-down connected.

To configure pull-down on the DP line from the USB controller in OTG mode, set these bits:

- OTG\_CTL [OTG\_EN] = 1
- OTG\_CTL [DP\_LOW] = 1

To configure pull-down on the DP line from the USB controller in host mode, set these bits:

- OTG\_CTL [OTG\_EN] = 0
- CTL [Host\_Mode\_en] = 1

After the USB controller generates the pull-down on the DP line, use the configuration explained in [Table 16-41](#) to route the pull-down to the external pin.

**Table 16-41. DP Line Pull-down Configuration**

DPPDOWN (OTGPIN)	Pull Down on DP from USB Controller	Description
0	x	PTA5 stays as normal I/O. The DP line has no internal pull-down connected. Additionally, you can use this configuration to provide a tri-state on the pin when the DP line uses an external pull-down resistor. Configure the pin as input with pull-up disabled in order to get a tri-state on the pin.
1	0	PTA5 becomes an external pull-down control (PTA5 should output Vdd). The DP line has no internal pull-down connected.
1	1	PTA5 becomes an external pull-down control (PTA5 should output Vss). The DP line has no internal pull-down connected.

To configure pull-up on the DP line from the USB controller in OTG mode, set these bits:

- OTG\_CTL [OTG\_EN] = 1
- OTG\_CTL [DP\_HIGH] = 1

To configure pull-up on the DP line from the USB controller in device mode, set these bits:

- USB\_OTG\_CONTROL [DPPULLUP\_NONOTG] = 1
- CTL [USB\_EN] = 1
- OTG\_CTL [OTG\_EN] = 0

After the USB controller generates the pull-up on the DP line, use the configuration explained in [Table 16-42](#) to route the pull-up to the internal transceiver or the external pin.

**Table 16-42. DP Line Pull-up Configuration**

PULLUP (OTGPIN)	USBPU (USBTRC0)	Pull Up on DP from USB Controller	Description
0	0	0	PTA3 stays as normal I/O. The DP line has no internal pull-up connected. Additionally, you can use this configuration to provide a tri-state on the pin when the DP line uses an external pull-up resistor. Configure the pin as input with pull-up disabled in order to get a tri-state on the pin.
0	1	1	PTA3 stays as normal I/O. The DP line has an internal pull-up connected
1	0	0	PTA3 becomes an external pull-up control (PTA3 should output Vss). The DP line has no internal pull-up connected.
1	0	1	PTA3 becomes an external pull up-control (PTA3 should output Vdd). The DP line has no internal pull-up connected.

## 16.7 Host Mode Operation Examples

The following sections illustrate the steps required to perform USB host functions using the USB-FS core. The following sections are useful to understand the interaction of the hardware and the software at a detailed level, but an understanding of the interactions at this level is not required to write host applications using the API software.

To enable host mode and discover a connected device:

1. Enable Host Mode (CTL[HOST\_MODE\_EN]=1). Pull down resistors enabled, pull-up disabled. SOF generation begins. SOF counter loaded with 12,000. Eliminate noise on the USB by disabling Start of Frame packet generation by writing the USB enable bit to 0 (CTL[USB\_EN]=0).
2. Enable the ATTACH interrupt (INT\_ENB[ATTACH]=1).
3. Wait for ATTACH interrupt (INT\_STAT[ATTACH]). Signaled by USB Target pull-up resistor changing the state of DPLUS or DMINUS from 0 to 1 (SE0 to J or K state).
4. Check the state of the JSTATE and SE0 bits in the control register. If the JSTATE bit is 0 then the connecting device is low speed. If the connecting device is low speed then set the low speed bit in the address registers (ADDR[LS\_EN]=1) and the host the host without hub bit in endpoint 0 register control (EP\_CTL0[HOST\_WO\_HUB]=1).
5. Enable RESET (CTL[RESET]=1) for 10 ms
6. Enable SOF packet to keep the connected device from going to suspend (CTL[USB\_EN]=1)
7. Start enumeration by sending a sequence of Chapter 9, device frame work packets to the default control pipe of the connected device.

To complete a control transaction to a connected device:

1. Complete all steps discover a connected device
2. Set up the endpoint control register for bidirectional control transfers EP\_CTL0[4:0] = 0x0d.
3. Place a copy of the device framework setup command in a memory buffer. See Chapter 9 of the USB 2.0 specification [2] for information on the device framework command set.
4. Initialize current (even or odd) TX EP0 BDT to transfer the 8 bytes of command data for a device framework command (i.e. a GET DEVICE DESCRIPTOR).
  - Set the BDT command word to 0x00080080 – Byte count to 8, own bit to 1
  - Set the BDT buffer address field to the start address of the 8 byte command buffer
5. Set the USB device address of the target device in the address register (ADDR[6:0]). After the USB bus reset, the device USB address is zero. It is set to some other value (usually 1) by the Set Address device framework command.
6. Write the token register with a SETUP to Endpoint 0 the target device default control pipe (TOKEN=0xD0). This initiates a setup token on the bus followed by a data packet. The device handshake is returned in the BDT PID field after the packets complete. When the BDT is written a token done (INT\_STAT[TOK\_DNE]) interrupt is asserted. This completes the setup phase of the setup transaction as referenced in chapter 9 of the USB specification.
7. To initiate the data phase of the setup transaction (i.e., get the data for the GET DEVICE descriptor command) set up a buffer in memory for the data to be transferred.
8. Initialize the current (even or odd) TX EP0 BDT to transfer the data.
  - Set the BDT command word to 0x004000C0 – Byte count to the length of the data buffer in this case 64, own bit to 1, Data toggle to Data1.
  - Set the BDT buffer address field to the start address of the data buffer
9. Write the token register with a IN or OUT token to Endpoint 0 the target device default control pipe, an IN token for a GET DEVICE DESCRIPTOR command (TOKEN=0x90). This initiates an IN token on the bus followed by a data packet from the device to the host. When the data packet completes the BDT is written and a token done (INT\_STAT[TOK\_DNE]) interrupt is asserted. For control transfers with a single packet data phase this completes the data phase of the setup transaction as referenced in chapter 9 of the USB specification.
10. To initiate the Status phase of the setup transaction set up a buffer in memory to receive or send the zero length status phase data packet.
11. Initialize the current (even or odd) TX EP0 BDT to transfer the status data.
  - Set the BDT command word to 0x00000080 – Byte count to the length of the data buffer in this case 0, own bit to 1, Data toggle to Data0.
  - Set the BDT buffer address field to the start address of the data buffer
12. Write the token register with a IN or OUT token to Endpoint 0 the target device default control pipe, an OUT token for a GET DEVICE DESCRIPTOR command (TOKEN=0x10). This initiates an OUT token on the bus followed by a zero length data packet from the host to the device. When the data packet completes the BDT is written with the handshake form the device and a token done (INT\_STAT[TOK\_DNE]) interrupt is asserted. This completes the data phase of the setup transaction as referenced in chapter 9 of the USB specification.

To send a Full speed bulk data transfer to a target device:

1. Complete all steps discover a connected device and to configure a connected device. Write the ADDR register with the address of the target device. Typically, there is only one other device on the USB bus in host mode so it is expected that the address is 0x01 and should remain constant.
2. Write the ENDPT0 to 0x1D register to enable transmit and receive transfers with handshaking enabled.
3. Setup the Even TX EP0 BDT to transfer up to 64 bytes.
4. Set the USB device address of the target device in the address register (ADDR[6:0]).
5. Write the TOKEN register with an OUT token to the desired endpoint. The write to this register triggers the USB-FS transmit state machines to begin transmitting the TOKEN and the data.
6. Setup the Odd TX EP0 BDT to transfer up to 64 bytes.
7. Write the TOKEN register with an OUT token as in step 4. Two Tokens can be queued at a time to allow the packets to be double buffered to achieve maximum throughput.
8. Wait for the TOK\_DNE interrupt. This indicates one of the BDTs has been released back to the microprocessor and that the transfer has completed. If the target device asserts NAKs, the USB-FS continues to retry the transfer indefinitely without processor intervention unless the RETRY\_DIS retry disable bit is set in the EP0 control register. If the retry disable bit is set, the handshake (ACK, NAK, STALL, or ERROR (0xf)) is returned in the BDT PID field. If a stall interrupt occurs, the pending packet must be dequeued and the error condition in the target device cleared. If a RESET interrupt occurs (SE0 for more than 2.5us), the target has detached.
9. After the TOK\_DNE interrupt occurs, the BDTs can be examined and the next data packet queued by returning to step 2.

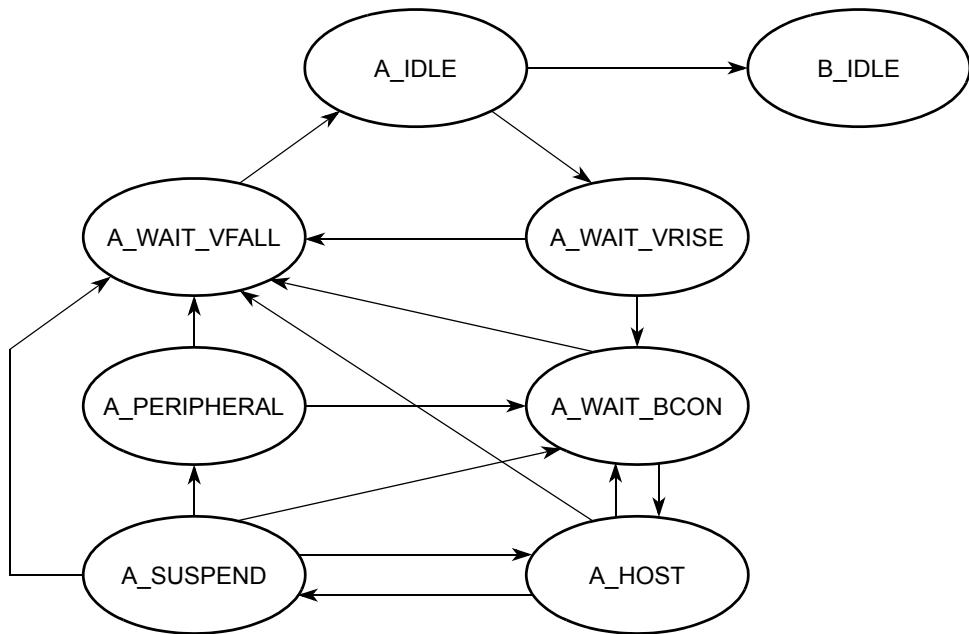
## 16.8 On-The-Go Operation

The USB-OTG core provides sensors and controls to enable On-The-Go (OTG) operation. These sensors are used by the OTG API software to implement the Host Negotiation Protocol (HNP) and Session Request Protocol (SRP). API calls are provided to give access the OTG protocol control signals, and include the OTG capabilities in the device application. The following state machines show the OTG operations involved with HNP and SRP protocols from either end of the USB cable.

### 16.8.1 OTG Dual Role A Device Operation

A device is considered the A device because of the type of cable attached. If the USB Type A connector or the USB Type Mini A connector is plugged into the device, he is considered the A device.

A dual role A device operates as the following flow diagram and state description table illustrates.



**Figure 16-37. Dual Role A Device Flow Diagram**

**Table 16-43. State Descriptions for Figure 16-37**

State	Action	Response
A_IDLE	If ID Interrupt. The cable has been un-plugged or a Type B cable has been attached. The device now acts as a Type B device.	Go to B_IDLE
	If the A application wants to use the bus or if the B device is doing an SRP as indicated by an A_SESS_VLD Interrupt or Attach or Port Status Change Interrupt check data line for 5 –10 msec pulsing.	Go to A_WAIT_VRISE Turn on DRV_VBUS
A_WAIT_VRISE	If ID Interrupt or if A_VBUS_VLD is false after 100 msec The cable has been changed or the A device cannot support the current required from the B device.	Go to A_WAIT_VFALL Turn off DRV_VBUS
	If A_VBUS_VLD interrupt	Go to A_WAIT_BCON
A_WAIT_BCON	After 200 msec without Attach or ID Interrupt. (This could wait forever if desired.)	Go to A_WAIT_FALL Turn off DRV_VBUS
	A_VBUS_VLD Interrupt and B device attaches	Go to A_HOST Turn on Host Mode

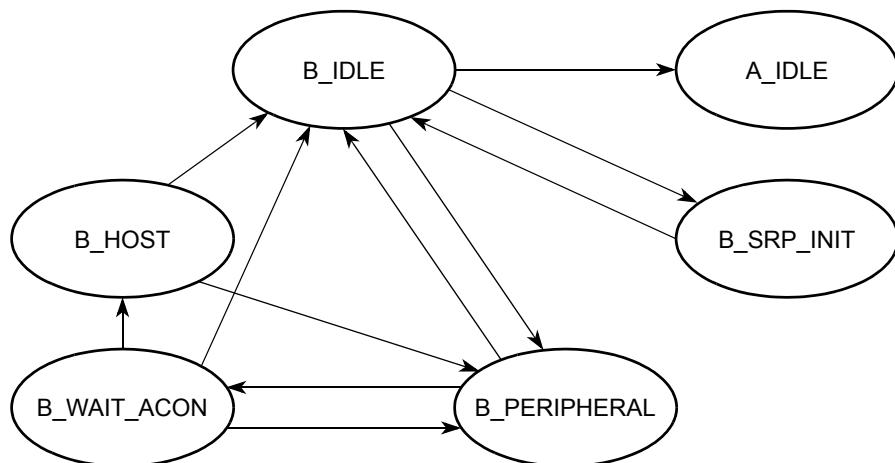
**Table 16-43. State Descriptions for Figure 16-37 (continued)**

State	Action	Response
A_HOST	Enumerate Device determine OTG Support.  If A_VBUS_VLD/ Interrupt or A device is done and doesn't think he wants to do something soon or the B device disconnects	Go to A_WAIT_VFALL Turn off Host Mode Turn off DRV_VBUS
	If the A device is finished with session or if the A device wants to allow the B device to take bus.	Go to A_SUSPEND
	ID Interrupt or the B device disconnects	Go to A_WAIT_BCON
	If ID Interrupt, or if 150 msec B disconnect timeout (This timeout value could be longer) or if A_VBUS_VLD\ Interrupt	Go to A_WAIT_VFALL Turn off DRV_VBUS
A_SUSPEND	If HNP enabled, and B disconnects in 150 msec then B device is becoming the host.	Go to A_PERIPHERAL Turn off Host Mode
	If A wants to start another session	Go to A_HOST
	If ID Interrupt or if A_VBUS_VLD interrupt	Go to A_WAIT_VFALL Turn off DRV_VBUS.
A_PERIPHERAL	If 3 –200 msec of Bus Idle	Go to A_WAIT_BCON Turn on Host Mode
	If ID Interrupt or (A_SESS_VLD/ & b_conn/)	Go to A_IDLE

### **16.8.2 OTG Dual Role B Device Operation**

A device is considered a B device if it connected to the bus with a USB Type B cable or a USB Type Mini B cable.

A dual role B device operates as the following flow diagram and state description table illustrates.



**Figure 16-38. Dual Role B Device Flow Diagram**

**Table 16-44. State Descriptions for Figure 16-38**

<b>State</b>	<b>Action</b>	<b>Response</b>
B_IDLE	If ID\ Interrupt. A Type A cable has been plugged in and the device should now respond as a Type A device.	Go to A_IDLE
	If B_SESS_VLD Interrupt. The A device has turned on VBUS and begins a session.	Go to B_PERIPHERAL Turn on DP_HIGH
	If B application wants the bus and Bus is Idle for 2 ms and the B_SESS_END bit is set, the B device can perform an SRP.	Go to B_SRPI_INIT Pulse CHRG_VBUS Pulse DP_HIGH 5-10 ms
B_SRPI_INIT	If ID\ Interrupt or SRP Done (SRP must be done in less than 100 msecs.)	Go to B_IDLE
B_PERIPHERAL	If HNP enabled and the bus is suspended and B wants the bus, the B device can become the host.	Go to B_WAIT_ACON Turn off DP_HIGH
B_WAIT_ACON	If A connects, an attach interrupt is received	Go to B_HOST Turn on Host Mode
	If ID\ Interrupt or B_SESS_VLD/ Interrupt If the cable changes or if VBUS goes away, the host doesn't support us. Go to B_IDLE	Go to B_IDLE
	If 3.125 ms expires or if a Resume occurs	Go to B_PERIPHERAL
B_HOST	If ID\ Interrupt or B_SESS_VLD\ Interrupt If the cable changes or if VBUS goes away, the host doesn't support us.	Go to B_IDLE
	If B application is done or A disconnects	Go to B_PERIPHERAL

### 16.8.3 Power

The USB-FS core is a fully synchronous static design. The power used by the design is dependant on the application usage of the core. Applications that transfer more data or cause a greater number of packets to be sent consumes a greater amount of power.

Because the design is synchronous and static, reducing the transitions on the clock net may conserve power. This may be done in the following ways.

The first is to reduce the clock frequency to the USB module. The clock frequency may not be reduced below the minimum recommended operating frequency of the USB module without first disabling the USB operation and disconnecting (via software disconnect) the USB module from the USB bus.

Alternately, the clock may be shut off to the core to conserve power. Again, this may only be done after the USB operations on the bus have been disabled and the device has been disconnected from the USB.

## 16.8.4 USB Suspend State

USB bus powered devices are required to respond to a 3ms lack of activity on the USB bus by going into a suspend state. Software is notified of the suspend condition via the transition in the port status and control register. Optionally, an interrupt can be generated that is controlled by the interrupt enable register. In the suspend state, a USB device has a maximum USB bus power budget of 500uA. To achieve that level of power conservation, most of the device circuits need to be switched off. When the clock is disabled to the USB-FScore all functions are disabled, but all operational states are retained. The transceiver VP and VM signals can be used to construct a circuit able to detect the resume signaling on the bus and restore the clocks to the rest of the circuit when the USB host takes the bus out of the suspend state.

# **Chapter 17**

## **Real-Time Counter (S08RTCV1)**

### **17.1 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 wake up from low power modes without the need of external components.

#### **NOTE**

For details on low-power mode operation, refer to [Table 3-2](#) in [Chapter 3](#), “[Modes of Operation](#)”.

## 17.1.1 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)

## 17.1.2 Modes of Operation

This section defines the operation in stop, wait and background debug modes.

### 17.1.2.1 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.

### 17.1.2.2 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.

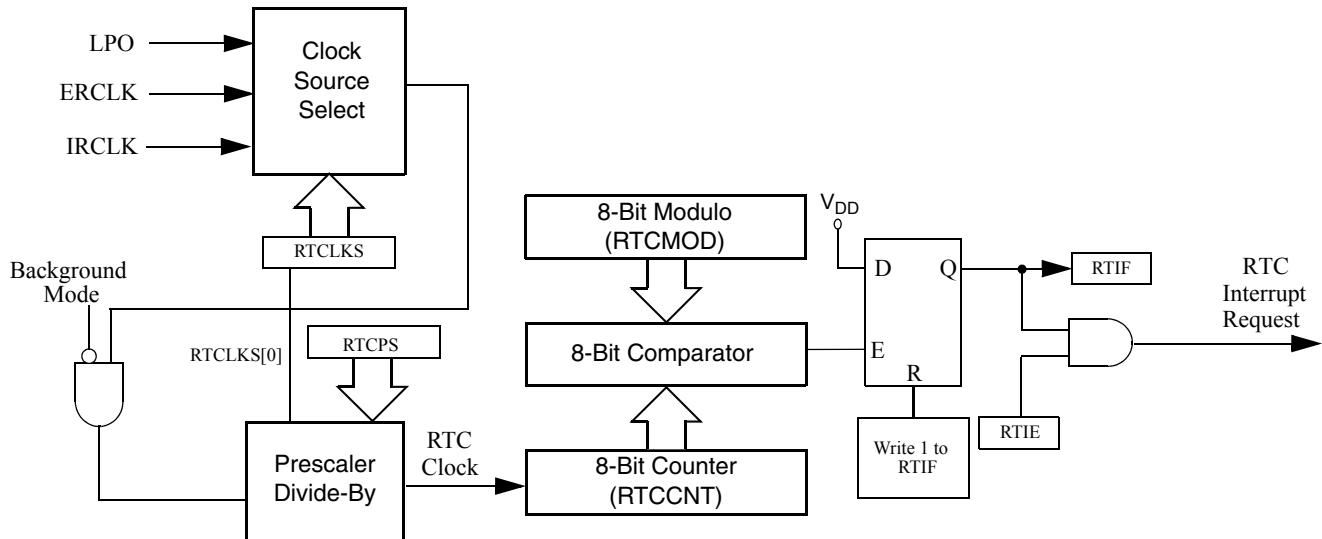
Power consumption is lower when all clock sources are disabled, but in that case, the real-time interrupt cannot wake up the MCU from stop modes.

### 17.1.2.3 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.

### 17.1.3 Block Diagram

The block diagram for the RTC module is shown in [Figure 17-1](#).



**Figure 17-1. Real-Time Counter (RTC) Block Diagram**

## 17.2 External Signal Description

The RTC does not include any off-chip signals.

## 17.3 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 the memory section of this document 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.

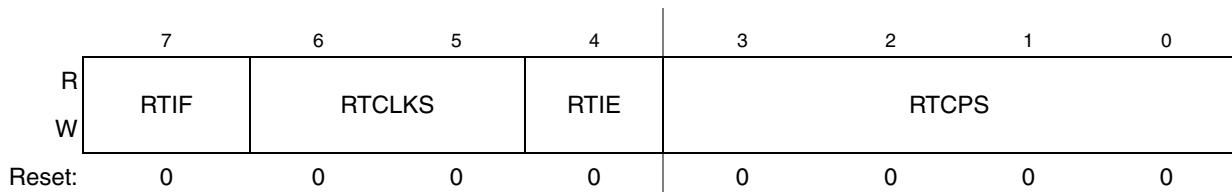
[Table 17-1](#) is a summary of RTC registers.

**Table 17-1. 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														

### 17.3.1 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 17-2. RTC Status and Control Register (RTCSC)**

**Table 17-2. RTCSC Field Descriptions**

Field	Description
7 RTIF	Real-Time Interrupt Flag 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	Real-Time Clock Source Select. 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	Real-Time Interrupt Enable. 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	Real-Time Clock Prescaler Select. These four read/write bits select binary-based or decimal-based divide-by values for the clock source. See <a href="#">Table 17-3</a> . Changing the prescaler value clears the prescaler and RTCCNT counters. Reset clears RTCPS.

**Table 17-3. 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	$2^{10}$	$2^{11}$	$2^{12}$	$2^{13}$	$2^{14}$	$2^{15}$	$2^{16}$	$10^3$	$2 \times 10^3$	$5 \times 10^3$	$10^4$	$2 \times 10^4$	$5 \times 10^4$	$10^5$	$2 \times 10^5$

### 17.3.2 RTC Counter Register (RTCCNT)

RTCCNT is the read-only value of the current RTC count of the 8-bit counter.

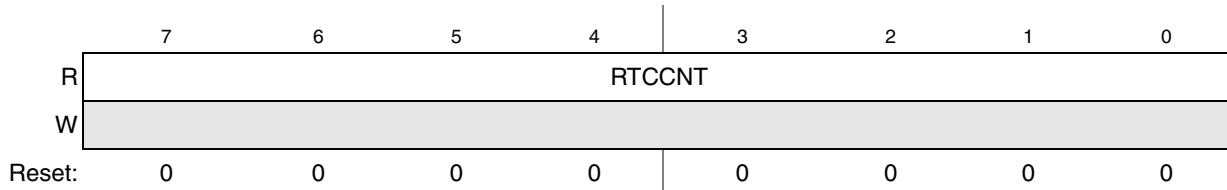


Figure 17-3. RTC Counter Register (RTCCNT)

Table 17-4. RTCCNT Field Descriptions

Field	Description
7:0 RTCCNT	RTC Count. 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.

### 17.3.3 RTC Modulo Register (RTCMOD)

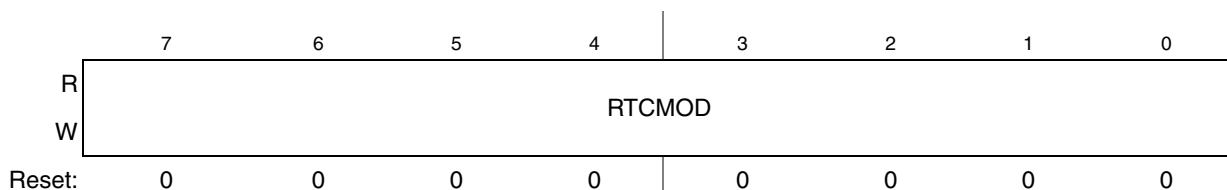


Figure 17-4. RTC Modulo Register (RTCMOD)

Table 17-5. RTCMOD Field Descriptions

Field	Description
7:0 RTCMOD	RTC Modulo. 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.

## 17.4 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 17-6](#) shows different prescaler period values.

**Table 17-6. Prescaler Period**

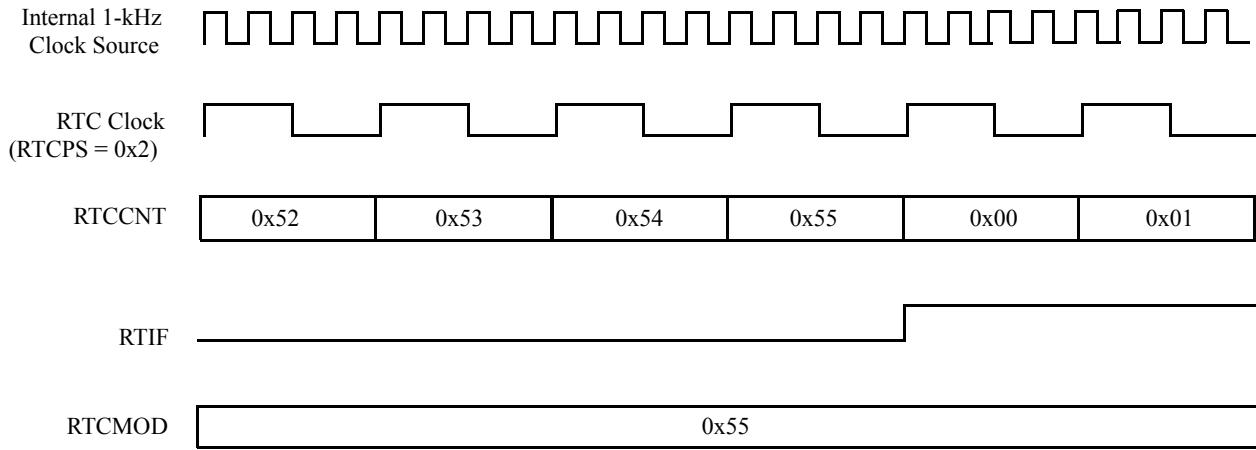
RTCPS	1-kHz Internal Clock (RTCLKS = 00)	1-MHz External Clock (RTCLKS = 01)	32-kHz Internal Clock (RTCLKS = 10)	32-kHz Internal Clock (RTCLKS = 11)
0000	Off	Off	Off	Off
0001	8 ms	1.024 ms	250 µs	32 ms
0010	32 ms	2.048 ms	1 ms	64 ms
0011	64 ms	4.096 ms	2 ms	128 ms
0100	128 ms	8.192 ms	4 ms	256 ms
0101	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.

### 17.4.1 RTC Operation Example

This section shows an example of the RTC operation as the counter reaches a matching value from the modulo register.

**Figure 17-5. RTC Counter Overflow Example**

In the example of [Figure 17-5](#), the selected clock source is the internal clock source. The prescaler (RTCPS) is set to 0x2 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.

'b00the clock ofthe clock of flip-flop is

## 17.5 Initialization/Application Information

This section provides example code to give some basic direction to a user 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)
```

## Real-Time Counter (S08RTCV1)

```
{  
    /* 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 18

## Cryptographic Acceleration Unit (CAU)

This chapter describes the Cryptographic Acceleration Unit (CAU) programming model. The CAU is an instruction level coprocessor accessed with ColdFire coprocessor instructions. The CAU supports acceleration of the following cryptographic algorithms: DES, 3DES, AES, MD5 and SHA-1.

### NOTE

To enhance readability, this chapter shows the CAU as coprocessor 0.  
Future implementations could have the CAU designated as coprocessor 1.

### 18.1 CAU Registers

The CAU register file consists of eight, 32-bit registers as shown in [Table 18-1](#). All registers can be read with the coprocessor store instruction (cp0st.l) and written with the coprocessor load instruction (cp0ld.l). However, only bits 0-1 of the CASR are writable. Bits 2-27 of CASR loads should be 0 for compatibility with future versions of the CAU. The CAU only supports long word accesses and register codes 0x8-0xF are reserved.

**Table 18-1. CAU Register File**

Code	Name	Description	DES	AES	SHA-1	MD5
0	CASR	status register	—	—	—	—
1	CAA	accumulator	—	—	T	a
2	CA0	general purpose 0	C	W0	A	—
3	CA1	general purpose 1	D	W1	B	b
4	CA2	general purpose 2	L	W2	C	c
5	CA3	general purpose 3	R	W3	D	d
6	CA4	general purpose 4	—	—	E	—
7	CA5	general purpose 5	—	—	W	—

#### 18.1.1 CAU Status Register

The status register (CASR) contains all of the status and configuration for the CAU. It has three defined fields and 26 bits reserved as shown in [Figure 18-1](#).

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



= Read Only or Reserved

**Figure 18-1. Status Register (CASR)**

IC — Illegal Command

1 = Illegal coprocessor command issued

0 = No illegal commands issued

DPE — DES Parity Error

1 = DES key parity error detected

0 = No error detected

VER — CAU Version

Indicates CAU version; only version 1 is defined.

## 18.2 CAU Operation

The cp0ld.l instruction is used to write to CAU registers and specify CAU operations. Operand 1 of the instruction is the source operand (if any) and the CAU destination register is encoded in the CMD field. All CAU load instruction commands have an execution time specifier of 0.

The cp0st.l instruction is used to read CAU registers. The CAU source register is encoded in the CMD. The CAU only supports long word stores. The CAU store instruction command has an execution time specifier of 0.

## 18.3 CAU Commands

The CAU supports 22 commands as shown in [Table 18-2](#) and described in the following sections, (see section [Section 18.4, “CAU Equate Values”](#) for assembly constant definitions). All other encodings are reserved. The IC bit in the CASR is set if any command is issued that is not defined in the encodings described in this section. A specific illegal command (ILL) is defined to allow for software self checking. Reserved commands should not be issued to ensure compatibility with future implementations.

### NOTE

The value CAX is any CAU register (CASR, CAA, CA0-CA5).

**Table 18-2. CAU Commands**

<b>Inst Type</b>	<b>Command Name</b>	<b>Description</b>	<b>CMD[8:4]</b>	<b>CMD[3:0]</b>	<b>Operation</b>
cp0ld.I	CNOP	No Operation	0x00	0x0	---
cp0ld.I	LDR	Load Reg	0x01	CAx	Op1 -> CAx
cp0st.I	STR	Store Reg	0x02	CAx	CAx -> Destination
cp0ld.I	ADR	Add	0x03	CAx	CAx + Op1 -> CAx
cp0ld.I	RADR	Reverse and Add	0x04	CAx	CAx + ByteRev(Op1) -> CAx
cp0ld.I	ADRA	Add Reg to Acc	0x05	CAx	CAx + CAA -> CAA
cp0ld.I	XOR	Exclusive Or	0x06	CAx	CAx ^ Op1 -> CAx
cp0ld.I	ROTL	Rotate Left	0x07	CAx	CAx <<< Op1 -> CAx
cp0ld.I	MVRA	Move Reg to Acc	0x08	CAx	CAx -> CAA
cp0ld.I	MVAR	Move Acc to Reg	0x09	CAx	CAA -> CAx
cp0ld.I	AESS	AES Sub Bytes	0x0A	CAx	SubBytes(CAx) -> CAx
cp0ld.I	AESIS	AES Inv Sub Bytes	0x0B	CAx	InvSubBytes(CAx) -> CAx
cp0ld.I	AESC	AES Column Op	0x0C	CAx	MixColumns(CAx)^Op1 -> CAx
cp0ld.I	AESIC	AES Inv Column Op	0x0D	CAx	InvMixColumns(CAx^Op1) -> CAx
cp0ld.I	AESR	AES Shift Rows	0x0E	0x0	ShiftRows(CA0-CA3) -> CA0-CA3
cp0ld.I	AESIR	AES Inv Shift Rows	0x0F	0x0	InvShiftRows(CA0-CA3) -> CA0-CA3
cp0ld.I	DESR	DES Round	0x10	IP FP KS[1:0]	DES Round(CA0-CA3)->CA0-CA3
cp0ld.I	DESK	DES Key Setup	0x11	0 0 CP DC	DES Key Op(CA0-CA1)->CA0-CA1 Key Parity Error & CP -> CASR[1]
cp0ld.I	HASH	Hash Function	0x12	0 HF[2:0]	Hash Func(CA1-CA3)+CAA->CAA
cp0ld.I	SHS	Secure Hash Shift	0x13	0x0	CAA <<< 5 -> CAA, CAA->CA0, CA0->CA1, CA1 <<< 30 -> CA2, CA2->CA3, CA3->CA4
cp0ld.I	MDS	Message Digest Shift	0x14	0x0	CA3->CAA, CAA->CA1, CA1->CA2, CA2->CA3,
cp0ld.I	ILL	Illegal Command	0x1F	0x0	0x1->CASR[0]

### 18.3.1 CNOP - coprocessor no operation

**cp0ld.1 #CNOP**

The CNOP command is the coprocessor cp0nop instruction defined for synchronization.

### 18.3.2 LDR - load register

**cp0ld.1 <ea>, #LDR+CAx**

The LDR command loads CAx with the source data specified by <ea>.

### 18.3.3 STR - store register

**cp0st.1 <ea>, #STR+CAx**

The STR command stores the value from CAx to the destination specified by <ea>.

### 18.3.4 ADR - add to register

`cp0ld.1 <ea>, #ADR+CAx`

The ADR command adds the source operand specified by `<ea>` to CAx and stores the result in CAx.

### 18.3.5 RADR - reverse and add to register

`cp0ld.1 <ea>, #RADR+CAx`

The RADR command does a byte reverse on the source operand specified by `<ea>`, adds that value to CAx and stores the result in CAx. An example is shown in [Table 18-3](#).

**Table 18-3. RADR Command Example**

Operand	CAx Before	CAx After
01020304	A0B0C0D0	A4B3C2D1

### 18.3.6 ADRA - add register to accumulator

`cp0ld.1 #ADRA+CAx`

The ADRA command adds CAx to CAA and stores the result in CAA.

### 18.3.7 XOR - exclusive or

`cp0ld.1 <ea>, #XOR+CAx`

The XOR command does an exclusive or of the source operand specified by `<ea>` with CAx and stores the result in CAx.

### 18.3.8 ROTL - rotate left

`cp0ld.1 <ea>, #ROTL+CAx`

The ROTL rotates the bits of CAx to the left with the result stored back to CAx. The number of bits to rotate is the value specified by `<ea>` modulo 32.

### 18.3.9 MVRA - move register to accumulator

`cp0ld.1 #MVRA+CAx`

The MVRA moves the value from the source register CAx to the destination register CAA.

### 18.3.10 MVAR - move accumulator to register

`cp0ld.1 #MVAR+CAx`

The MVAR command moves the value from source register CAA to the destination register CAx.

### 18.3.11 AESSION - AES substitution

**cp0ld.1 #AESSION+CAx**

The AESSION command performs the AES byte substitution operation on CAx and stores the result back to CAx.

### 18.3.12 AESIS - AES inverse substitution

**cp0ld.1 #AESIS+CAx**

The AESIS command performs the AES inverse byte substitution operation on CAx and stores the result back to CAx.

### 18.3.13 AESC - AES column operation

**cp0ld.1 <ea>, #AESC+CAx**

The AESC command performs the AES columns operation on the contents of CAx then performs an exclusive or of that result with the source operand specified by <ea> and stores the result in CAx.

### 18.3.14 AESIC - AES inverse column operation

**cp0ld.1 <ea>, #AESIC+CAx**

The AESIC command performs an exclusive or operation of the source operand specified by <ea> on the contents of CAx followed by the AES inverse mix columns operation on that result and stores the result back in CAx.

### 18.3.15 AESR - AES shift rows

**cp0ld.1 #AESR**

The AESR command performs the AES shift rows operation on registers CA0, CA1, CA2 and CA3. An example is shown in [Table 18-4](#).

**Table 18-4. AESR Command Example**

Register	Before	After
CA0	01020304	01060B00
CA1	05060708	050A0F04
CA2	090A0B0C	090E0308
CA3	0D0E0F00	0D02070C

### 18.3.16 AESIR - AES inverse shift rows

**cp0ld.1 #AESIR**

The AESIR command performs the AES inverse shift rows operation on registers CA0, CA1, CA2 and CA3. An example is shown in [Table 18-5](#).

**Table 18-5. AESIR Command Example**

Register	Before	After
CA0	01060B00	01020304
CA1	050A0F04	05060708
CA2	090E0308	090A0B0C
CA3	0D02070C	0D0E0F00

### 18.3.17 DESR - DES round

**cp01d.1 #DESR+{IP}+{FP}+{KSx}**

The DESR command performs a round of the DES algorithm and a key schedule update with the following source and destination designations: CA0=C, CA1=D, CA2=L, CA3=R. If the IP bit is set then the DES initial permutation is performed on CA2 and CA3 before the round operation. If the FP bit is set then the DES final permutation (inverse initial permutation) is performed on CA2 and CA3 after the round operation. The round operation uses the source values from registers CA0 and CA1 for the key addition operation. The KSx field specifies the shift to use for the key schedule operation used to update the values in CA0 and CA1. The specific shift function performed is based on the KSx field as defined in [Table 18-6](#).

**Table 18-6. Key Shift Function Codes**

KSx Code	KSx Define	Shift Function
0	KSL1	Left 1
1	KSL2	Left 2
2	KSR1	Right 1
3	KSR2	Right 2

### 18.3.18 DESK - DES key setup

**cp01d.1 #DESK+{CP}+{DC}**

The DESK command performs the initial key transformation (permuted choice 1) defined by the DES algorithm on CA0 and CA1 with CA0 containing bits 1-32 of the key and CA1 containing bits 33-64 of the key<sup>1</sup>. If the DC bit is set then no shift operation is performed and the values C<sub>0</sub> and D<sub>0</sub> are stored back to CA0 and CA1 respectively. The DC bit should be set for decrypt operations. If the DC bit is not set then a left shift by 1 is also performed and the values C<sub>1</sub> and D<sub>1</sub> are stored back to CA0 and CA1 respectively. The DC bit should be 0 for encrypt operations. If the CP bit is set and a key parity error is detected then the DPE bit of the CASR is set, otherwise it is cleared.

### 18.3.19 HASH - hash function

**cp01d.1 #HASH+HFx**

The HASH command performs a hashing operation on CA1, CA2 and CA3 and adds that result to the value in CAA and stores the result in CAA. The specific hash function performed is based on the HFx field as defined in [Table 18-7](#).

1. The DES algorithm numbers the most significant bit of a block as bit1 and the least significant as bit 64.

**Table 18-7. Hash Function Codes**

HFx Code	HFx Define	Hash Function	Hash Logic
0	HFF	MD5 F()	CA1&CA2   ~CA1&CA3
1	HFG	MD5 G()	CA1&CA3   CA2&~CA3
2	HFH	MD5 H(), SHA Parity()	CA1^CA2^CA3
3	HFI	MD5 I()	CA2^(CA1 ~CA3)
4	HFC	SHA Ch()	CA1&CA2 ^ ~CA1&CA3
5	HFM	SHA Maj()	CA1&CA2 ^ CA1&CA3 ^ CA2&CA3

### 18.3.20 SHS - secure hash shift

`cp01d.1 #SHS`

The SHS command does a set of register to register move and shift operations in parallel that is useful for implementing SHA-1. The following source and destination assignments are made: CAA=CAA<<<5, CA0=CAA, CA1=CA0, CA2=CA1<<<30, CA3=CA2, CA4=CA3.

### 18.3.21 MDS - message digest shift

`cp01d.1 #MDS`

The MDS command does a set of register to register move operations in parallel that is useful for implementing MD5. The following source and destination assignments are made: CAA=CA3, CA1=CAA, CA2=CA1, CA3=CA2.

### 18.3.22 ILL - illegal command

`cp01d.1 #ILL`

The ILL command is a specific illegal command that sets the IC bit in the CASR. All undefined commands are reserved for use in future implementations.

## 18.4 CAU Equate Values

```
; CAU Registers (CAx)
.set    CASR,0x0
.set    CAA,0x1
.set    CA0,0x2
.set    CA1,0x3
.set    CA2,0x4
.set    CA3,0x5
.set    CA4,0x6
.set    CA5,0x7

; CAU Commands
.set    CNOP,0x000
.set    LDR,0x010
.set    STR,0x020
.set    ADR,0x030
```

## Cryptographic Acceleration Unit (CAU)

```
.set      RADR, 0x040
.set      ADRA, 0x050
.set      XOR, 0x060
.set      ROTL, 0x070
.set      MVRA, 0x080
.set      MVAR, 0x090
.set      AESSION, 0x0A0
.set      AESIS, 0x0B0
.set      AESC, 0x0C0
.set      AESIC, 0x0D0
.set      AESR, 0x0E0
.set      AESIR, 0x0F0
.set      DESR, 0x100
.set      DESK, 0x110
.set      HASH, 0x120
.set      SHS, 0x130
.set      MDS, 0x140
.set      ILL, 0x1F0

; DESR Fields
.set      IP, 0x08          ; initial permutation
.set      FP, 0x04          ; final permutation
.set      KSL1, 0x00         ; key schedule left 1 bit
.set      KSL2, 0x01         ; key schedule left 2 bits
.set      KSR1, 0x02         ; key schedule right 1 bit
.set      KSR2, 0x03         ; key schedule right 2 bits

; DESK Field
.set      DC, 0x01          ; decrypt key schedule
.set      CP, 0x02          ; check parity

; HASH Functions Codes
.set      HFF, 0x0           ; MD5 F() CA1&CA2 | ~CA1&CA3
.set      HFG, 0x1           ; MD5 G() CA1&CA3 | CA2&~CA3
.set      HFH, 0x2           ; MD5 H(), SHA Parity() CA1^CA2^CA3
.set      HFI, 0x3           ; MD5 I() CA2^(CA1|~CA3)
.set      HFC, 0x4           ; SHA Ch() CA1&CA2 ^ ~CA1&CA3
.set      HFM, 0x5           ; SHA Maj() CA1&CA2 ^ CA1&CA3 ^ CA2&CA3
```

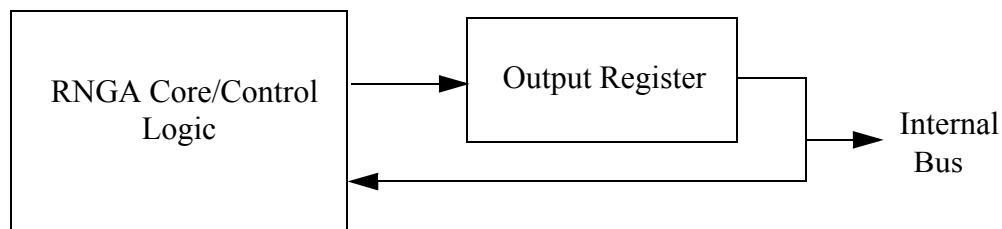
# Chapter 19

## Random Number Generator Accelerator (RNGA)

### 19.1 Overview

The RNGA (Random Number Generator Accelerator) module is a digital integrated circuit capable of generating 32-bit random numbers. It is designed to comply with FIPS-140 standards for randomness and non-determinism. The random bits are generated by clocking shift registers with clocks derived from ring oscillators. The configuration of the shift registers ensures statistically good data (i.e. data that looks random). The oscillators with their unknown frequencies provide the required entropy needed to create random data.

A top level block diagram of the RNGA is shown in [Figure 19-1](#). The module connects to the IP Bus defined in SRS version 3.1.1 IP Interface Specification.



**Figure 19-1. RNGA Block Diagram**

There is no known cryptographic proof showing that this is a secure method of generating random data. In fact, there may be an attack against the random number generator described in this document if its output is used directly in a cryptographic application (the attack is based on the linearity of the internal shift registers). In light of this, it is *highly* recommended that the random data produced by this module be used as an input seed to a NIST approved (based on DES or SHA-1) or cryptographically secure (RSA Generator or BBS Generator) random number generation algorithm. It is also recommended that other sources of entropy be used along with the RNGA to generate the seed to the pseudorandom algorithm. The more random sources combined to create the seed the better. The following is a list of sources that could be easily combined with the output of this module.

- Current time using highest precision possible.
- Mouse and keyboard motions (or equivalent if being used on a cell phone or PDA).
- Other entropy supplied directly by the user.

## 19.2 Features

The RNGA includes these distinctive features:

- 32-bit interface
- 32-bit Output Register
- secure mode
- power saving mode

## 19.3 Modes of Operation

For details on low-power mode operation, refer to [Table 3-2](#) in [Chapter 3, “Modes of Operation”](#).

Although the RNGA has several modes, only one is intended for use during normal operation. The other modes were created to aid in verification and testability of the module. The Normal and Oscillator Frequency Test Modes are entered by setting the appropriate bits in the RNGA Mode register. The Sleep Mode is entered by setting the appropriate bit in the RNGA Control register. These registers are described in more detail in [Section 19.4.1, “Register Descriptions](#).

- Normal Mode

In this mode the RNGA generates random data. Because this is the default mode of operation, the user is not required to change the mode before requesting random data. This is also the only valid mode when in the secure state. While in this mode, the internal shift registers are driven by internally generated clocks with unknown frequency. Depending on the internal state of the RNGA, these clocks are derived from the RNGA’s oscillators or a deterministic clock (based on the system clock). For simplicity sake, throughout the rest of this document these clocks are referred to as the oscillator clocks.

- Secure Mode

In this mode the RNGA is forced into the Normal mode described above. Secure Mode is equivalent to the condition where the RNGA is in Normal mode and is unable to exit that mode. The low power Sleep Mode can be entered while in the Secure Mode.

- Verification Mode

This mode is provided for verification and testing of the module. While in this mode, the random output is generated by a counter rather than the usual shift registers. The deterministic result allows for easy verification of the surrounding RNGA control logic.

- Oscillator Frequency Test Mode

This mode is provided for testing of the RNGA’s ring oscillators. While in this mode, the shift registers are clocked exclusively by the oscillator clocks (this may not be the case in the Normal Mode) allowing the oscillator frequency counters (described in [Section 19.4.1, “Register Descriptions](#)) to accurately count the pulses received from the oscillator clocks during a given amount of time.

- Sleep Mode

In this mode the RNGA's oscillator clocks are shut off. The mode is entered by writing to the Sleep bit in the Control Register. When in this mode, the Output Register is not loaded.

- Scan Mode

In this mode the RNGA reconfigures much of its untestable logic so it is testable by scan. The mode is entered by driving the block input `ipt_mode` active. This mode should only be used when scan is used to test the module.

In all the low power modes, the RNGA goes into the sleep mode and the oscillator clocks are shut off. It can't wake up the core and would wake up from any of the low power mode along with the core only.

These are high level descriptions only, detailed descriptions of operating modes are contained in later sections.

## 19.4 Memory Map/Register Definition

The RNGA registers are summarized in [Table 19-1](#). The following subsections describe each addressable register in more detail.

Accessing any RNGA register takes four (4) clock cycles.

**Table 19-1. RNGA Registers**

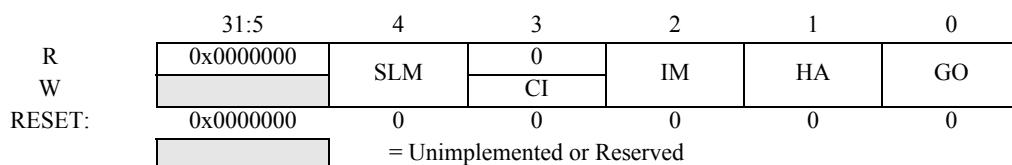
Register	Access
Control Register (RNGCR)	R/W
Status Register (RNGSR)	R
Entropy Register (RNGER)	W
Output Register (RNGOUT)	R

### 19.4.1 Register Descriptions

This section consists of register descriptions in address order. All RNGA\_32IP registers are 32-bit access only.

#### 19.4.1.1 Control Register (RNGCR)

The fields in RNGCR are defined in [Figure 19-2](#). Immediately after reset the RNGA begins generating entropy in its internal shift registers. Random data is not pushed to the RNGOUT until RNGCR[GO] is set. After this, a random 32-bit word is pushed to RNGOUT every 256 cycles. If RNGOUT is full, no push occurs. In this way RNGOUT is kept as full as possible.



**Figure 19-2. Control Register (RNGCR)**

**GO**

The Go Bit must be set before the RNGA begins loading data into RNGOUT. This bit is sticky and can only be cleared by a hardware reset or by changing to Secure Mode. Setting the GO bit does not bring the RNGA out of Sleep Mode. Furthermore, the Go bit does not need to be reset after exiting Sleep Mode.

- 1 = RNGOUT is loaded with random data.
- 0 = RNGOUT is not loaded with random data.

**HA (High Assurance)**

While this bit is high, the RNGA notifies the SCC if a security violation occurs (i.e. RNGOUT is read while empty). This bit enables RNGSR[SV] as well as the output `rnga_scc_debug` port. This bit is sticky and can only be cleared through a hardware reset.

- 1 = Enable notification of security violations.
- 0 = Disable notification of security violations.

**IM (Interrupt Mask)**

This bit masks the error interrupt, `ipi_error_int`.

- 1 = Interrupt `ipi_error_int` is masked.
- 0 = Interrupt `ipi_error_int` is enabled.

**CI (Clear Interrupt)**

Writing a one to this bit clears the error interrupt as well as the error status bit in the Status Register. The bit is self clearing.

- 1 = Clear interrupt `ipi_error_int`.
- 0 = Do not clear interrupt `ipi_error_int`.

**SLM (Sleep Mode)**

The RNGA can be placed in low power mode by asserting the module input `ipg_doze` or by setting this Sleep bit. If either of these conditions are met, the oscillators are disabled. De-asserting `ipg_doze` and resetting the Sleep Bit causes the RNGA to exit Sleep Mode. RNGOUT is not pushed while the RNGA is in Sleep Mode.

- 1 = RNGA is in Sleep Mode.
- 0 = RNGA is not in Sleep Mode.

**19.4.1.2 Status Register (RNGSR)**

The Status Register (RNGSR), shown in [Figure 19-3](#), is a read-only register that reflects the internal status of the RNGA. Only 32-bit reads of this register are supported.

	31	30:24	23-16	15:8	7:5	4	3	2	1	0
R	OD	0x00	ORS	ORL	0x0	SLP	EI	OUF	LRS	SV
W										
RESET:	0	0x00	0x01	0x00	0x0	0	0	0	0	0

= Unimplemented or Reserved

**Figure 19-3. Status Register (RNGSR)**

## SV (Security Violation)

When enabled by RNGCR[HA], this bit signals that a security violation has occurred. Currently, RNGOUT underflow is the only condition considered a security violation. The bit is sticky and can only be cleared by a hardware reset. The output `rnga_scc_debug` is driven off this bit.

1 = A security violation has occurred.

0 = No security violations have occurred or RNGCR[HA] is not set.

## LRS (Last Read Status)

This bit is always enabled and reflects the status of the most recent read of RNGOUT.

1 = Last read was performed while RNGOUT was empty (underflow condition).

0 = Last read was performed while RNGOUT was not empty.

## OUF (Output Register Underflow)

This bit is always enabled and signals a RNGOUT underflow condition. The bit is reset by reading RNGSR.

1 = RNGOUT has been read while empty since the last read of RNGSR.

0 = RNGOUT has not been read while empty since the last read of RNGSR.

## EI (Error Interrupt)

This bit is always enabled and signals a RNGOUT underflow condition. This bit is different from the previous two bits in that it is only reset by a write to RNGCR[CI]. This bit is not masked by RNGCR[IM].

1 = RNGOUT has been read while empty.

0 = RNGOUT has not been read while empty.

## SLP (Sleep)

This bit reflects whether the RNGA is in Sleep mode (RNGCR[SLM] is set or the `ipg_doze` input is asserted). When this bit is set, the RNGA is in Sleep Mode and the oscillator clocks are inactive. While in this mode, RNGOUT is not loaded and RNGSR[ORL] does not increase.

1 = The RNGA is in Sleep Mode.

0 = The RNGA is not in Sleep Mode.

## ORL (Output Register Level)

Signals how many random words are currently resident in RNGOUT. The bits should be interpreted as an integer (the value 0b00001001 = signals that 0x09 random words are in RNGOUT).

For this device, the maximum value of ORL is 1.

## ORS (Output Register Size)

Signals the actual size of RNGOUT. In other words, this is the maximum possible value for RNGSR[ORL]. The bits should be interpreted as an integer.

For this device, the maximum value of ORS is 1.

## OD (Oscillator Dead)

Indicates that at least one of the shift registers is stuck in its reset state. This information can be used to determine whether the oscillator clocks are operational (i.e. not dead or broken).

- 1 = At least one oscillator is broken
- 0 = Both oscillators are operational

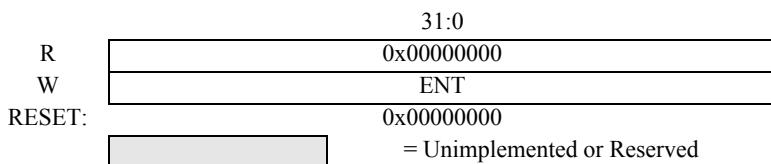
#### NOTE

This bit is intended to be used for silicon (production) test purposes.

### 19.4.1.3 Entropy Register (RNGER)

RNGER is a write-only register that allows the user to insert entropy into the RNGA. This register allows an external user to continually seed the RNGA with externally generated random data. Although the use of this register is recommended, it is also optional. RNGER can be written at any time during operation and cannot be written too quickly.

Each time RNGER is written, the written value is used to update the internal state of the RNGA. The update is performed in such a way that the entropy in the RNGA's internal state is preserved. Use of RNGER can increase the entropy but never decrease it.



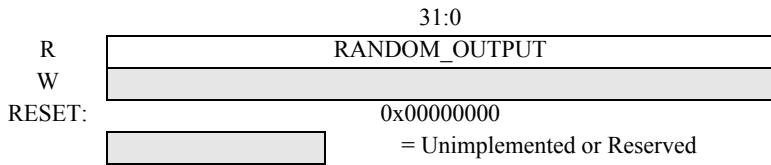
**Figure 19-4. Entropy Register (RNGER)**

#### External Entropy (ENT)

The bits in this field are used to update the internal state of the RNGA.

### 19.4.1.4 Output Register (RNGOUT)

RNGOUT provides temporary storage for random data generated by the RNGA. As long as RNGOUT is not empty, a read of this register returns 32 bits of random data. If RNGOUT is read when it is empty, the RNGSR[EI], RNGSR[OUF], and RNGSR[LRS] are set. If the interrupt is enabled in the RNGCR, ipi\_error\_int is also driven active. If RNGCR[HA] is set, then rnga\_scc\_debug is also driven high. RNGSR[ORL], described in [Section 19.4.1.2, “Status Register \(RNGSR\)”](#), can be polled to monitor how many 32-bit words are currently resident in RNGOUT. When in Normal Mode, a new random word is pushed into RNGOUT every 256 clock cycles (as long as RNGOUT is not full). It is important that the host polls RNGSR to make sure random values are present before reading from RNGOUT.



**Figure 19-5. Output Register (RNGOUT)**

Random Output (RANDOM\_OUTPUT)

32 bits of random data.

## 19.5 Functional Description

The RNGA has three functional areas. They are the Output Register (RNGOUT), IF Unit and the RNGA Core/Control Logic blocks. Each of these can be seen in [Figure 19-1](#). The following sections describe the blocks in more detail.

### 19.5.1 Output Register

The Output Register (RNGOUT) provides temporary storage for random data generated by the RNGA Core/Control Logic block. The Status register, RNGSR, described in [Section 19.4.1.2, “Status Register \(RNGSR\)”](#), allows the host to monitor the number of random words in RNGOUT through the RNGSR[ORL] field. If the host reads from RNGOUT when it is empty and the interrupt is enabled, the RNGA indicates an error. It is important that the host polls RNGSR to make sure random values are present before reading from RNGOUT.

### 19.5.2 Interface Block

This block translates the interface block signals for slave control of the RNGA. The interface supports 32-bit word aligned accesses only.

### 19.5.3 RNGA Core/Control Logic Block

This block contains the RNGA’s control logic as well as its core engine used to generate random data. A diagram is shown in [Figure 19-6](#).

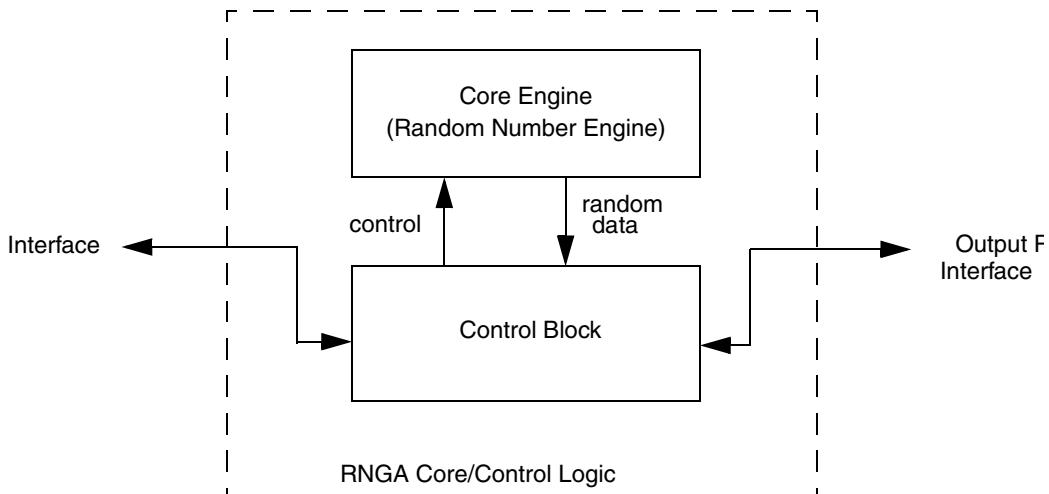


Figure 19-6. RNGA Logic Block Diagram

### 19.5.3.1 Control

The Control Block contains the address decoder, all addressable registers, and control state machines for the RNGA. This block is responsible for communication with the Slave interface and the RNGOUT interface. The block also controls the Core Engine to generate random data. The general functionality of the block is as follows. After reset, entropy is generated and stored in the RNGA's shift registers. After RNGCR[GO] is written, RNGOUT is loaded with a random word every 256 cycles. The process of loading RNGOUT continues as long as RNGOUT is not full.

### 19.5.3.2 Core Engine

The Core Engine Block contains the logic used to generate random data. The logic within the Core Engine contains the internal shift registers as well as the logic used to generate the two oscillator based clocks. This logic is brainless and must be controlled by the Control Block. The Control Block controls how the shift registers are configured as well as when the oscillator clocks are turned on.

## 19.6 Initialization/Application Information

The intended general operation of the RNGA is as follows:

1. Reset/initialize.
2. Set RNGCR[IM], RNGCR[HA], and RNGCR[GO].
3. Poll RNGSR[ORL] to ensure RNGOUT contains random data.
4. Read available random data from RNGOUT.
5. Repeat steps 3 and 4 as needed.

# Chapter 20

## Analog Comparator (S08ACMPV2)

### 20.1 Introduction

The analog comparator module (ACMP) provides a circuit for comparing two analog input voltages or for comparing one analog input voltage to an internal reference voltage. The comparator circuit is designed to operate across the full range of the supply voltage (rail-to-rail operation).

#### NOTE

Ignore any references to stop1 low-power mode in this chapter, because this device does not support it.

For details on low-power mode operation, refer to [Table 3-2](#) in [Chapter 3](#), “[Modes of Operation](#)”.

#### 20.1.1 ACMP Configuration Information

When using the bandgap reference voltage for input to ACMP+, the user must enable the bandgap buffer by setting SPMSC1[BGBE] (see [Section 5.7.6, “System Power Management Status and Control 1 Register \(SPMSC1\)”](#)). See this device’s data sheet for the value of the bandgap voltage.

#### 20.1.2 ACMP/TPM Configuration Information

The ACMP module can be configured to connect the output of the analog comparator to a TPM1 input capture channel 0 by setting the corresponding ACIC bit in the SOPT2 register (see [Section 5.7.4, “System Options 2 \(SOPT2\) Register](#)). With ACIC set, the TPM1CH0 pin is not available externally regardless of the configuration of the TPM1 module for channel 0.

#### 20.1.3 ACMP Clock Gating

The bus clock to the ACMP module can be gated on and off using the SCGC2[ACMP] bit (see [Section 5.7.9, “System Clock Gating Control 2 Register \(SCGC2\)”](#)). This bit is set after any reset, which enables the bus clock to this module. To conserve power, the SCGC2[ACMP] bit can be cleared to disable the clock to this module when not in use. See [Section 5.6, “Peripheral Clock Gating,”](#) for details.

## 20.1.4 Features

The ACMP has the following features:

- Full rail to rail supply operation.
- Selectable interrupt on rising edge, falling edge, or either rising or falling edges of comparator output.
- Option to compare to fixed internal bandgap reference voltage.
- Option to allow comparator output to be visible on a pin, ACMPO.

## 20.1.5 Modes of Operation

This section defines the ACMP operation in wait, stop and background debug modes.

### 20.1.5.1 ACMP in Wait Mode

The ACMP continues to run in wait mode if enabled before executing the WAIT instruction. Therefore, the ACMP can be used to bring the MCU out of wait mode if the ACMP interrupt, ACIE is enabled. For lowest possible current consumption, the ACMP should be disabled by software if not required as an interrupt source during wait mode.

### 20.1.5.2 ACMP in Stop Modes

#### 20.1.5.2.1 Stop3 Mode Operation

The ACMP continues to operate in Stop3 mode if enabled and compare operation remains active. If ACOPE is enabled, comparator output operates as in the normal operating mode and comparator output is placed onto the external pin. The MCU is brought out of stop when a compare event occurs and ACIE is enabled; ACF flag sets accordingly.

If stop is exited with a reset, the ACMP will be put into its reset state.

#### 20.1.5.2.2 Stop2 and Stop1 Mode Operation

During either Stop2 and Stop1 mode, the ACMP module will be fully powered down. Upon wake-up from Stop2 or Stop1 mode, the ACMP module will be in the reset state.

### 20.1.5.3 ACMP in Active Background Mode

When the microcontroller is in active background mode, the ACMP will continue to operate normally.

## 20.1.6 Block Diagram

The block diagram for the Analog Comparator module is shown [Figure 20-1](#).

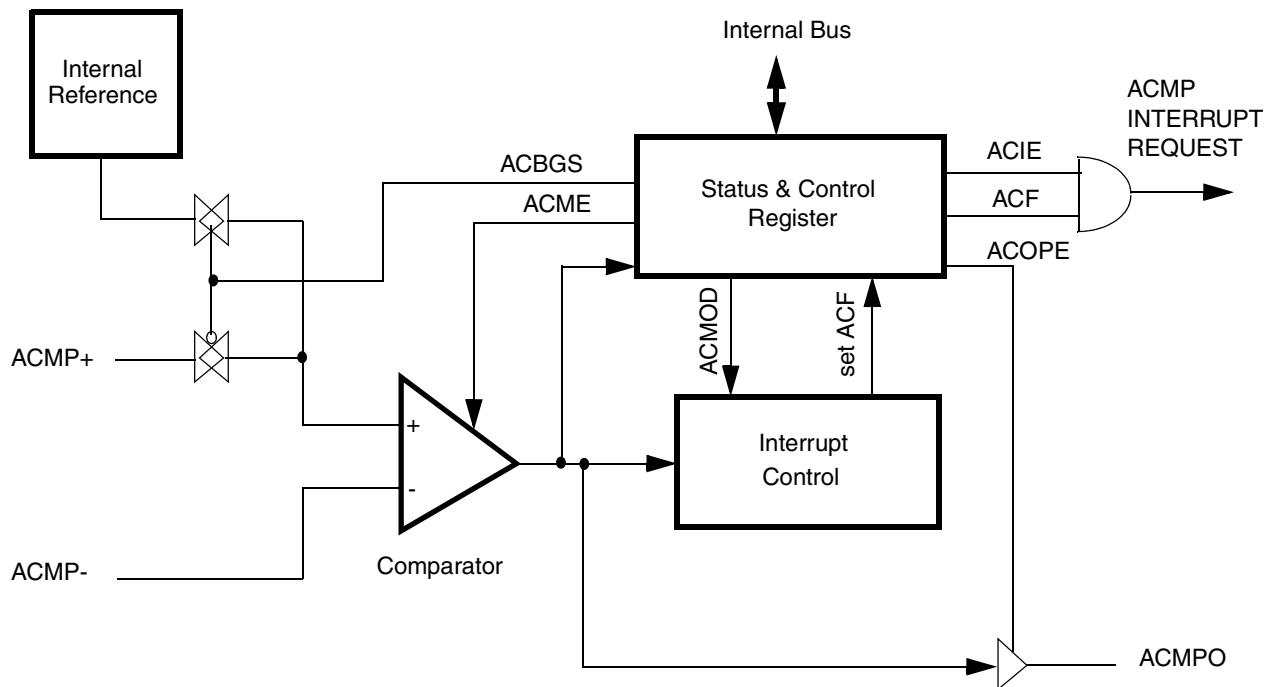


Figure 20-1. Analog Comparator 5 V Block Diagram

## 20.2 External Signal Description

The ACMP has two analog input pins, ACMP+ and ACMP-, and one digital output pin ACMPO. Each of these pins can accept an input voltage that varies across the full operating voltage range of the MCU. As shown in Figure 20-1, the ACMP- pin is connected to the inverting input of the comparator, and the ACMP+ pin is connected to the comparator non-inverting input if ACBGS is a 0. As shown in Figure 20-1, the ACMPO pin can be enabled to drive an external pin.

The signal properties of ACMP are shown in Table 20-1.

Table 20-1. Signal Properties

Signal	Function	I/O
ACMP-	Inverting analog input to the ACMP. (Minus input)	I
ACMP+	Non-inverting analog input to the ACMP. (Positive input)	I
ACMPO	Digital output of the ACMP.	O

## 20.3 Memory Map and Register Definition

### 20.3.1 Memory Map (Register Summary)

**Table 20-2. ACMP Register Summary**

Name		7	6	5	4	3	2	1	0
ACMPSC	R W	ACME	ACBGS	ACF	ACIE	ACO	ACOPE	ACMOD	

### 20.3.2 Register Descriptions

The ACMP includes one register:

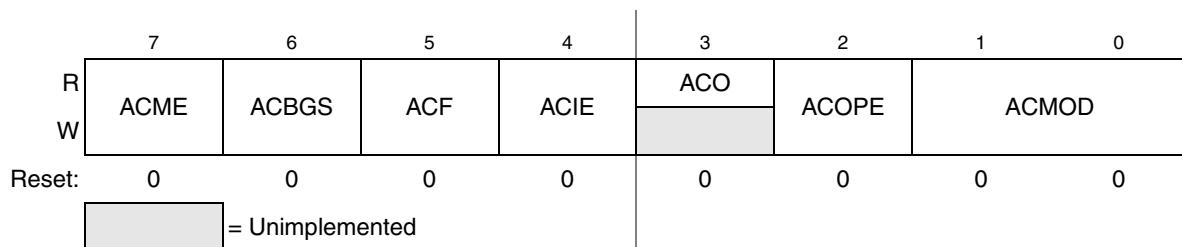
- An 8-bit status and control register

Refer to the direct-page register summary in the memory section of this data sheet for the absolute address assignments for all ACMP registers. This section refers to registers and control bits only by their names.

Some MCUs may have more than one ACMP, so register names include placeholder characters to identify which ACMP is being referenced.

#### 20.3.2.1 ACMP Status and Control Register (ACMPSC)

ACMPSC contains the status flag and control bits which are used to enable and configure the ACMP.



**Figure 20-2. ACMP Status and Control Register**

**Table 20-3. ACMP Status and Control Register Field Descriptions**

Field	Description
7 ACME	<b>Analog Comparator Module Enable</b> — ACME enables the ACMP module. 0 ACMP not enabled 1 ACMP is enabled
6 ACBGS	<b>Analog Comparator Bandgap Select</b> — ACBGS is used to select between the bandgap reference voltage or the ACMP+ pin as the input to the non-inverting input of the analog comparator. 0 External pin ACMP+ selected as non-inverting input to comparator 1 Internal reference select as non-inverting input to comparator Note: refer to this chapter introduction to verify if any other config bits are necessary to enable the bandgap reference in the chip level.
5 ACF	<b>Analog Comparator Flag</b> — ACF is set when a compare event occurs. Compare events are defined by ACMOD. ACF is cleared by writing a one to ACF. 0 Compare event has not occurred 1 Compare event has occurred
4 ACIE	<b>Analog Comparator Interrupt Enable</b> — ACIE enables the interrupt from the ACMP. When ACIE is set, an interrupt will be asserted when ACF is set. 0 Interrupt disabled 1 Interrupt enabled
3 ACO	<b>Analog Comparator Output</b> — Reading ACO will return the current value of the analog comparator output. ACO is reset to a 0 and will read as a 0 when the ACMP is disabled (ACME = 0).
2 ACOPE	<b>Analog Comparator Output Pin Enable</b> — ACOPE is used to enable the comparator output to be placed onto the external pin, ACMPO. 0 Analog comparator output not available on ACMPO 1 Analog comparator output is driven out on ACMPO
1:0 ACMOD	<b>Analog Comparator Mode</b> — ACMOD selects the type of compare event which sets ACF. 00 Encoding 0 — Comparator output falling edge 01 Encoding 1 — Comparator output rising edge 10 Encoding 2 — Comparator output falling edge 11 Encoding 3 — Comparator output rising or falling edge

## 20.4 Functional Description

The analog comparator can be used to compare two analog input voltages applied to ACMP+ and ACMP-; or it can be used to compare an analog input voltage applied to ACMP- with an internal bandgap reference voltage. ACBGS is used to select between the bandgap reference voltage or the ACMP+ pin as the input to the non-inverting input of the analog comparator. The comparator output is high when the non-inverting input is greater than the inverting input, and is low when the non-inverting input is less than the inverting input. ACMOD is used to select the condition which will cause ACF to be set. ACF can be set on a rising edge of the comparator output, a falling edge of the comparator output, or either a rising or a falling edge (toggle). The comparator output can be read directly through ACO. The comparator output can be driven onto the ACMPO pin using ACOPE.



# Chapter 21

## Analog-to-Digital Converter (S08ADC12V1)

### 21.1 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

Ignore any references to stop1 low-power mode in this chapter, because this device does not support it.

For details on low-power mode operation, refer to [Table 3-2](#) in [Chapter 3](#), “[Modes of Operation](#)”.

#### 21.1.1 ADC Clock Gating

The bus clock to the ADC can be gated on and off using the SCGC1[ADC] bit (see [Section 5.7.8, “System Clock Gating Control 1 Register \(SCGC1\)”](#)). This bit is set after any reset, which 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.

#### 21.1.2 Module Configurations

This section provides information for configuring the ADC on this device.

##### 21.1.2.1 Channel Assignments

The ADC channel assignments for this device are shown in [Table 21-1](#). Reserved channels convert to an unknown value.

**Table 21-1. ADC Channel Assignment**

ADCH	Channel	Input
00000	AD0	PTB0/MISO2/ADP0
00001	AD1	PTB1/MOSI2/ADP1
00010	AD2	PTB2/SPSCK2/ADP2
00011	AD3	PTB3/ $\overline{SS2}$ /ADP3
00100	AD4	PTB4/KBIP4/ADP4
00101	AD5	PTB5/KBIP5/ADP5
00110	AD6	PTB6/ADP6
ADCH	Channel	Input
10000	AD16	$V_{REFL}$
10001	AD17	$V_{REFL}$
10010	AD18	$V_{REFL}$
10011	AD19	$V_{REFL}$
10100	AD20	$V_{REFL}$
10101	AD21	$V_{REFL}$
10110	AD22	Reserved

**Table 21-1. ADC Channel Assignment (continued)**

<b>ADCH</b>	<b>Channel</b>	<b>Input</b>	<b>ADCH</b>	<b>Channel</b>	<b>Input</b>
00111	AD7	PTB7/ADP7	10111	AD23	Reserved
01000	AD8	PTD0/ADP8/ACMP+	11000	AD24	Reserved
01001	AD9	PTD1/ADP9/ACMP-	11001	AD25	Reserved
01010	AD10	PTD3/KBIP3/ADP10	11010	AD26	Temperature Sensor <sup>1</sup>
01011	AD11	PTD4/ADP11	11011	AD27	Internal Bandgap
01100	AD12	$V_{REFL}$	11100	—	Reserved
01101	AD13	$V_{REFL}$	11101	$V_{REFH}$	$V_{DD}$
01110	AD14	$V_{REFL}$	11110	$V_{REFL}$	$V_{SS}$
01111	AD15	$V_{REFL}$	11111	Module Disabled	None

<sup>1</sup> For information, see [Section 21.1.2.4, “Temperature Sensor.”](#)

### NOTE

Selecting the internal bandgap channel requires SPMSC1[BGBE] to be set (see [Section 5.7.6, “System Power Management Status and Control 1 Register \(SPMSC1\)”](#)). See this device’s data sheet for the value of the bandgap voltage.

#### 21.1.2.2 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 MCGERCLK. See [Chapter 7, “Multipurpose Clock Generator \(S08MCGV3\),”](#) for more information.

#### 21.1.2.3 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 RTCINT matches RTCMOD. The RTC interrupt does not have to be enabled to trigger the ADC.

The RTC can be configured to cause a hardware trigger in MCU run, wait, and stop3.

#### 21.1.2.4 Temperature Sensor

The ADC module includes a temperature sensor whose output is connected to one of the ADC analog channel inputs. [Equation 21-1](#) provides an approximate transfer function of the temperature sensor.

$$\text{Temp} = 25 - ((V_{TEMP} - V_{TEMP25}) \div m) \quad \text{Eqn. 21-1}$$

where:

- $V_{TEMP}$  is the voltage of the temperature sensor channel at the ambient temperature.
- $V_{TEMP25}$  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  $V_{TEMP25}$  and  $m$  values in the data sheet.

In application code, the user reads the temperature sensor channel, calculates  $V_{TEMP}$ , and compares to  $V_{TEMP25}$ . If  $V_{TEMP}$  is greater than  $V_{TEMP25}$  the cold slope value is applied in [Equation 21-1](#). If  $V_{TEMP}$  is less than  $V_{TEMP25}$  the hot slope value is applied in [Equation 21-1](#).

### 21.1.3 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
- Temperature sensor

### 21.1.4 ADC Module Block Diagram

Figure 21-1 provides a block diagram of the ADC module.

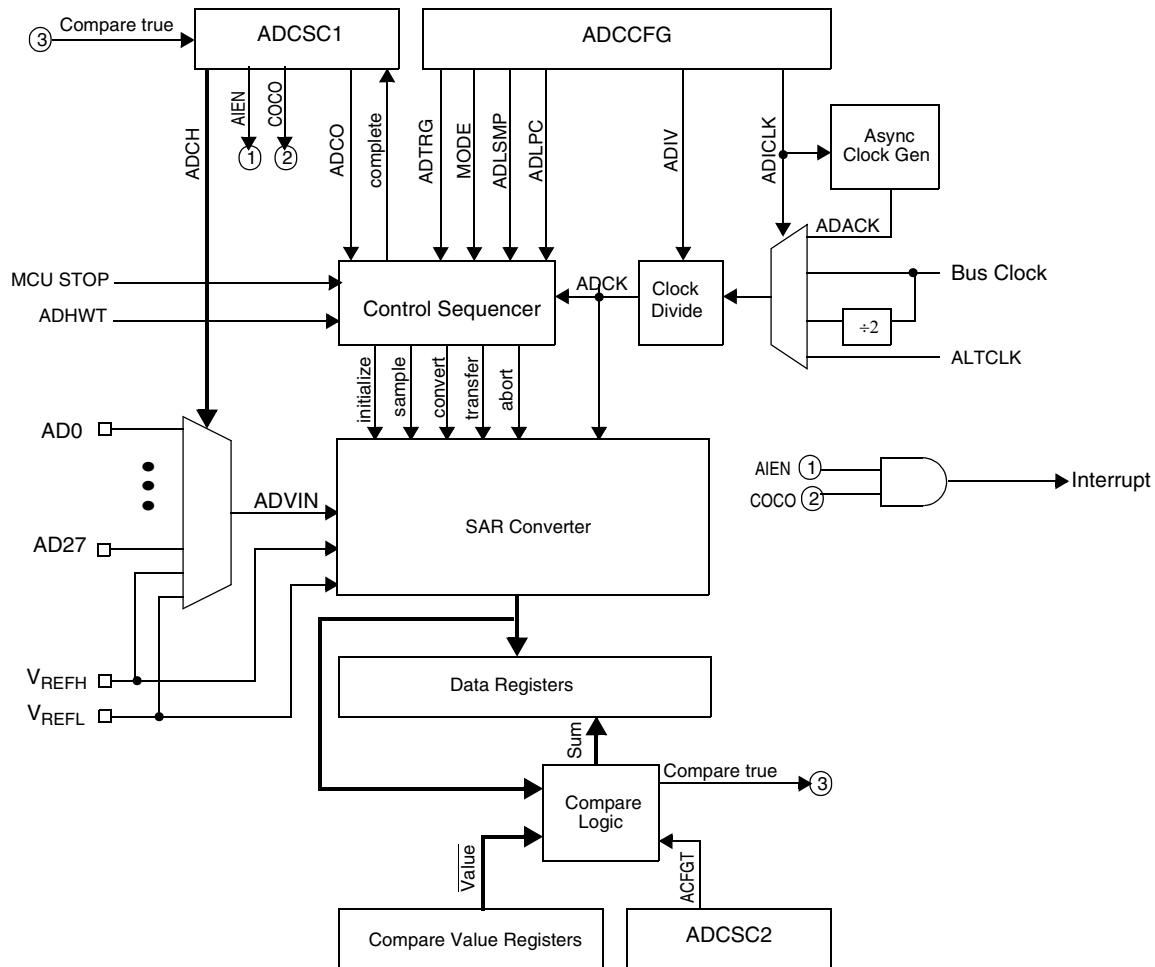


Figure 21-1. ADC Block Diagram

## 21.2 External Signal Description

The ADC module supports up to 28 separate analog inputs. It also requires four supply/reference/ground connections.

Table 21-2. 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>DDA</sub>	Analog power supply
V <sub>SSA</sub>	Analog ground

### 21.2.1 Analog Power ( $V_{DDA}$ )

The ADC analog portion uses  $V_{DDA}$  as its power connection. In some packages,  $V_{DDA}$  is connected internally to  $V_{DD}$ . If externally available, connect the  $V_{DDA}$  pin to the same voltage potential as  $V_{DD}$ . External filtering may be necessary to ensure clean  $V_{DDA}$  for good results.

### 21.2.2 Analog Ground ( $V_{SSA}$ )

The ADC analog portion uses  $V_{SSA}$  as its ground connection. In some packages,  $V_{SSA}$  is connected internally to  $V_{SS}$ . If externally available, connect the  $V_{SSA}$  pin to the same voltage potential as  $V_{SS}$ .

### 21.2.3 Voltage Reference High ( $V_{REFH}$ )

$V_{REFH}$  is the high reference voltage for the converter. In some packages,  $V_{REFH}$  is connected internally to  $V_{DDA}$ . If externally available,  $V_{REFH}$  may be connected to the same potential as  $V_{DDA}$  or may be driven by an external source between the minimum  $V_{DDA}$  spec and the  $V_{DDA}$  potential ( $V_{REFH}$  must never exceed  $V_{DDA}$ ).

### 21.2.4 Voltage Reference Low ( $V_{REFL}$ )

$V_{REFL}$  is the low-reference voltage for the converter. In some packages,  $V_{REFL}$  is connected internally to  $V_{SSA}$ . If externally available, connect the  $V_{REFL}$  pin to the same voltage potential as  $V_{SSA}$ .

### 21.2.5 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.

## 21.3 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
- Pin control registers, APCTL1, APCTL2, APCTL3

### 21.3.1 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).

	7	6	5	4		3	2	1	0
R	COCO	AIEN	ADCO		ADCH				
W									
Reset:	0	0	0	1		1	1	1	1

Figure 21-2. Status and Control Register (ADCSC1)

Table 21-3. 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 21-4</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 21-4. 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

### 21.3.2 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 21-3. Status and Control Register 2 (ADCSC2)

<sup>1</sup> Bits 1 and 0 are reserved bits that must always be written to 0.

Table 21-5. 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 triggers 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 value 1 Compare triggers when input is greater than or equal to compare value

### 21.3.3 Data Result High Register (ADCRH)

In 12-bit operation, ADCRH contains the upper four bits of 12-bit conversion data. In 10-bit operation, ADCRH contains the upper two bits of 10-bit conversion data. 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. When configured for 10-bit mode, ADR[11:10] are cleared. When configured for 8-bit mode, ADR[11:8] are cleared.

When automatic compare is not enabled, the value stored in ADCRH are the upper bits of the conversion result. When automatic compare is enabled, the conversion result is manipulated as described in [Section 21.4.5, “Automatic Compare Function”](#) prior to storage in ADCRH:ADCRL registers.

In 12-bit and 10-bit mode, reading ADCRH prevents the ADC from transferring subsequent conversion data into the result registers until ADCRL is read. If ADCRL is not read until after the next conversion is completed, the intermediate conversion data 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

Figure 21-4. Data Result High Register (ADCRH)

### 21.3.4 Data Result Low Register (ADCRL)

ADCRL contains the lower eight bits of a 12-bit or 10-bit conversion data, and all eight bits of 8-bit conversion data. ADCRL is updated each time a conversion completes except when automatic compare is enabled and the compare condition is not met.

When automatic compare is not enabled, the value stored in ADCRL is the lower eight bits of the conversion result. When automatic compare is enabled, the conversion result is manipulated as described in [Section 21.4.5, “Automatic Compare Function”](#) prior to storage in ADCRH:ADCRL registers.

In 12-bit and 10-bit mode, reading ADCRH prevents the ADC from transferring subsequent conversion data into the result registers until ADCRL is read. If ADCRL is not read until after the next conversion is completed, the intermediate conversion data is 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

Figure 21-5. Data Result Low Register (ADCRL)

### 21.3.5 Compare Value High Register (ADCCVH)

In 12-bit mode, the ADCCVH register holds the upper four bits of the 12-bit compare value. When the compare function is enabled, these bits are compared to the upper four bits of the result following a conversion in 12-bit mode.

	7	6	5	4		3	2	1	0
R	0	0	0	0		ADCV11	ADCV10	ADCV9	ADCV8
W						0	0	0	0
Reset:	0	0	0	0		0	0	0	0

Figure 21-6. Compare Value High Register (ADCCVH)

In 10-bit mode, the ADCCVH register holds the upper two bits of the 10-bit compare value (ADCV[9:8]). 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.

### 21.3.6 Compare Value Low Register (ADCCVL)

This register holds the lower eight bits of the 12-bit or 10-bit compare value or all eight bits of the 8-bit compare value. When the compare function is enabled, bits ADCV[7:0] are compared to the lower eight 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	ADCV7	ADCV6	ADCV5	ADCV4		ADCV3	ADCV2	ADCV1	ADCV0
W						0	0	0	0
Reset:	0	0	0	0		0	0	0	0

Figure 21-7. Compare Value Low Register(ADCCVL)

### 21.3.7 Configuration Register (ADCCFG)

ADCCFG selects the mode of operation, clock source, clock divide, and configures for low power and long sample time.

	7	6	5	4		3	2	1	0
R	ADLPC	ADIV	ADLSMP		MODE		ADICLK		
W						0	0	0	0
Reset:	0	0	0	0		0	0	0	0

Figure 21-8. Configuration Register (ADCCFG)

**Table 21-6. ADCCFG Register Field Descriptions**

<b>Field</b>	<b>Description</b>
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 21-7</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
3:2 MODE	Conversion Mode Selection. MODE bits are used to select between 12-, 10-, or 8-bit operation. See <a href="#">Table 21-8</a> .
1:0 ADICLK	Input Clock Select. ADICLK bits select the input clock source to generate the internal clock ADCK. See <a href="#">Table 21-9</a> .

**Table 21-7. 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 21-8. Conversion Modes**

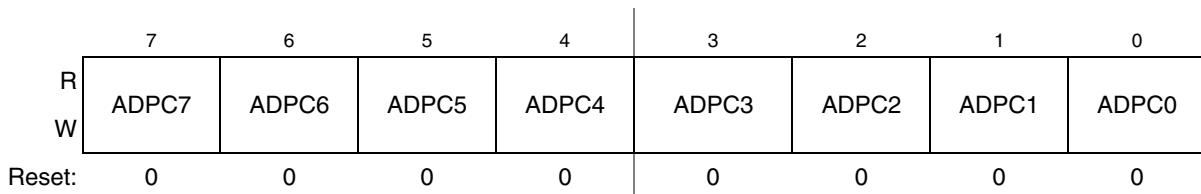
<b>MODE</b>	<b>Mode Description</b>
00	8-bit conversion (N=8)
01	12-bit conversion (N=12)
10	10-bit conversion (N=10)
11	Reserved

**Table 21-9. 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)

### 21.3.8 Pin Control 1 Register (APCTL1)

The pin control registers disable the I/O port control of MCU pins used as analog inputs. APCTL1 is used to control the pins associated with channels 0–7 of the ADC module.

**Figure 21-9. Pin Control 1 Register (APCTL1)****Table 21-10. APCTL1 Register Field Descriptions**

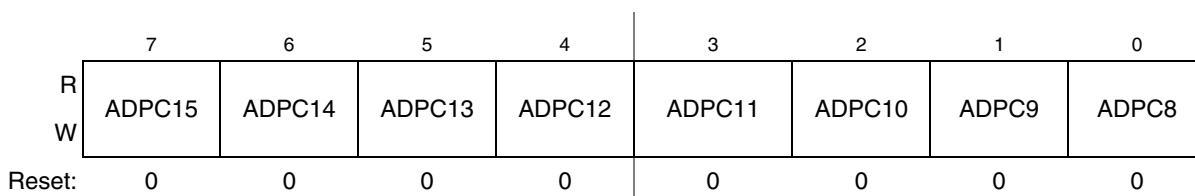
<b>Field</b>	<b>Description</b>
7 ADPC7	ADC Pin Control 7. ADPC7 controls the pin associated with channel AD7. 0 AD7 pin I/O control enabled 1 AD7 pin I/O control disabled
6 ADPC6	ADC Pin Control 6. ADPC6 controls the pin associated with channel AD6. 0 AD6 pin I/O control enabled 1 AD6 pin I/O control disabled
5 ADPC5	ADC Pin Control 5. ADPC5 controls the pin associated with channel AD5. 0 AD5 pin I/O control enabled 1 AD5 pin I/O control disabled
4 ADPC4	ADC Pin Control 4. ADPC4 controls the pin associated with channel AD4. 0 AD4 pin I/O control enabled 1 AD4 pin I/O control disabled
3 ADPC3	ADC Pin Control 3. ADPC3 controls the pin associated with channel AD3. 0 AD3 pin I/O control enabled 1 AD3 pin I/O control disabled
2 ADPC2	ADC Pin Control 2. ADPC2 controls the pin associated with channel AD2. 0 AD2 pin I/O control enabled 1 AD2 pin I/O control disabled

**Table 21-10. APCTL1 Register Field Descriptions (continued)**

Field	Description
1 ADPC1	ADC Pin Control 1. ADPC1 controls the pin associated with channel AD1. 0 AD1 pin I/O control enabled 1 AD1 pin I/O control disabled
0 ADPC0	ADC Pin Control 0. ADPC0 controls the pin associated with channel AD0. 0 AD0 pin I/O control enabled 1 AD0 pin I/O control disabled

### 21.3.9 Pin Control 2 Register (APCTL2)

APCTL2 controls channels 8–15 of the ADC module.

**Figure 21-10. Pin Control 2 Register (APCTL2)****Table 21-11. APCTL2 Register Field Descriptions**

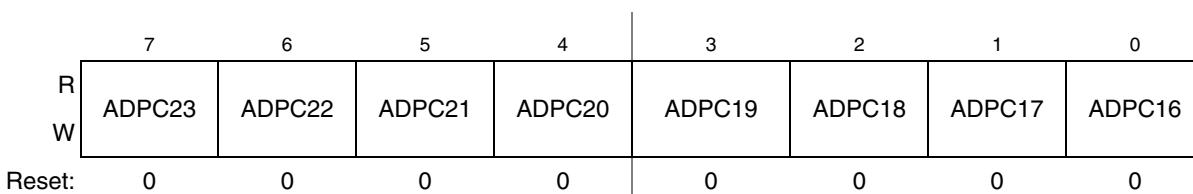
Field	Description
7 ADPC15	ADC Pin Control 15. ADPC15 controls the pin associated with channel AD15. 0 AD15 pin I/O control enabled 1 AD15 pin I/O control disabled
6 ADPC14	ADC Pin Control 14. ADPC14 controls the pin associated with channel AD14. 0 AD14 pin I/O control enabled 1 AD14 pin I/O control disabled
5 ADPC13	ADC Pin Control 13. ADPC13 controls the pin associated with channel AD13. 0 AD13 pin I/O control enabled 1 AD13 pin I/O control disabled
4 ADPC12	ADC Pin Control 12. ADPC12 controls the pin associated with channel AD12. 0 AD12 pin I/O control enabled 1 AD12 pin I/O control disabled
3 ADPC11	ADC Pin Control 11. ADPC11 controls the pin associated with channel AD11. 0 AD11 pin I/O control enabled 1 AD11 pin I/O control disabled
2 ADPC10	ADC Pin Control 10. ADPC10 controls the pin associated with channel AD10. 0 AD10 pin I/O control enabled 1 AD10 pin I/O control disabled

**Table 21-11. APCTL2 Register Field Descriptions (continued)**

Field	Description
1 ADPC9	ADC Pin Control 9. ADPC9 controls the pin associated with channel AD9. 0 AD9 pin I/O control enabled 1 AD9 pin I/O control disabled
0 ADPC8	ADC Pin Control 8. ADPC8 controls the pin associated with channel AD8. 0 AD8 pin I/O control enabled 1 AD8 pin I/O control disabled

### 21.3.10 Pin Control 3 Register (APCTL3)

APCTL3 controls channels 16–23 of the ADC module.

**Figure 21-11. Pin Control 3 Register (APCTL3)****Table 21-12. APCTL3 Register Field Descriptions**

Field	Description
7 ADPC23	ADC Pin Control 23. ADPC23 controls the pin associated with channel AD23. 0 AD23 pin I/O control enabled 1 AD23 pin I/O control disabled
6 ADPC22	ADC Pin Control 22. ADPC22 controls the pin associated with channel AD22. 0 AD22 pin I/O control enabled 1 AD22 pin I/O control disabled
5 ADPC21	ADC Pin Control 21. ADPC21 controls the pin associated with channel AD21. 0 AD21 pin I/O control enabled 1 AD21 pin I/O control disabled
4 ADPC20	ADC Pin Control 20. ADPC20 controls the pin associated with channel AD20. 0 AD20 pin I/O control enabled 1 AD20 pin I/O control disabled
3 ADPC19	ADC Pin Control 19. ADPC19 controls the pin associated with channel AD19. 0 AD19 pin I/O control enabled 1 AD19 pin I/O control disabled
2 ADPC18	ADC Pin Control 18. ADPC18 controls the pin associated with channel AD18. 0 AD18 pin I/O control enabled 1 AD18 pin I/O control disabled

**Table 21-12. APCTL3 Register Field Descriptions (continued)**

Field	Description
1 ADPC17	ADC Pin Control 17. ADPC17 controls the pin associated with channel AD17. 0 AD17 pin I/O control enabled 1 AD17 pin I/O control disabled
0 ADPC16	ADC Pin Control 16. ADPC16 controls the pin associated with channel AD16. 0 AD16 pin I/O control enabled 1 AD16 pin I/O control disabled

## 21.4 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.

### 21.4.1 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, which is equal to the frequency at which software is executed. 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.

## 21.4.2 Input Select and Pin Control

The pin control registers (APCTL3, APCTL2, and APCTL1) 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.

## 21.4.3 Hardware Trigger

The ADC module has a selectable asynchronous hardware conversion trigger, ADHWT, that is 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.

## 21.4.4 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.

### 21.4.4.1 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.

#### 21.4.4.2 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.

#### 21.4.4.3 Aborting Conversions

Any conversion in progress is aborted when:

- A write to ADCSC1 occurs (the current conversion will be aborted and a new conversion will be 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.

#### 21.4.4.4 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).

#### 21.4.4.5 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 21-13](#).

**Table 21-13. 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
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.

## 21.4.5 Automatic Compare Function

The compare function is enabled by the ACFE bit. The compare function can be configured to check for an upper or lower limit. After the input is sampled and converted, the compare value (ADCCVH and ADCCVL) is subtracted from the conversion result. When comparing to an upper limit (ACFGT = 1), if the conversion 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. An ADC interrupt is generated upon the setting of COCO if the ADC interrupt is enabled (AIEN = 1).

The subtract operation of two positive values (the conversion result less the compare value) results in a signed value that is 1-bit wider than the bit-width of the two terms. The final value transferred to the ADCRH and ADCRL registers is the result of the subtraction operation, excluding the sign bit. The value of the sign bit can be derived based on ACFGT control setting. When ACFGT=1, the sign bit of any value stored in ADCRH and ADCRL is always 0, indicating a positive result for the subtract operation. When ACFGT = 1, the sign bit of any result is always 1, indicating a negative result for the subtract operation.

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.

### 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.

An example of compare operation eases understanding of the compare feature. If the ADC is configured for 10-bit operation, ACFGT=0, and ADCCVH:ADCCVL= 0x200, then a conversion result of 0x080 causes the compare condition to be met and the COCO bit is set. A value of 0x280 is stored in ADCRH:ADCRL. This is signed data without the sign bit and must be combined with a derived sign bit to have meaning. The value stored in ADCRH:ADCRL is calculated as follows.

The value to interpret from the data is (Result – Compare Value) = (0x080 – 0x200) = -0x180. A standard method for handling subtraction is to convert the second term to its 2's complement, and then add the two terms. First calculate the 2's complement of 0x200 by complementing each bit and adding 1. Note that prior to complementing, a sign bit of 0 is added so that the 10-bit compare value becomes a 11-bit signed value that is always positive.

$$\begin{array}{r}
 \%101\ 1111\ 1111 & \text{<= 1's complement of 0x200 compare value} \\
 + & \%1 \\
 \hline
 \%110\ 0000\ 0000 & \text{<= 2's complement of 0x200 compare value}
 \end{array}$$

Then the conversion result of 0x080 is added to 2's complement of 0x200:

$$\begin{array}{r}
 \%000\ 1000\ 0000 \\
 + \%110\ 0000\ 0000 \\
 \hline
 \%110\ 1000\ 0000 & \text{<= Subtraction result is -0x180 in signed 11-bit data}
 \end{array}$$

The subtraction result is an 11-bit signed value. The lower 10 bits (0x280) are stored in ADCRH:ADCRL. The sign bit is known to be 1 (negative) because the ACFG<sub>T</sub>=0, the COCO bit was set, and conversion data was updated in ADCRH:ADCRL.

A simpler way to use the data stored in ADCRH:ADCRL is to apply the following rules. When comparing for upper limit (ACFG<sub>T</sub>=1), the value in ADCRH:ADCRL is a positive value and does not need to be manipulated. This value is the difference between the conversion result and the compare value. When comparing for lower limit (ACFG<sub>T</sub>=0), ADCRH:ADCRL is a negative value without the sign bit. If the value from these registers is complemented and then a value of 1 is added, then the calculated value is the unsigned (i.e., absolute) difference between the conversion result and the compare value. In the previous example, 0x280 is stored in ADCRH:ADCRL. The following example shows how the absolute value of the difference is calculated.

```
%01 0111 1111 <= Complement of 10-bit value stored in ADCRH:ADCRL
+
%1
-----
%01 1000 0000<= Unsigned value 0x180 is the absolute value of (Result - Compare Value)
```

## 21.4.6 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).

## 21.4.7 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.

### 21.4.7.1 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.

### 21.4.7.2 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 21.4.4.2, “Completing Conversions](#)) is cleared when entering stop3 and continuing ADC conversions.

### 21.4.8 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.

## 21.5 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 21-7](#), [Table 21-8](#), and [Table 21-9](#) 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.

### 21.5.1 ADC Module Initialization Example

#### 21.5.1.1 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 will be continuous or completed only once, and to enable or disable conversion complete interrupts. The input channel on which conversions will be performed is also selected here.

### 21.5.1.2 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
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 which is 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

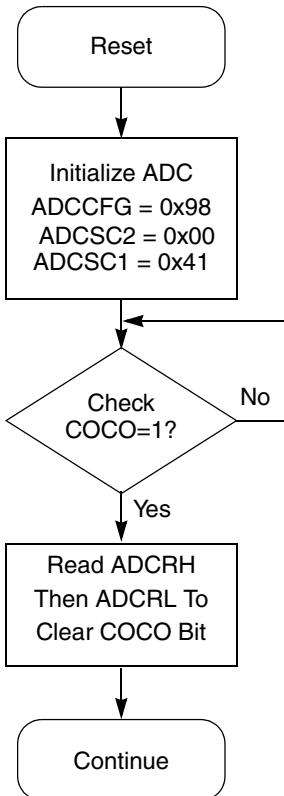
Holds compare value when compare function enabled

#### APCTL1=0x02

AD1 pin I/O control disabled. All other AD pins remain general purpose I/O pins

#### APCTL2=0x00

All other AD pins remain general purpose I/O pins



**Figure 21-12. Initialization Flowchart for Example**

## 21.6 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.

### 21.6.1 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.

#### 21.6.1.1 Analog Supply Pins

The ADC module has analog power and ground supplies ( $V_{DDA}$  and  $V_{SSA}$ ) available as separate pins on some devices.  $V_{SSA}$  is shared on the same pin as the MCU digital  $V_{SS}$  on some devices. On other devices,  $V_{SSA}$  and  $V_{DDA}$  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, both  $V_{DDA}$  and  $V_{SSA}$  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  $V_{SSA}$  pin. This should be the only ground connection between these supplies if possible. The  $V_{SSA}$  pin makes a good single point ground location.

### 21.6.1.2 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  $V_{DDA}$  on some devices. The low reference is  $V_{REFL}$ , which may be shared on the same pin as  $V_{SSA}$  on some devices.

When available on a separate pin,  $V_{REFH}$  may be connected to the same potential as  $V_{DDA}$ , or may be driven by an external source between the minimum  $V_{DDA}$  spec and the  $V_{DDA}$  potential ( $V_{REFH}$  must never exceed  $V_{DDA}$ ). When available on a separate pin,  $V_{REFL}$  must be connected to the same voltage potential as  $V_{SSA}$ .  $V_{REFH}$  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).

### 21.6.1.3 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  $V_{DD}$  or  $V_{SS}$ . 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  $V_{SSA}$ .

For proper conversion, the input voltage must fall between  $V_{REFH}$  and  $V_{REFL}$ . If the input is equal to or exceeds  $V_{REFH}$ , 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  $V_{REFL}$ , the converter circuit converts it to 0x000. Input voltages between  $V_{REFH}$  and  $V_{REFL}$  are straight-line linear conversions. There is a brief current associated with  $V_{REFL}$  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.

## 21.6.2 Sources of Error

Several sources of error exist for A/D conversions. These are discussed in the following sections.

### 21.6.2.1 Sampling Error

For proper conversions, the input must be sampled long enough to achieve the proper accuracy. Given the maximum input resistance of approximately  $7\text{k}\Omega$  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 ( $R_{AS}$ ) 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.

### 21.6.2.2 Pin Leakage Error

Leakage on the I/O pins can cause conversion error if the external analog source resistance ( $R_{AS}$ ) is high. If this error cannot be tolerated by the application, keep  $R_{AS}$  lower than  $V_{DDA} / (2^N * I_{LEAK})$  for less than 1/4LSB leakage error ( $N = 8$  in 8-bit, 10 in 10-bit or 12 in 12-bit mode).

### 21.6.2.3 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  $V_{REFH}$  to  $V_{REFL}$ .
- There is a 0.1  $\mu\text{F}$  low-ESR capacitor from  $V_{DDA}$  to  $V_{SSA}$ .
- If inductive isolation is used from the primary supply, an additional 1  $\mu\text{F}$  capacitor is placed from  $V_{DDA}$  to  $V_{SSA}$ .
- $V_{SSA}$  (and  $V_{REFL}$ , if connected) is connected to  $V_{SS}$  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 wait instruction or stop instruction.
  - For stop3 mode operation, select ADACK as the clock source. Operation in stop3 reduces  $V_{DD}$  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<sub>DD</sub> 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  $\mu$ F capacitor (C<sub>AS</sub>) on the selected input channel to V<sub>REFL</sub> or V<sub>SSA</sub> (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 that is synchronous to ADCK cannot be averaged out.

#### 21.6.2.4 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. 21-2*

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 will be  $\pm 1/2$  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.

#### 21.6.2.5 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<sub>ZS</sub>) (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<sub>FS</sub>) — 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.

### 21.6.2.6 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 21.6.2.3](#) 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 22

## Timer/PWM Module

### 22.1 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 allows the TPM to be configured such that all channels may be 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.

#### 22.1.1 Features

The TPM includes these distinctive features:

- One to eight channels:
  - Each channel may be 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 may be configured for buffered, center-aligned pulse-width-modulation (CPWM) on all channels
- Timer clock source selectable as prescaled bus clock, fixed system clock, or an external clock pin
  - Prescale taps for divide-by 1, 2, 4, 8, 16, 32, 64, or 128
  - Fixed system clock source are synchronized to the bus clock by an on-chip synchronization circuit
  - External clock pin may be shared with any timer channel pin or a separated input pin
- 16-bit free-running or modulo up/down count operation
- Timer system enable
- One interrupt per channel plus terminal count interrupt

#### 22.1.2 Modes of Operation

For details on low-power mode operation, refer to [Table 3-2](#) in [Chapter 3, “Modes of Operation”](#).

In general, TPM channels may be 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 microcontroller is in active BDM background or BDM foreground mode, the TPM temporarily suspends all counting until the microcontroller returns to normal user operating mode. During stop mode, all system clocks, including the main oscillator, are stopped; therefore, the TPM is effectively disabled until clocks resume. During wait mode, the TPM continues to operate normally. Provided the TPM does not need to produce a real time reference or provide the interrupt source(s) needed to wake the MCU from wait mode, the user can save power 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) may be 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 may be 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. The user may 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, which is the 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.

### 22.1.3 Block Diagram

The TPM uses one input/output (I/O) pin per channel, TPMxCHn (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 22-1 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.

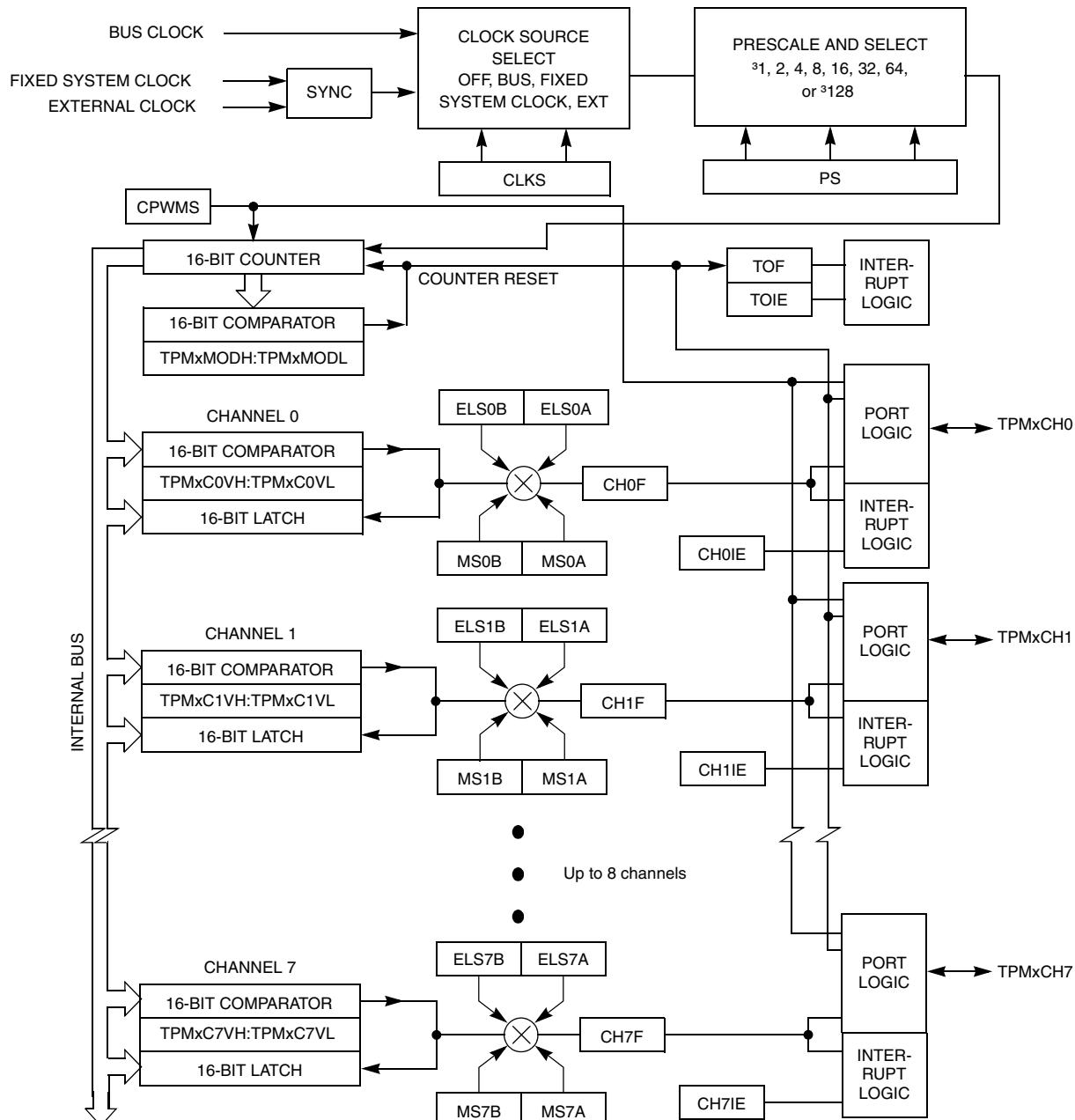


Figure 22-1. TPM Block Diagram

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.

If a channel is configured as input capture, an internal pullup device may be enabled for that channel. The details of how a module interacts with pin controls depends upon the chip implementation because the I/O pins and associated general purpose I/O controls are not part of the module. Refer to the discussion of the I/O port logic in a full-chip specification.

Because center-aligned PWMs are usually used to drive 3-phase AC-induction motors and brushless DC motors, they are typically used in sets of three or six channels.

## 22.2 Signal Description

[Table 22-1](#) shows the user-accessible signals for the TPM. The number of channels may be 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 22-1. Signal Properties**

Name	Function
EXTCLK <sup>1</sup>	External clock source that may be selected to drive the TPM counter.
TPMxCHn <sup>2</sup>	I/O pin associated with TPM channel n

<sup>1</sup> When preset, this signal can share 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 to 8)

Refer to documentation for the full-chip for details about reset states, port connections, and whether there is any pullup device on these pins.

TPM channel pins can be associated with general purpose I/O pins and have passive pullup devices that can be enabled with a control bit when the TPM or general purpose I/O controls have configured the associated pin as an input. When no TPM function is enabled to use a corresponding pin, the pin reverts to being controlled by general purpose I/O controls, including the port-data and data-direction registers. Immediately after reset, no TPM functions are enabled, so all associated pins revert to general purpose I/O control.

### 22.2.1 Detailed Signal Descriptions

This section describes each user-accessible pin signal in detail. Although [Table 22-1](#) grouped all channel pins together, any TPM pin can be shared with the external clock source signal. Because I/O pin logic is not part of the TPM, refer to full-chip documentation for a specific derivative for more details about the interaction of TPM pin functions and general purpose I/O controls including port data, data direction, and pullup controls.

### 22.2.1.1 EXTCLK — External Clock Source

Control bits in the timer status and control register allow the user to select nothing (timer disable), the bus-rate clock (the normal default source), a crystal-related clock, or an external clock as the clock that drives the TPM prescaler and subsequently the 16-bit TPM counter. The external clock source is synchronized in the TPM. The bus clock clocks the synchronizer; the frequency of the external source must be no more than one-fourth the frequency of the bus-rate clock, to meet Nyquist criteria and allowing for jitter.

The external clock signal shares the same pin as a channel I/O pin, so the channel pin is not usable for channel I/O function when selected as the external clock source. It is the user's responsibility to avoid such settings. If this pin is used as an external clock source (CLKSB:CLKSA = 1:1), the channel can be used in output compare mode as a software timer (ELSnB:ELSnA = 0:0).

### 22.2.1.2 TPMxCHn — TPM Channel n I/O Pin(s)

Each TPM channel is associated with an I/O pin on the MCU. The function of this pin depends on the channel configuration. The TPM pins share with general purpose I/O pins, where each pin has a port data register bit, and a data direction control bit, and the port has optional passive pullups that may be enabled when a port pin is acting as an input.

The TPM channel does not control the I/O pin when (ELSnB:ELSnA = 0:0) or when (CLKS = 00) so it normally reverts to general purpose I/O control. When CPWMS = 1 (and ELSnB:ELSnA not = 0:0), all channels within the TPM are configured for center-aligned PWM and the TPMxCHn pins are all controlled by the TPM system. When CPWMS=0, the MSnB:MSnA control bits determine whether the channel is configured for input capture, output compare, or edge-aligned PWM.

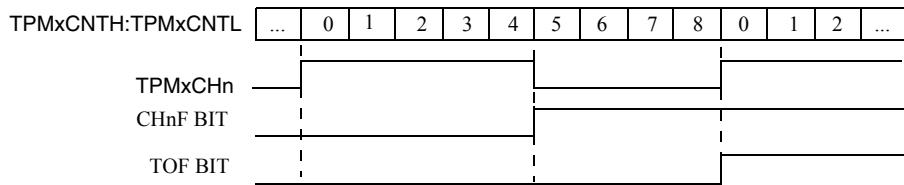
When a channel is configured for input capture (CPWMS=0, MSnB:MSnA = 0:0 and ELSnB:ELSnA not = 0:0), the TPMxCHn pin is forced to act as an edge-sensitive input to the TPM. ELSnB:ELSnA control bits determine what polarity edge or edges trigger input-capture events. A synchronizer based on the bus clock is used to synchronize input edges to 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). TPM uses this pin as an input capture input to override the port data and data direction controls for the same pin.

When a channel is configured for output compare (CPWMS=0, MSnB:MSnA = 0:1 and ELSnB:ELSnA not = 0:0), the associated data direction control is overridden, the TPMxCHn pin is considered an output controlled by the TPM, and the ELSnB:ELSnA control bits determine how the pin is controlled. The remaining three combinations of ELSnB:ELSnA determine whether the TPMxCHn pin is toggled, cleared, or set each time the 16-bit channel value register matches the timer 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—then the pin is toggled.

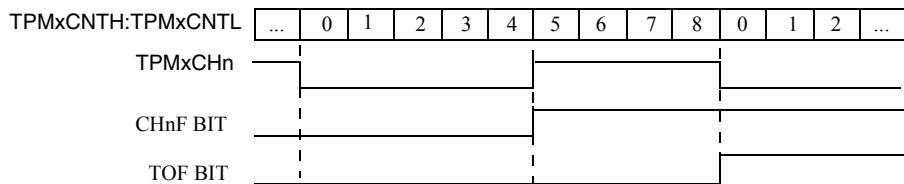
When a channel is configured for edge-aligned PWM (CPWMS=0, MSnB=1 and ELSnB:ELSnA not = 0:0), the data direction is overridden, the TPMxCHn pin is forced to be an output controlled by the TPM, and ELSnA controls the polarity of the PWM output signal on the pin. When ELSnB:ELSnA=1:0, the TPMxCHn pin is forced high at the start of each new period (TPMxCNT=0x0000), and the pin is forced low when the channel value register matches the timer counter. When ELSnA=1, the TPMxCHn pin is forced low at the start of each new period (TPMxCNT=0x0000), and the pin is forced high when the channel value register matches the timer counter.

TPMxMODH:TPMxMODL = 0x0008  
TPMxMODH:TPMxMODL = 0x0005



**Figure 22-2. High-True Pulse of an Edge-Aligned PWM**

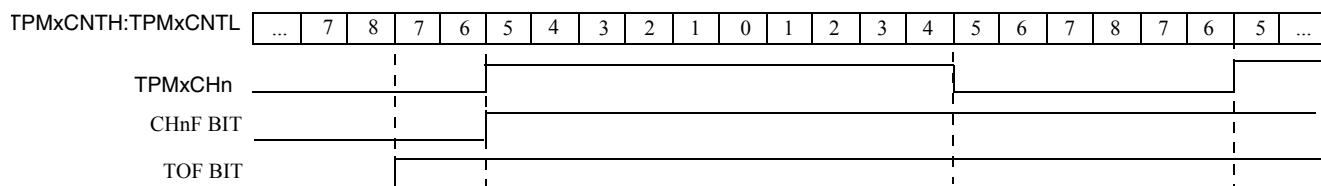
TPMxMODH:TPMxMODL = 0x0008  
TPMxMODH:TPMxMODL = 0x0005



**Figure 22-3. Low-True Pulse of an Edge-Aligned PWM**

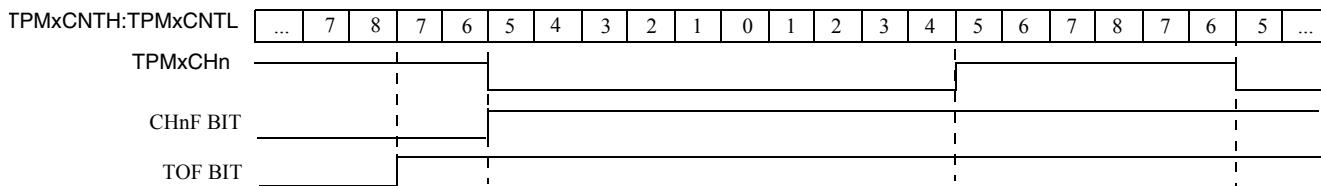
When the TPM is configured for center-aligned PWM (and ELSnB:ELSnA not = 0:0), the data direction for all channels in this TPM are overridden, the TPMxCHn pins are forced to be outputs controlled by the TPM, and the ELSnA bits control the polarity of each TPMxCHn output. If ELSnB:ELSnA=1:0, the corresponding TPMxCHn pin is cleared when the timer counter is counting up, and the channel value register matches the timer counter; the TPMxCHn pin is set when the timer counter is counting down, and the channel value register matches the timer counter. If ELSnA=1, the corresponding TPMxCHn pin is set when the timer counter is counting up and the channel value register matches the timer counter; the TPMxCHn pin is cleared when the timer counter is counting down and the channel value register matches the timer counter.

TPMxMODH:TPMxMODL = 0x0008  
TPMxMODH:TPMxMODL = 0x0005



**Figure 22-4. High-True Pulse of a Center-Aligned PWM**

TPMxMODH:TPMxMODL = 0x0008  
TPMxMODH:TPMxMODL = 0x0005



**Figure 22-5. Low-True Pulse of a Center-Aligned PWM**

## 22.3 Register Definition

This section consists of register descriptions in address order.

### 22.3.1 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.

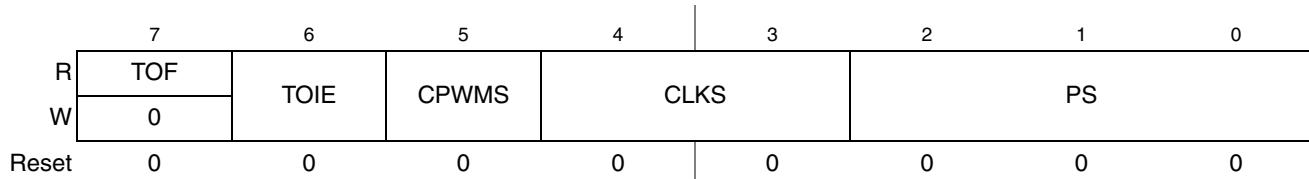


Figure 22-6. TPM Status and Control Register (TPMxSC)

Table 22-2. 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 complete, the sequence is reset so TOF would remain 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. When present, 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	Clock source selects. As shown in <a href="#">Table 22-3</a> , this 2-bit field is used to disable the TPM system or select one of three clock sources to drive the counter prescaler. The fixed system clock source is only meaningful in systems with a PLL-based system clock. When there is no PLL, the fixed-system clock source is the same as the bus rate clock. The external source is synchronized to the bus clock by TPM module, and the fixed system clock source (when a PLL is present) is synchronized to the bus clock by an on-chip synchronization circuit. When a PLL is present but not enabled, the fixed-system clock source is the same as the bus-rate clock.
2-0 PS	Prescale factor select. This 3-bit field selects one of 8 division factors for the TPM clock input as shown in <a href="#">Table 22-4</a> . This prescaler is located after any clock source synchronization or clock source selection so it affects the clock source selected to drive the TPM system. The new prescale factor affects the clock source on the next system clock cycle after the new value is updated into the register bits.

**Table 22-3. TPM-Clock-Source Selection**

<b>CLKS</b>	<b>TPM Clock Source to Prescaler Input</b>
00	No clock selected (TPM counter disable)
01	Bus rate clock
10	Fixed system clock
11	External source

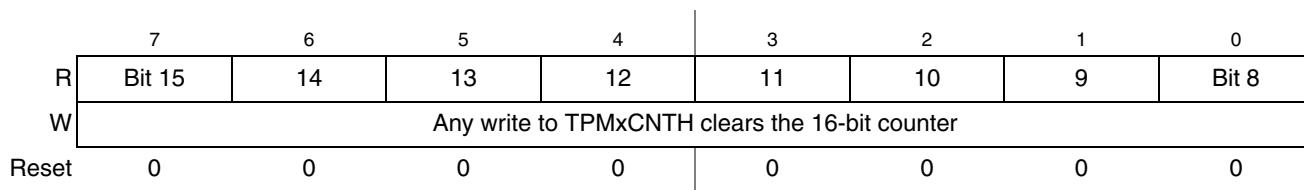
**Table 22-4. Prescale Factor Selection**

<b>PS</b>	<b>TPM Clock Source Divided-by</b>
000	1
001	2
010	4
011	8
100	16
101	32
110	64
111	128

### 22.3.2 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 22-7. TPM Counter Register High (TPMxCNTH)**

	7	6	5	4	3	2	1	0
R	Bit 7	6	5	4	3	2	1	Bit 0
W	Any write to TPMxCNTL clears the 16-bit counter							
Reset	0	0	0	0	0	0	0	0

**Figure 22-8. TPM Counter Register Low (TPMxCNTL)**

When BDM is active, the timer counter is frozen (this is the value read by user); the coherency mechanism is frozen such that 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 the user was 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.

### 22.3.3 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).

Writing to either byte (TPMxMODH or TPMxMODL) latches the value into a buffer and the registers are updated with the value of their write buffer according to the value of CLKS bits, so:

- If (CLKS[1:0] = 00), then the registers are updated when the second byte is written
- If (CLKS[1:0] not = 00), then 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 may be manually reset by writing to the TPMxSC address (whether BDM is active or not).

When BDM is active, the coherency mechanism is frozen such that 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.

	7	6	5	4	3	2	1	0
R	Bit 15	14	13	12	11	10	9	Bit 8
W	Any write to TPMxMODH bypasses the buffer latches and directly writes to the modulo register while BDM is active.							
Reset	0	0	0	0	0	0	0	0

**Figure 22-9. TPM Counter Modulo Register High (TPMxMODH)**

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

Reset the TPM counter before writing to the TPM modulo registers to avoid confusion about when the first counter overflow occurs.

### 22.3.4 TPM Channel n Status and Control Register (TPMxCnSC)

TPMxCnSC contains the channel-interrupt-status flag and control bits used to configure the interrupt enable, channel configuration, and pin function.

	7	6	5	4	3	2	1	0
R W	CHnF	CHnIE	MSnB	MSnA	ELSnB	ELSnA	0	0
Reset	0	0	0	0	0	0	0	0
= Unimplemented or Reserved								

Figure 22-11. TPM Channel n Status and Control Register (TPMxCnSC)

Table 22-5. 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 pin. 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 CHnF is set and interrupts are enabled (CHnIE = 1). Clear CHnF by reading TPMxCnSC while CHnF is set and then writing a logic 0 to CHnF. If another interrupt request occurs before the clearing sequence is complete, the sequence is reset so CHnF remains set after the clear sequence completed for the earlier CHnF. This is done so a CHnF interrupt request cannot be lost due to clearing a previous CHnF. Reset clears the CHnF 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 CHnIE. 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=0, MSnB=1 configures TPM channel n for edge-aligned PWM mode. Refer to the summary of channel mode and setup controls in Table 22-6.

**Table 22-5. TPMxCnSC Field Descriptions (continued)**

Field	Description
4 MSnA	Mode select A for TPM channel n. When CPWMS=0 and MSnB=0, MSnA configures TPM channel n for input-capture mode or output compare mode. Refer to <a href="#">Table 22-6</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 22-6</a> , these bits select the polarity of the input edge that triggers an input capture event, select the level driven in response to an output compare match, or select the polarity of the PWM output. Setting ELSnB:ELSnA to 0:0 configures the related timer pin as a general purpose I/O pin not related to any timer functions. This function is typically used to temporarily disable an input capture channel or to make the timer pin available as a general purpose I/O pin when the associated timer channel is set up as a software timer that does not require the use of a pin.

**Table 22-6. Mode, Edge, and Level Selection**

CPWMS	MSnB:MSnA	ELSnB:ELSnA	Mode	Configuration
X	XX	00		Pin not used for TPM - revert 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	01	Output compare	Toggle output on compare
		10		Clear output on compare
		11		Set output on compare
	1X	10	Edge-aligned PWM	High-true pulses (clear output on compare)
		X1		Low-true pulses (set output on compare)
	XX	10	Center-aligned PWM	High-true pulses (clear output on compare-up)
		X1		Low-true pulses (set output on compare-up)

### 22.3.5 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.

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

Figure 22-12. TPM Channel Value Register High (TPMxCnVH)

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

Figure 22-13. 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 such that 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 the user was 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 are written, they are transferred as a coherent 16-bit value into the timer-channel registers according to the value of CLKS bits and the selected mode, so:

- If (CLKS[1:0] = 00), then the registers are updated when the second byte is written.
- If (CLKS[1:0] not = 00 and in output compare mode) then 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 (CLKS[1:0] not = 00 and in EPWM or CPWM modes), then the registers are updated after the 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 then the update is made when the TPM counter changes from 0xFFFFE to 0xFFFF.

The latching mechanism may be manually reset by writing to the TPMxCnSC register (whether BDM mode is active or not). This latching mechanism allows coherent 16-bit writes in big-endian or little-endian order that is friendly to various compiler implementations.

When BDM is active, the coherency mechanism is frozen such that 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 has been fully exercised, the channel registers are updated using the buffered values written (while BDM was not active) by the user.

## 22.4 Functional Description

All TPM functions are associated with a central 16-bit counter that allows flexible selection of the clock source and prescale factor. There is also a 16-bit modulo register associated with the main 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 main 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 the main 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.

### 22.4.1 Counter

All timer functions are based on the main 16-bit counter (TPMxCNTH:TPMxCNTL). This section discusses selection of the clock source, end-of-count overflow, up-counting vs. up/down counting, and manual counter reset.

#### 22.4.1.1 Counter Clock Source

The 2-bit field, CLKS, in the timer status and control register (TPMxSC) selects one of three possible clock sources or OFF (which effectively disables the TPM). See [Table 22-3](#). After any MCU reset, CLKS=00 so no clock source is selected, and the TPM is in a low power state. These control bits may be read or written at any time and disabling the timer (writing 00 to the CLKS field) does not affect the values in the counter or other timer registers.

**Table 22-7. TPM Clock Source Selection**

<b>CLKS</b>	<b>TPM Clock Source to Prescaler Input</b>
00	No clock selected (TPM counter disabled)
01	Bus rate clock
10	Fixed system clock
11	External source

The bus rate clock is the main system bus clock for the MCU. This clock source requires no synchronization because it is the clock used for all internal MCU activities including operation of the CPU and buses.

In MCUs that have no PLL or the PLL is not engaged, the fixed system clock source is the same as the bus-rate-clock source, and it does not go through a synchronizer. When a PLL is present and engaged, a synchronizer is required between the crystal divided-by two clock source and the timer counter so counter transitions are properly aligned to bus-clock transitions. A synchronizer is used at chip level to synchronize the crystal-related source clock to the bus clock.

The external clock source may be connected to any TPM channel pin. This clock source always has to pass through a synchronizer to assure that counter transitions are properly aligned to bus clock transitions. The bus-rate clock drives the synchronizer; therefore, to meet Nyquist criteria even with jitter, the frequency of the external clock source must not be faster than the bus rate divided-by four. With ideal clocks the external clock can be as fast as bus clock divided by four.

When the external clock source shares the TPM channel pin, this pin should not be used for other channel timing functions. For example, it would be ambiguous to configure channel 0 for input capture when the TPM channel 0 pin was also being used as the timer external clock source. (It is the user's responsibility to avoid such settings.) The TPM channel could be used in output compare mode for software timing functions (pin controls set not to affect the TPM channel pin).

#### **22.4.1.2 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 hardware interrupt is generated, or interrupt-driven operation (TOIE=1) where a static hardware interrupt is generated when the TOF flag is equal to one.

The conditions causing TOF to become set depend on whether the TPM is configured for center-aligned PWM (CPWMS=1). In the simplest mode, there is no modulus limit and the TPM is not in CPWMS=1 mode. In this case, the 16-bit timer counter counts from 0x0000 through 0xFFFF and overflows to 0x0000 on the next counting clock. TOF becomes set at the transition from 0xFFFF to 0x0000. When a modulus limit is set, TOF becomes 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 gets 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).

### 22.4.1.3 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 then 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) becomes set at the end of the terminal-count period (as the count changes to the next lower count value).

### 22.4.1.4 Manual Counter Reset

The main timer counter can be manually reset at any time by writing any value to half of 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.

## 22.4.2 Channel Mode Selection

Provided CPWMS=0, the MSnB and MSnA control bits in the channel n status and control registers determine the basic mode of operation for the corresponding channel. Choices include input capture, output compare, and edge-aligned PWM.

### 22.4.2.1 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 may be chosen as the active edge that triggers an input capture.

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 the channel status/control register (TPMxCnSC).

An input capture event sets a flag bit (CHnF) that may optionally generate a CPU interrupt request.

While in BDM, the input capture function works as configured by the user. When an external event occurs, the TPM latches the contents of the TPM counter (which is frozen because of the BDM mode) into the channel value registers and sets the flag bit.

### 22.4.2.2 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 the channel-value registers of an output-compare channel, the TPM can set, clear, or toggle the channel pin.

In output compare mode, values are transferred to the corresponding timer channel registers only after 8-bit halves of a 16-bit register have been written and according to the value of CLKS bits, so:

- If (CLKS[1:0] = 00), the registers are updated when the second byte is written
- If (CLKS[1:0] not = 00), the registers are updated at the next change of the TPM counter (end of the prescaler counting) after the second byte is written.

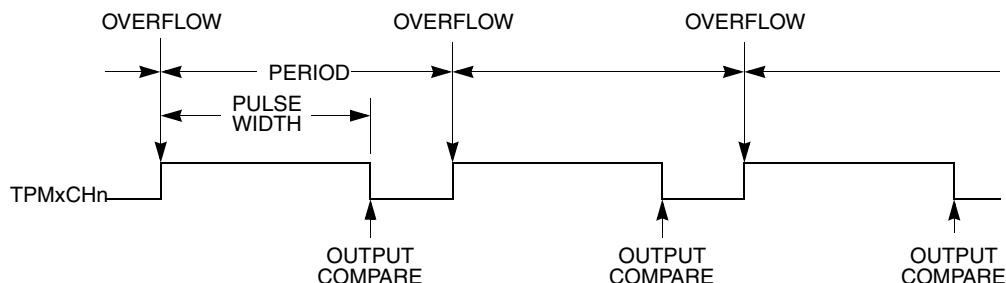
The coherency sequence can be manually reset by writing to the channel status/control register (TPMxCnSC).

An output compare event sets a flag bit (CHnF) that may optionally generate a CPU-interrupt request.

#### 22.4.2.3 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 setting in the timer channel register (TPMxCnVH:TPMxCnVL). The polarity of this PWM signal is determined by the setting in the ELSnA control bit. 0% and 100% duty cycle cases are possible.

The output compare value in the TPM channel registers determines the pulse width (duty cycle) of the PWM signal ([Figure 22-14](#)). The time between the modulus overflow and the output compare is the pulse width. If ELSnA=0, the counter overflow forces the PWM signal high, and the output compare forces the PWM signal low. If ELSnA=1, the counter overflow forces the PWM signal low, and the output compare forces the PWM signal high.



**Figure 22-14. PWM Period and Pulse Width (ELSnA=0)**

When the channel value register is set to 0x0000, the duty cycle is 0%. 100% duty cycle can be achieved by setting the timer-channel register (TPMxCnVH:TPMxCnVL) to a value greater than the modulus setting. This implies that the modulus setting must be less than 0xFFFF to get 100% duty cycle.

Because the TPM may be used in an 8-bit MCU, the settings in 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 edge-aligned PWM mode, values are transferred to the corresponding timer-channel registers according to the value of CLKS bits, so:

- If (CLKS[1:0] = 00), the registers are updated when the second byte is written
- If (CLKS[1:0] not = 00), the registers are updated after the 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 then the update is made when the TPM counter changes from 0xFFFFE to 0xFFFF.

#### 22.4.2.4 Center-Aligned PWM Mode

This type of PWM output uses the up/down counting mode of the timer counter (CPWMS=1). The output compare 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 should 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 output.

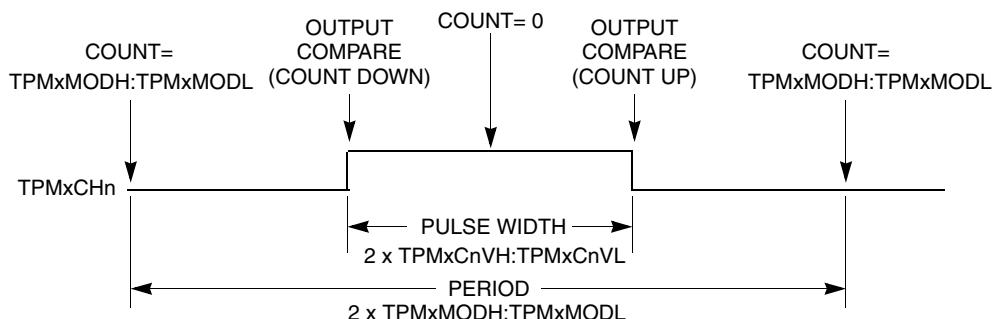
$$\text{pulse width} = 2 \times (\text{TPMxCnVH:TPMxCnVL})$$

$$\text{period} = 2 \times (\text{TPMxMODH:TPMxMODL}); \text{TPMxMODH:TPMxMODL}=0x0001-0x7FFF$$

If the channel-value register 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 duty cycle compare 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 would be much longer than required for normal applications.

TPMxMODH:TPMxMODL=0x0000 is a special case that should not be used with center-aligned PWM mode. When CPWMS=0, this case corresponds to the counter running free from 0x0000 through 0xFFFF, but when CPWMS=1 the counter needs a valid match to the modulus register somewhere other than at 0x0000 to change directions from up-counting to down-counting.

The output compare value in the TPM channel registers (times 2) determines the pulse width (duty cycle) of the CPWM signal (Figure 22-15). If ELSnA=0, a compare occurred while counting up forces the CPWM output signal low and a compare occurred while counting down forces the output high. 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 22-15. 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=1.

The TPM may be used in an 8-bit MCU. The settings in 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 TPMxMODH, TPMxMODL, TPMxCnVH, and TPMxCnVL, actually write to buffer registers.

In center-aligned PWM mode, the TPMxCnVH:L registers are updated with the value of their write buffer according to the value of CLKS bits, so:

- If (CLKS[1:0] = 00), the registers are updated when the second byte is written
- If (CLKS[1:0] not = 00), the registers are updated after the 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.

When TPMxCNTH:TPMxCNTL=TPMxMODH:TPMxMODL, the TPM can optionally generate a TOF interrupt (at the end of this count).

Writing to TPMxSC cancels any values written to TPMxMODH and/or TPMxMODL and resets the coherency mechanism for the modulo registers. Writing to TPMxCnSC cancels any values written to the channel value registers and resets the coherency mechanism for TPMxCnVH:TPMxCnVL.

## 22.5 Reset Overview

### 22.5.1 General

The TPM is reset when any MCU reset occurs.

### 22.5.2 Description of Reset Operation

Reset clears the TPMxSC register that disables clocks to the TPM and disables timer overflow interrupts (TOIE=0). CPWMS, MSnB, MSnA, ELSnB, and ELSnA are all cleared, which configures all TPM channels for input-capture operation with the associated pins disconnected from I/O pin logic (so all MCU pins related to the TPM revert to general purpose I/O pins).

## 22.6 Interrupts

### 22.6.1 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 22-8](#) which shows the interrupt name, the name of any local enable that can block the interrupt request from leaving the TPM and getting recognized by the separate interrupt processing logic.

**Table 22-8. Interrupt Summary**

Interrupt	Local Enable	Source	Description
TOF	TOIE	Counter overflow	Set each time the timer counter reaches its terminal count (at transition to next count value which is usually 0x0000)
CHnF	CHnIE	Channel event	An input capture or output compare event took place on channel n

The TPM module provides a high-true interrupt signal. Vectors and priorities are determined at chip integration time in the interrupt module so refer to the user's guide for the interrupt module or to the chip's complete documentation for details.

## 22.6.2 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 may be read (polled) by software to determine that the action has occurred, or an associated enable bit (TOIE or CHnIE) can be set to enable hardware interrupt generation. While the interrupt enable bit is set, a static interrupt generates when the associated interrupt flag equals one. The user's 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 (1) followed by a write of zero (0) 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.

### 22.6.2.1 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.

#### 22.6.2.1.1 Normal Case

Normally TOF is set when the timer counter changes from 0xFFFF to 0x0000. When the TPM is not configured for center-aligned PWM (CPWMS=0), TOF gets set when the timer counter changes from the terminal count (the value in the modulo register) to 0x0000. This case corresponds to the normal meaning of counter overflow.

### 22.6.2.1.2 Center-Aligned PWM Case

When CPWMS=1, TOF gets 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). In this case the TOF corresponds to the end of a PWM period.

### 22.6.2.2 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).

#### 22.6.2.2.1 Input Capture Events

When a channel is configured as an input capture channel, the ELSnB:ELSnA control bits select no edge (off), 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 22.6.2, “Description of Interrupt Operation.”](#)

#### 22.6.2.2.2 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 [Section 22.6.2, “Description of Interrupt Operation.”](#)

#### 22.6.2.2.3 PWM End-of-Duty-Cycle Events

For channels configured for PWM operation there are two possibilities. When the channel is configured for edge-aligned PWM, the channel flag gets 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 which are the times when the timer counter matches the channel value register. The flag is cleared by the two-step sequence described [Section 22.6.2, “Description of Interrupt Operation.”](#)



# **Chapter 23**

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

### **23.1 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 23-1](#).

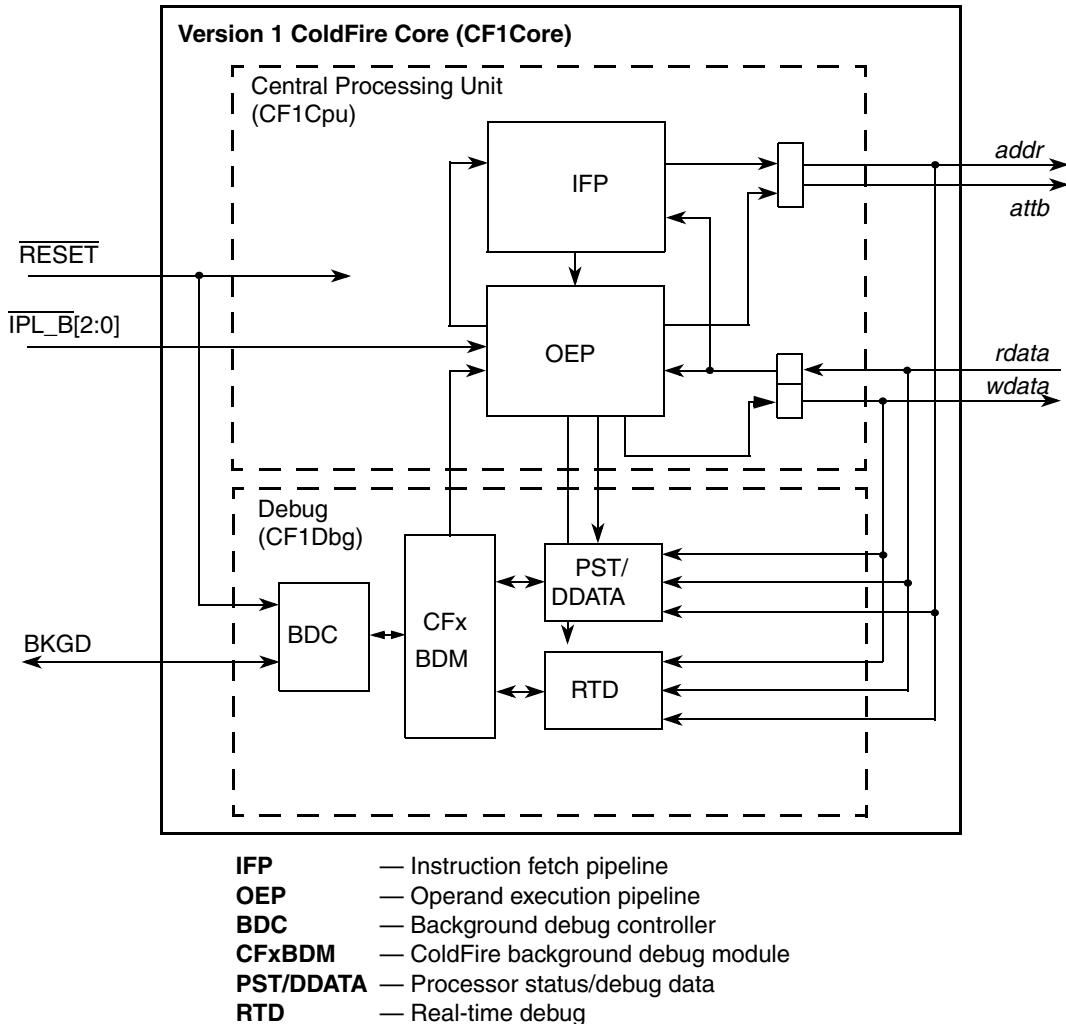


Figure 23-1. Simplified Version 1 ColdFire Core Block Diagram

### 23.1.1 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 23.4.1, “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 23.4.2, “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 23.4.3, “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 23-1](#) summarizes the various debug revisions.

**Table 23-1. 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

### 23.1.2 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
- Debug resources are accessible via single-pin BDM interface or the privileged WDEBUG instruction from the core

### 23.1.3 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 mode 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 23-2](#).

**Table 23-2. 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 23.4.1.5, “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 [Section 23.3.3, “Configuration/Status Register 2 \(CSR2\),”](#) for details)
- A background debug force reset occurs (CSR2[BDFR] is set) and CSR2[BFHBR] is set
- reset occurs and CSR2[HR] 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

While in halt mode, the core waits for serial background commands rather than executing instructions from the application program.

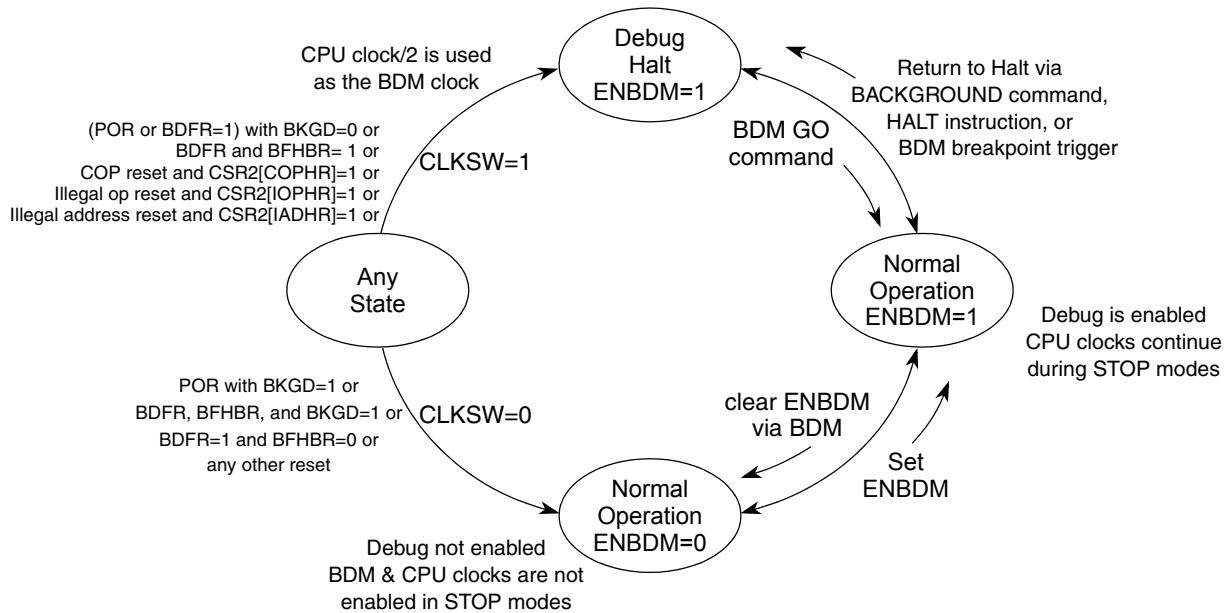
**Figure 23-2. Debug Modes State Transition Diagram**

Figure 23-2 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.

## 23.2 External Signal Descriptions

Table 23-3 describes the debug module's 1-pin external signal (BKGD). A standard 6-pin debug connector is shown in Section 23.4.4, “Freescale-Recommended BDM Pinout”.

**Table 23-3. 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.

## 23.3 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 23-4](#), 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 23.4.1.5, “BDM Command Set Summary.”](#) These commands contain a 5-bit field, DR<sub>c</sub>, that specifies the register, as shown in [Table 23-4](#).

**Table 23-4. Debug Module Memory Map**

DR <sub>c</sub>	Register Name	Width (bits)	Access	Reset Value	Section/ Page
0x00	Configuration/status register (CSR)	32	R/W (BDM), W (CPU)	0x0090_0000	<a href="#">23.3.1/23-7</a>
0x01	Extended Configuration/Status Register (XCSR)	32	R/W <sup>1</sup> (BDM), W (CPU)	0x0000_0000	<a href="#">23.3.2/23-10</a>
0x02	Configuration/Status Register 2 (CSR2)	32	R/W <sup>1</sup> (BDM), W (CPU)	See Section	<a href="#">23.3.3/23-13</a>
0x03	Configuration/Status Register 3 (CSR3)	32 <sup>2</sup>	R/W <sup>1</sup> (BDM), W (CPU)	0x0000_0000	<a href="#">23.3.4/23-16</a>
0x05	BDM address attribute register (BAAR)	32 <sup>2</sup>	W	0x0000_0005	<a href="#">23.3.5/23-17</a>
0x06	Address attribute trigger register (AATR)	32 <sup>2</sup>	W	0x0000_0005	<a href="#">23.3.6/23-18</a>
0x07	Trigger definition register (TDR)	32	W	0x0000_0000	<a href="#">23.3.7/23-19</a>
0x08	PC breakpoint register 0 (PBR0)	32	W	Undefined, Unaffected	<a href="#">23.3.8/23-22</a>
0x09	PC breakpoint mask register (PBMR)	32	W	Undefined, Unaffected	<a href="#">23.3.8/23-22</a>
0x0C	Address breakpoint high register (ABHR)	32	W	Undefined, Unaffected	<a href="#">23.3.9/23-24</a>
0x0D	Address breakpoint low register (ABLR)	32	W	0x0000_0000	<a href="#">23.3.9/23-24</a>
0x0E	Data breakpoint register (DBR)	32	W	0x0000_0000	<a href="#">23.3.10/23-25</a>
0x0F	Data breakpoint mask register (DBMR)	32	W	0x0000_0000	<a href="#">23.3.10/23-25</a>

**Table 23-4. 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="#">23.3.8/23-22</a>
0x1A	PC breakpoint register 2 (PBR2)	32	W	PBR2[0] = 0	<a href="#">23.3.8/23-22</a>
0x1B	PC breakpoint register 3 (PBR3)	32	W	PBR3[0] = 0	<a href="#">23.3.8/23-22</a>
—	PST Trace Buffer $n$ (PSTB $n$ ); $n = 0\text{--}11$	32	R (BDM) <sup>3</sup>	Undefined, Unaffected	<a href="#">23.4.3.2/23-60</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.

### 23.3.1 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
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	TRC	0	DDC	UHE	BTB		0	NPL	IPI	SSM		0	0	FID	DDH
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 23-3. Configuration/Status Register (CSR)

Table 23-5. 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. 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 the BKPT input was asserted or a BDM BACKGROUND command received, forcing 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 23-5. 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 23-26</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 23.4.3.1, "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 23-5. 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

### 23.3.2 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 23-6](#).

**Table 23-6. 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	0
W			ESEQC			ERASE										
Reset	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	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

Figure 23-4. Extended Configuration/Status Register (XCSR)

Table 23-7. 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.		
	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 23-7. 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 23-2</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 23-7. 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> $\text{PC address capture period} = \frac{2^{(\text{APCSC} + 1)} \times 1024}{16^{\text{APCDIV16}}} \quad \text{Eqn. 23-1}$ <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.</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>																																							

### 23.3.3 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 23-8](#).

**Table 23-8. 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 23-8. 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		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	0
	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 23-5. Configuration/Status Register 2 (CSR2)****Table 23-9. 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 HR	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-operatingproperly reset, the device immediately enters normal operation mode. 1 A computer-operatingproperly 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.
26	Reserved, must be cleared.

**Table 23-9. CSR2 Field Descriptions (continued)**

Field	Description						
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.						
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: auto; margin-right: auto;"> <thead> <tr> <th>PSTBWA[7]</th> <th>PSTB Valid Data Locations (Oldest to Newest)</th> </tr> </thead> <tbody> <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> </tbody> </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. 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						
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.						

**Table 23-9. CSR2 Field Descriptions (continued)**

Field	Description																										
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 Normal recording mode 01 10 11																										
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"> <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																									

### 23.3.4 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 23-10](#).

**Table 23-10. 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.
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

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	BFC	BFCDIV					0	0	0	0	0	0	0	0	0
W		DIV8														
Reset	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	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

Figure 23-6. Configuration/Status Register 3 (CSR3)

Table 23-11. 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.

### 23.3.5 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	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	0	1	0	1	

Figure 23-7. BDM Address Attribute Register (BAAR)

Table 23-12. 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 23.3.6, “Address Attribute Trigger Register (AATR)”</a> .
2–0 TM	Transfer modifier. See the TM definition in the AATR description, <a href="#">Section 23.3.6, “Address Attribute Trigger Register (AATR)”</a> .

### 23.3.6 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	RM	SZM	TTM	TMM	R	SZ	TT	TM													
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1		

Figure 23-8. Address Attribute Trigger Register (AATR)

**Table 23-13. 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

### 23.3.7 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 23-9. Trigger Definition Register (TDR)

Table 23-14. 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 23-14. 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 23-14. 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.																	

### 23.3.8 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

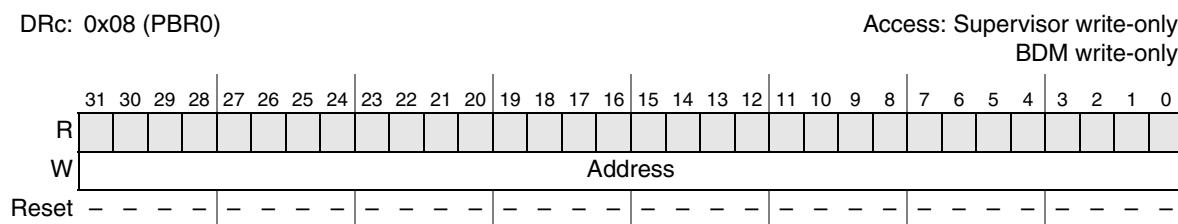
valid bit. These registers' contents are compared with the processor's program counter register when TDR is configured appropriately.

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 23.4.1.4, “BDM Command Set Descriptions”](#).

## **NOTE**

Version 1 ColdFire core devices implement a 24-bit, 16 MB address map.

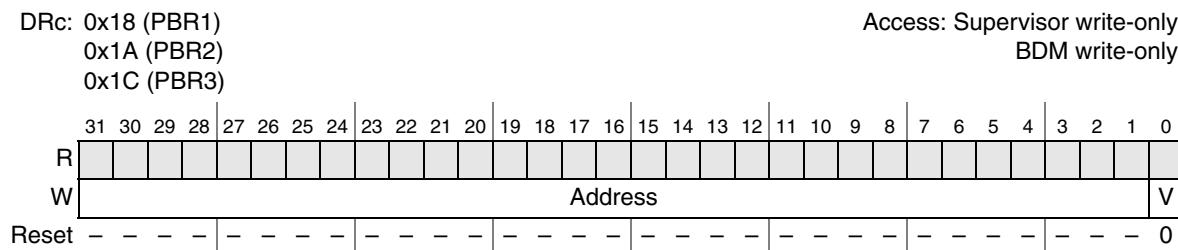
When programming these registers with a 32-bit address, the upper byte should be zero-filled when referencing the flash, RAM, and RGPIO regions, and set to 0xFF when referencing any of the slave peripheral devices.



**Figure 23-10. Program Counter Breakpoint Register 0 (PBR0)**

**Table 23-15. 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.

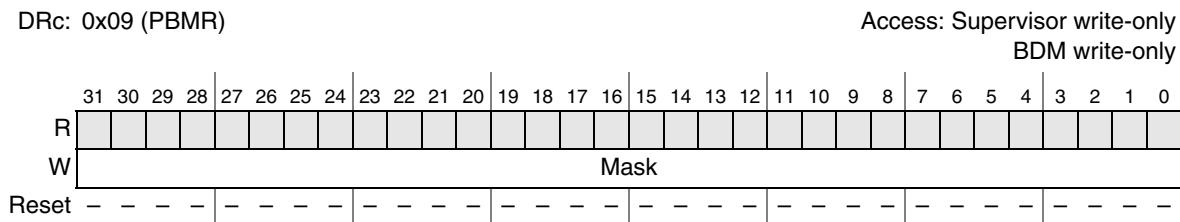


**Figure 23-11. Program Counter Breakpoint Register  $n$  (PBR $n$ ,  $n = 1,2,3$ )**

**Table 23-16. PBRn 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 23-12 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 23-12. Program Counter Breakpoint Mask Register (PBMR)**

**Table 23-17. 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.

### 23.3.9 Address Breakpoint Registers (ABLR, ABHR)

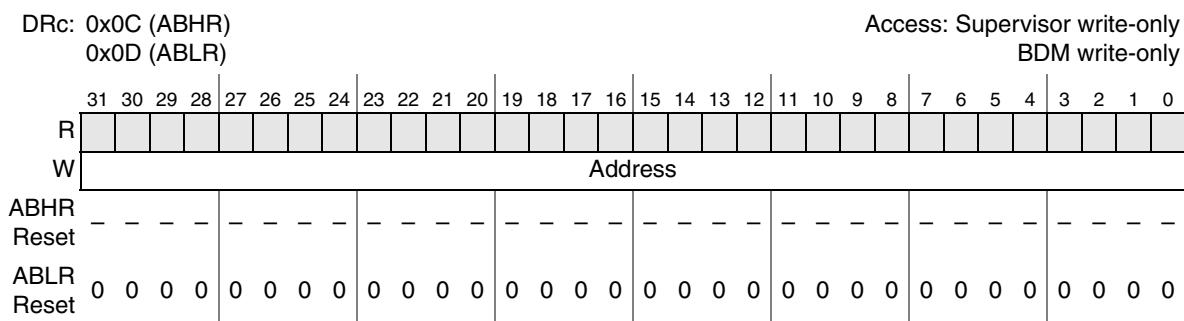
The ABLR and ABHR define regions in the processor's data address space that can be used as part of the trigger. These register values are compared with the address for each transfer on the processor's high-speed local bus. The trigger definition register (TDR) identifies the trigger as one of three cases:

- 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 23.4.1.4, “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 should be zero-filled when referencing the flash, RAM, and RGPIO regions and set to 0xFF when referencing any of the slave peripheral devices.



**Figure 23-13. Address Breakpoint Registers (ABLR, ABHR)**

**Table 23-18. 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.

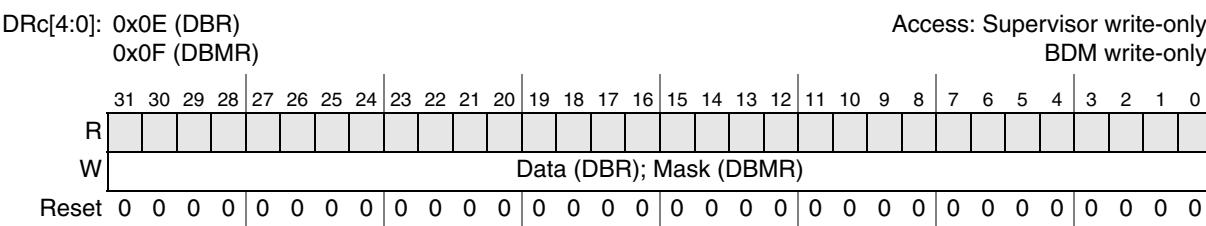
**Table 23-19. ABHR Field Description**

Field	Description
31–0 Address	High address. Holds the 32-bit address marking the upper bound of the address breakpoint range.

### 23.3.10 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 23-14. Data Breakpoint & Mask Registers (DBR, DBMR)****Table 23-20. 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 23-21. 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 23-22](#) shows the relationships between processor address, access size, and location within the 32-bit data bus.

**Table 23-22. 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]

### 23.3.11 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 23.3.3, “Configuration/Status Register 2 \(CSR2\)”](#).

## 23.4 Functional Description

### 23.4.1 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 23-1](#), 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.

### 23.4.1.1 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 23-23. 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.

**Table 23-23. CPU Halt Sources (continued)**

Halt Source	Halt Timing	Description		
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.
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).

**Table 23-23. CPU Halt Sources (continued)**

Halt Source	Halt Timing	Description	
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. The debug GO command also clears CSR[26–24].

### 23.4.1.2 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 23.4.1.3, “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 23.4.1.3, “BDM Communication Details,”](#) for more details.

When no debugger pod is connected to the standard 6-pin BDM interface connector ([Section 23.4.4, “Freescale-Recommended BDM Pinout”](#)), the internal pullup on BKGD chooses normal operating mode.

When a development system is connected, it can pull BKGD and RESET low, release 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].

### 23.4.1.3 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 23-7](#).

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.

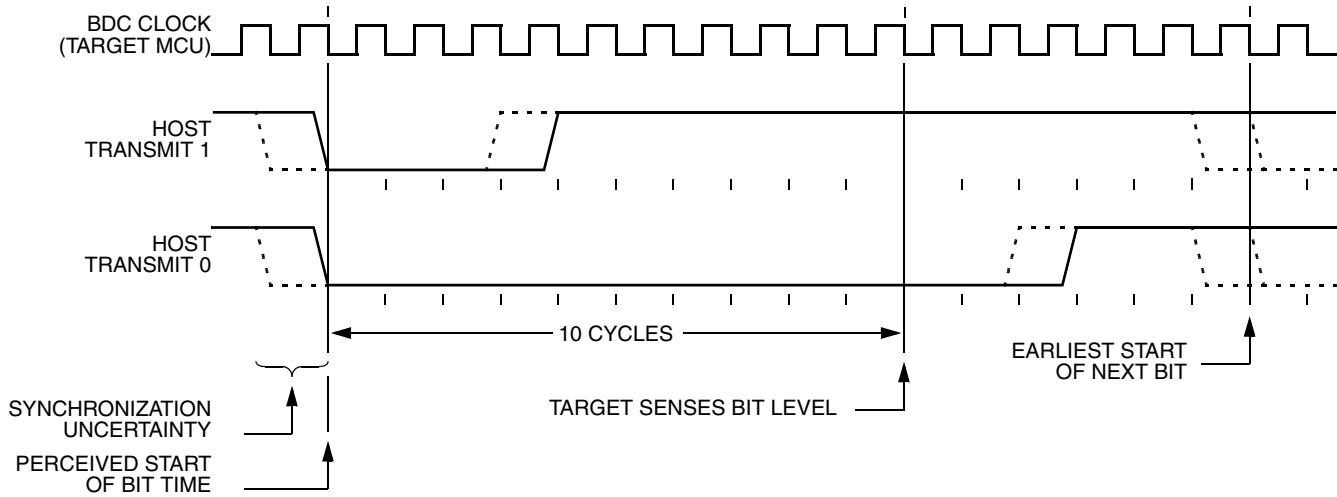
Normally, setting CLKSW should not be used for general debugging because there is no way to ensure the application program does not change the clock generator settings. This is especially true in the case of application programs that are not yet fully debugged.

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\_BYTEx 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 23-15](#) 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 23-15. BDC Host-to-Target Serial Bit Timing**

Figure 23-16 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.

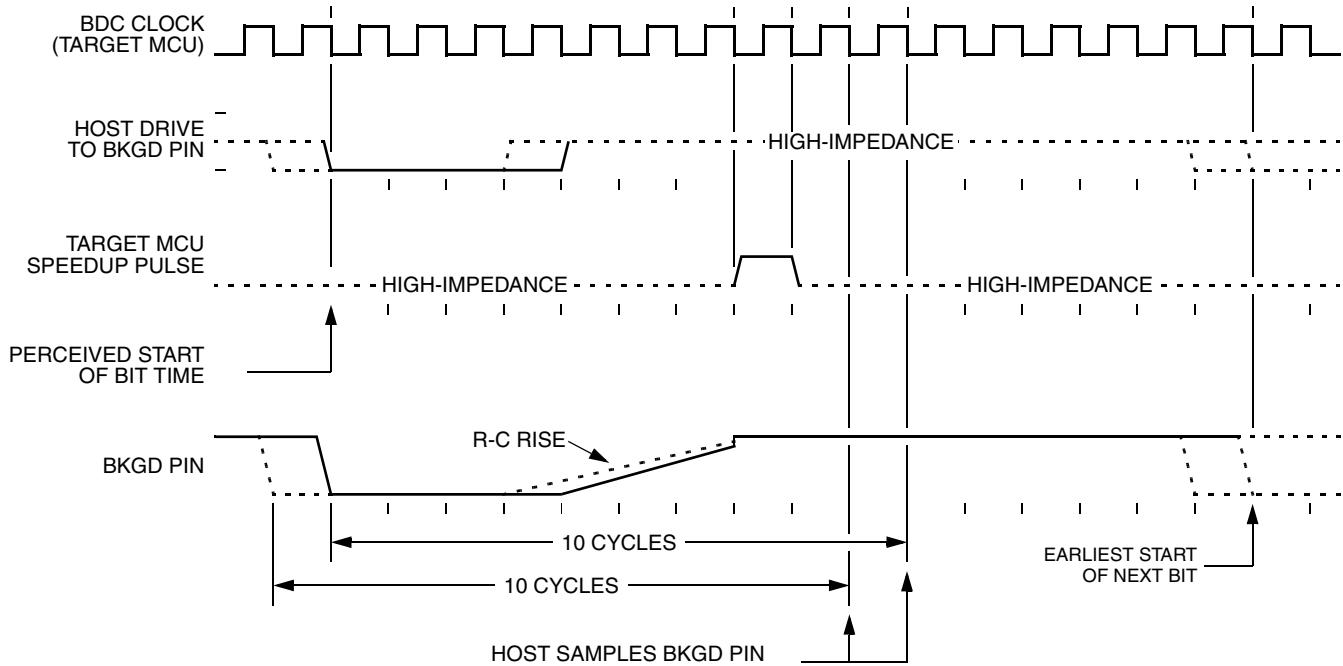


Figure 23-16. BDC Target-to-Host Serial Bit Timing (Logic 1)

Figure 23-17 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.

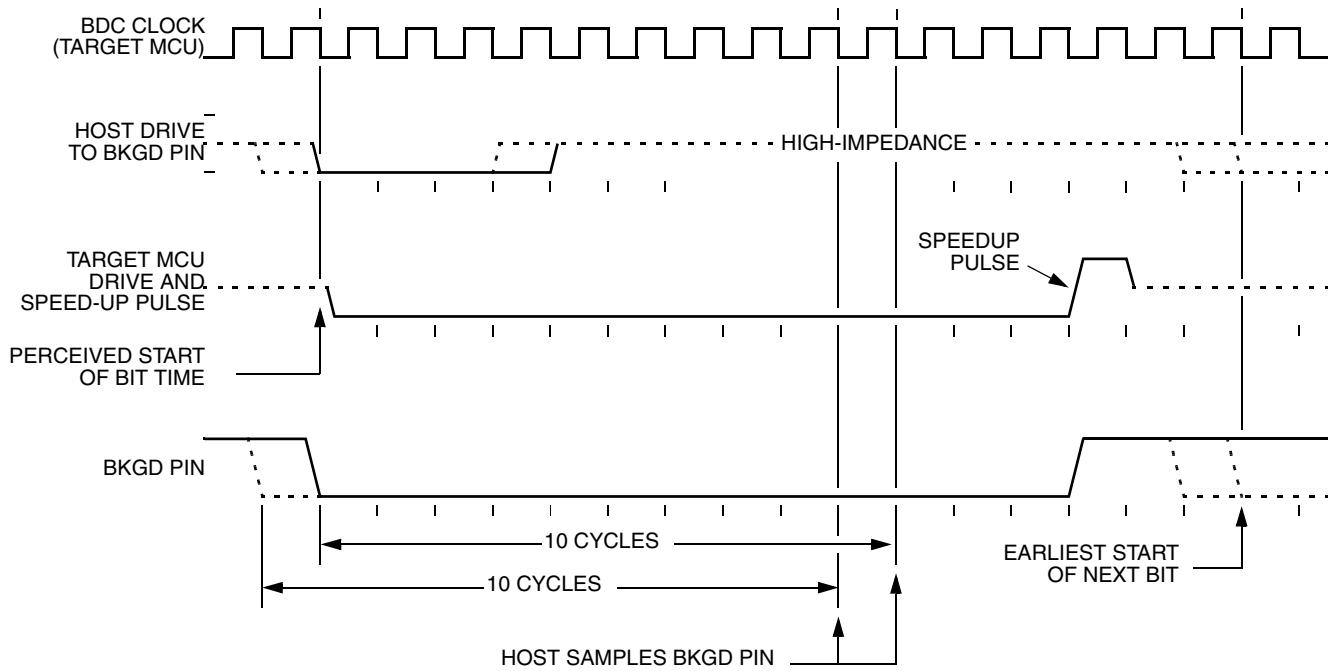


Figure 23-17. BDM Target-to-Host Serial Bit Timing (Logic 0)

#### 23.4.1.4 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 23-18](#).

**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		MCMD
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 23-19</a>				
R					Trace Buffer Data[23–16], see <a href="#">Figure 23-19</a>				
R					Trace Buffer Data[15–08], see <a href="#">Figure 23-19</a>				
R					Trace Buffer Data[07–00], see <a href="#">Figure 23-19</a>				

**Figure 23-18. BDM Command Encodings**

**Table 23-24. 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 DBG'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 23-4</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 23-4</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 23-4</a>																											
11	0x00	OTHER_A7																										
	0x01	VBR																										
	0x02	CPUCR																										
	0x0E	SR																										
	0x0F	PC																										

### 23.4.1.5 BDM Command Set Summary

[Table 23-25](#) summarizes the BDM command set. Subsequent paragraphs contain detailed descriptions of each command. The nomenclature below is used in [Table 23-25](#) 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 23-25. 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 23-25. 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>
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 23-25. 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 23.4.1.7, "Hardware Handshake Abort Procedure,"](#) 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.

### 23.4.1.5.1 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.

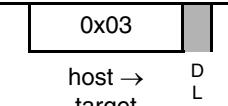
Upon detecting the sync request from the host (which is a much longer low time than would ever occur during normal BDC communications), the target:

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.

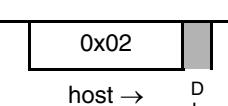
The host measures the low time of this 128-cycle sync response pulse and determines the correct speed for subsequent BDC communications. Typically, the host can determine the correct communication speed within a few percent of the actual target speed and the serial protocol can easily tolerate this speed error.

#### **23.4.1.5.2 ACK\_DISABLE**

<b>Disable host/target handshake protocol</b>	<b>Always Available</b>
 host →      D target      L Y	

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.

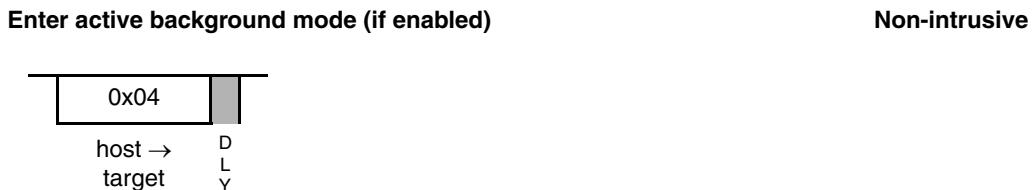
#### **23.4.1.5.3 ACK\_ENABLE**

<b>Enable host/target handshake protocol</b>	<b>Always Available</b>
 host →      D target      L Y	

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 23.4.1.6, “Serial Interface Hardware Handshake Protocol,”](#) and [Section 23.4.1.7, “Hardware Handshake Abort Procedure.”](#)

### 23.4.1.5.4 BACKGROUND

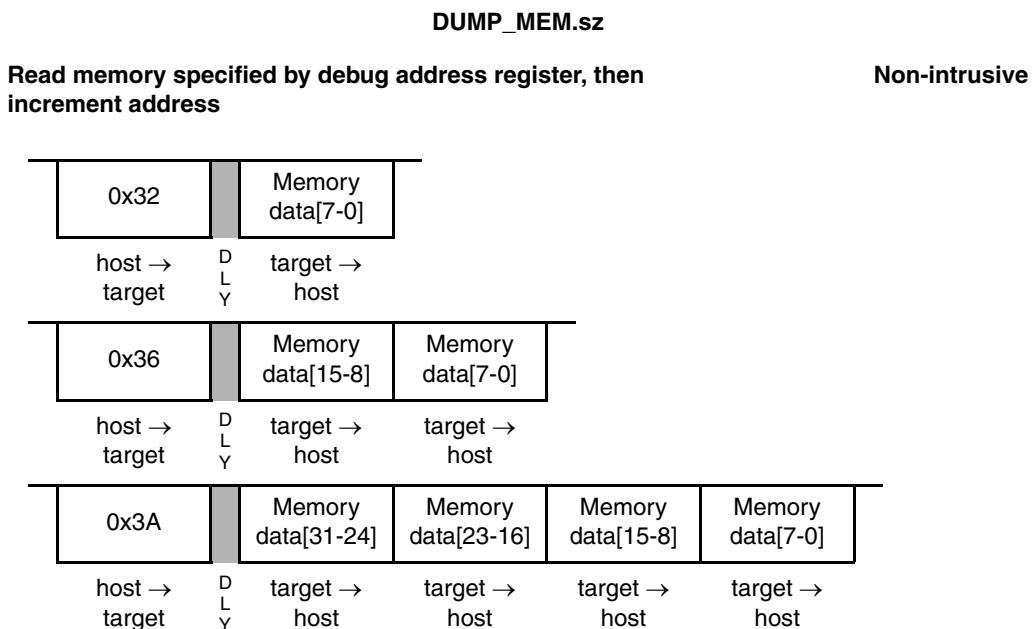


Provided XCSR[ENBDM] is set (BDM enabled), the BACKGROUND command causes the target MCU to enter active background (halt) mode as soon as the current CPU instruction finishes. If ENBDM is cleared (its default value), the BACKGROUND command is ignored.

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.

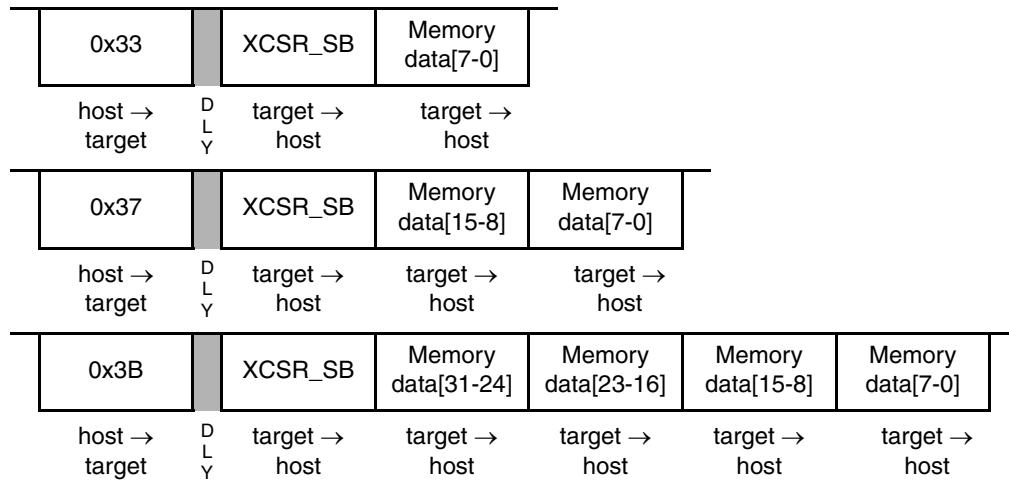
### 23.4.1.5.5 DUMP\_MEM.sz, DUMP\_MEM.sz\_WS



**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.

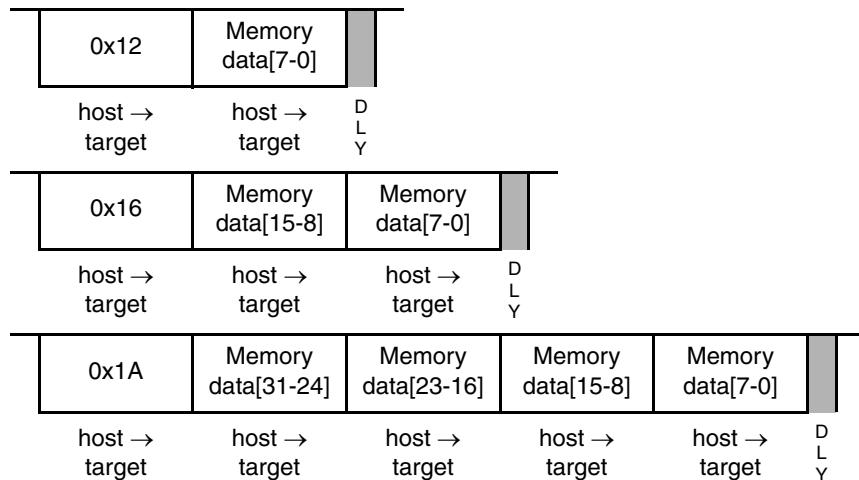
The size field (sz) is examined each time a DUMP\_MEM{\_WS} command is processed, allowing the operand size to be dynamically altered. The examples show the DUMP\_MEM.B{\_WS}, DUMP\_MEM.W{\_WS} and DUMP\_MEM.L{\_WS} commands.

### 23.4.1.5.6 FILL\_MEM.sz, FILL\_MEM.sz\_WS

#### FILL\_MEM.sz

**Write memory specified by debug address register, then increment address**

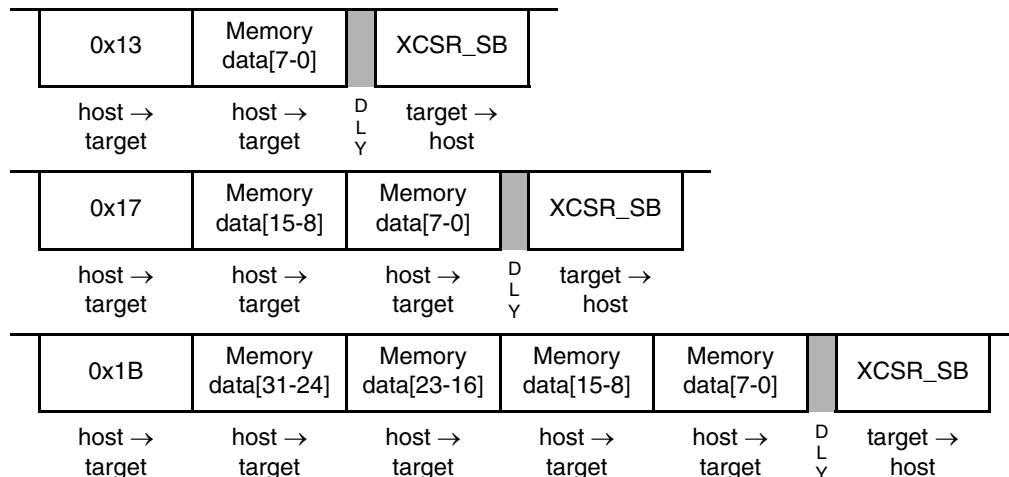
**Non-intrusive**



#### 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.

#### 23.4.1.5.7 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.

#### 23.4.1.5.8 NOP



NOP performs no operation and may be used as a null command where required.

### 23.4.1.5.9 READ\_CREG

Read CPU control register		Active Background			
0xE0+CRN		CREG data [31-24]	CREG data [23-16]	CREG data [15-8]	CREG data [7-0]
host → target	D L Y	target → host	target → host	target → host	target → host

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 23-24](#) 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.

### 23.4.1.5.10 READ\_DREG

Read debug control register		Non-intrusive			
0xA0+CRN		DREG data [31-24]	DREG data [23-16]	DREG data [15-8]	DREG data [7-0]
host → target	D L Y	target → host	target → host	target → host	target → host

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 23-4](#) for CRN details.

### 23.4.1.5.11 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

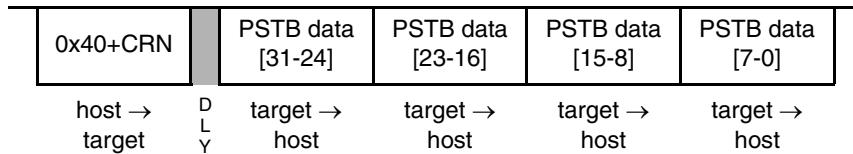
0x31	Address[23-0]	D L Y	XCSR_SB	Memory data[7-0]			
host → target	host → target		target → host	target → host			
0x35	Address[23-0]	D L Y	XCSR_SB	Memory data [15-8]	Memory data [7-0]		
host → target	host → target		target → host	target → host	target → host		
0x39	Address[23-0]	D L Y	XCSR_SB	Memory data[31-24]	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.

### 23.4.1.5.12 READ\_PSTB

**Read PST trace buffer at the specified address** Non-intrusive



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 23-19](#) for an illustration of how the buffer entries are packed.

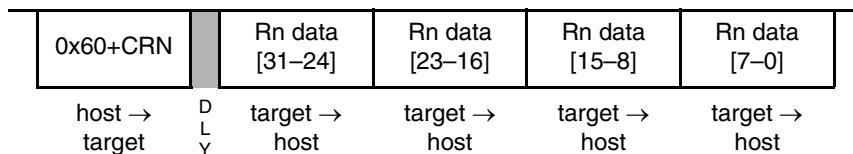
The write pointer for the trace buffer is available as CSR2[PSTBWA]. Using this pointer, it is possible to determine the oldest-to-newest entries in the trace buffer.

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				TB #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		TB #16		TB #17		TB #18		TB #19		TB #20		TB #21[5:4]											
0x13	TB #21[3:0]		TB #22		TB #23		TB #24		TB #25		TB #26[5:2]		TB #27		TB #28		TB #29		TB #30		TB #31		TB #32		TB #33		TB #34		TB #35		TB #36		37[5:4]	
0x14	TB #37[3:0]		TB #38		TB #39		TB #40		TB #41		TB #42[5:2]		TB #43		TB #44		TB #45		TB #46		TB #47		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]		TB #59		TB #60		TB #61		TB #62		TB #63		58[1:0]		59		60		61		62		63	

**Figure 23-19. PST Trace Buffer Entries and Locations**

### 23.4.1.5.13 READ\_Rn

**Read general-purpose CPU register** Active Background



If the processor is halted, this command reads the selected CPU general-purpose register (An, Dn) and returns the 32-bit result. See [Table 23-24](#) for the CRN details when CRG is 01.

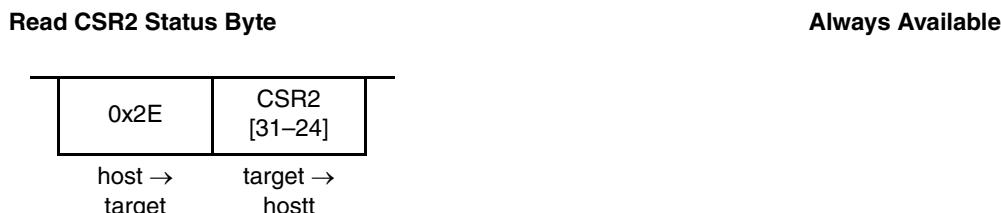
If the processor is not halted, this command is rejected as an illegal operation and no operation is performed.

#### 23.4.1.5.14 READ\_XCSR\_BYTE



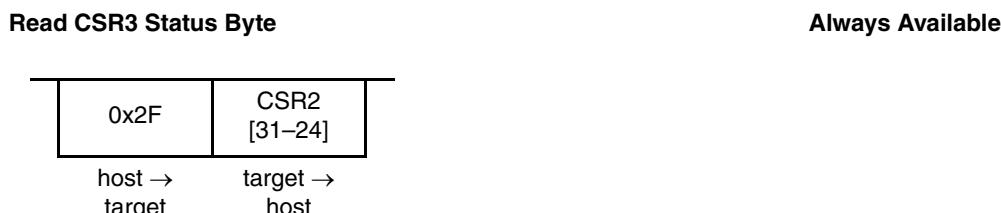
Read the special status byte of XCSR (XCSR[31–24]). This command can be executed in any mode.

#### 23.4.1.5.15 READ\_CSR2\_BYTE



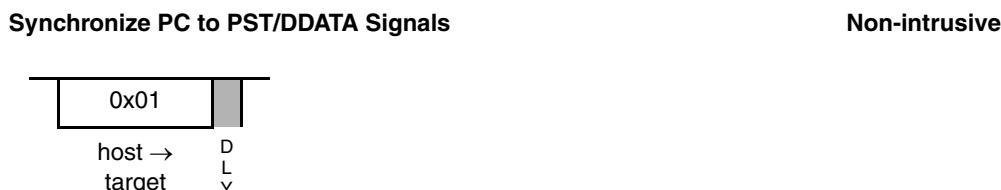
Read the most significant byte of CSR2 (CSR2[31–24]). This command can be executed in any mode.

#### 23.4.1.5.16 READ\_CSR3\_BYTE



Read the most significant byte of the CSR3 (CSR3[31–24]). This command can be executed in any mode.

#### 23.4.1.5.17 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.

#### 23.4.1.5.18 WRITE\_CREG

Write CPU control register					Active Background
0xC0+CRN	CREG data [31–24]	CREG data [23–16]	CREG data [15–8]	CREG data [7–0]	
host → target	host → target	host → target	host → target	host → target	D L Y

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 23-24](#) 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.

#### 23.4.1.5.19 WRITE\_DREG

Write debug control register					Non-intrusive
0x80+CRN	DREG data [31–24]	DREG data [23–16]	DREG data [15–8]	DREG data [7–0]	
host → target	host → target	host → target	host → target	host → target	D L Y

This command writes the 32-bit operand to the selected debug control register. This grouping includes all the debug control registers ({X}CSRn, BAAR, AATR, TDR, PBRn, 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 23-4](#) for CRN details.

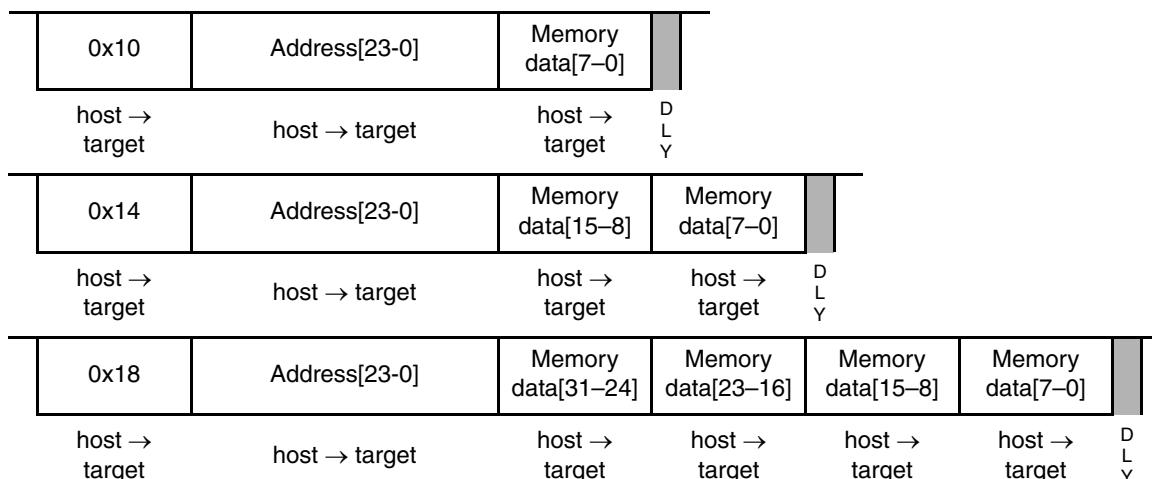
### NOTE

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\_BYTE, WRITE\_CSR2\_BYTE, and WRITE\_CSR3\_BYTE commands.

#### 23.4.1.5.20 WRITE\_MEM.sz, WRITE\_MEM.sz\_WS

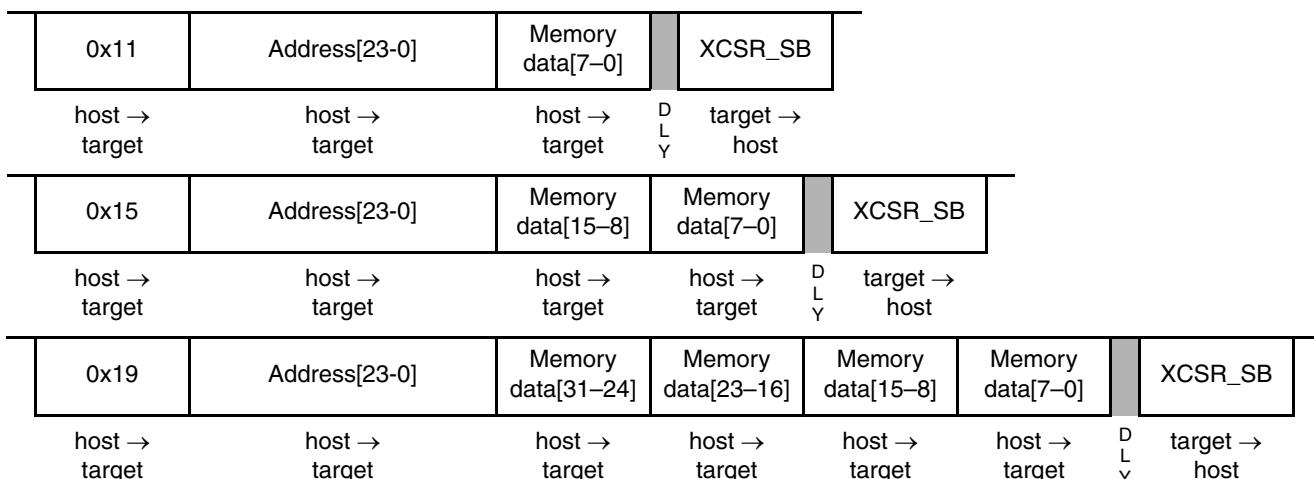
##### WRITE\_MEM.sz

**Write memory at the specified address** Non-intrusive



##### WRITE\_MEM.sz\_WS

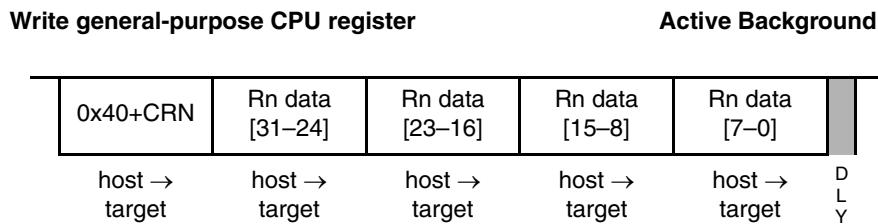
**Write memory at the specified address with status** Non-intrusive



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 read was performed.

The examples show the WRITE\_MEM.B{\_WS}, WRITE\_MEM.W{\_WS}, and WRITE\_MEM.L{\_WS} commands.

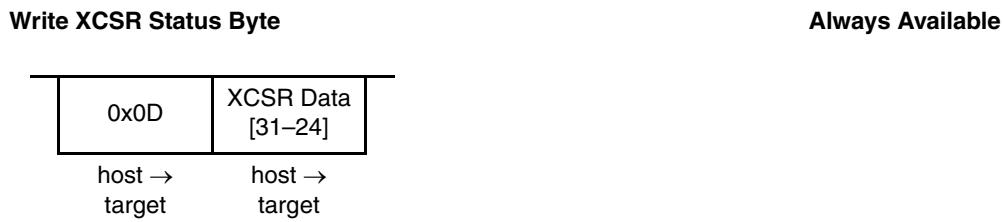
#### 23.4.1.5.21 WRITE\_Rn



If the processor is halted, this command writes the 32-bit operand to the selected CPU general-purpose register (An, Dn). See [Table 23-24](#) for the CRN details when CRG is 01.

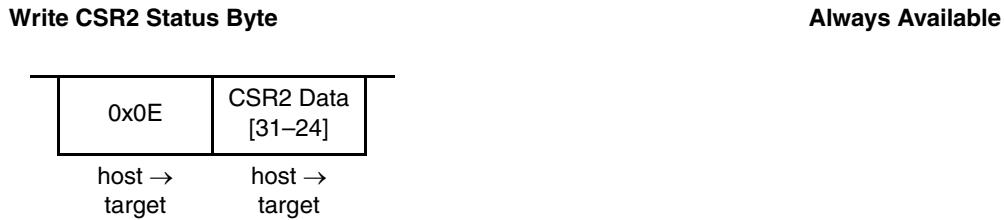
If the processor is not halted, this command is rejected as an illegal operation and no operation is performed.

#### 23.4.1.5.22 WRITE\_XCSR\_BYTE



Write the special status byte of XCSR (XCSR[31–24]). This command can be executed in any mode.

#### 23.4.1.5.23 WRITE\_CSR2\_BYTE



Write the most significant byte of CSR2 (CSR2[31–24]). This command can be executed in any mode.

### 23.4.1.5.24 WRITE\_CSR3\_BYTE

Write CSR3 Status Byte	Always Available		
<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td style="text-align: center;">0x0F</td> <td style="text-align: center;">CSR3 Data [31–24]</td> </tr> </table> <p style="text-align: center;">host →      host → target      target</p>	0x0F	CSR3 Data [31–24]	
0x0F	CSR3 Data [31–24]		

Write the most significant byte of CSR3 (CSR3[31–24]). This command can be executed in any mode.

### 23.4.1.6 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 23-20](#). 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.

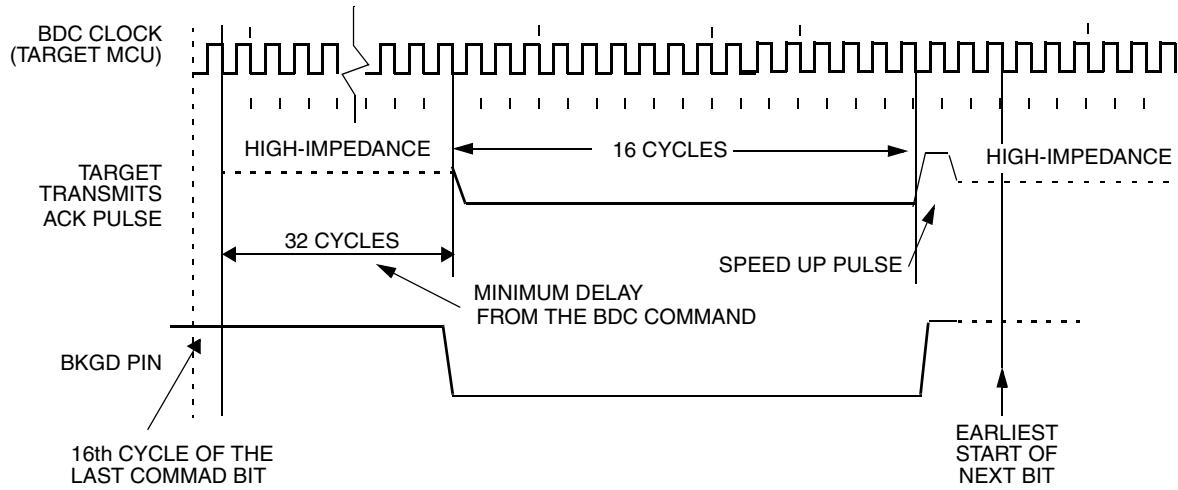


Figure 23-20. Target Acknowledge Pulse (ACK)

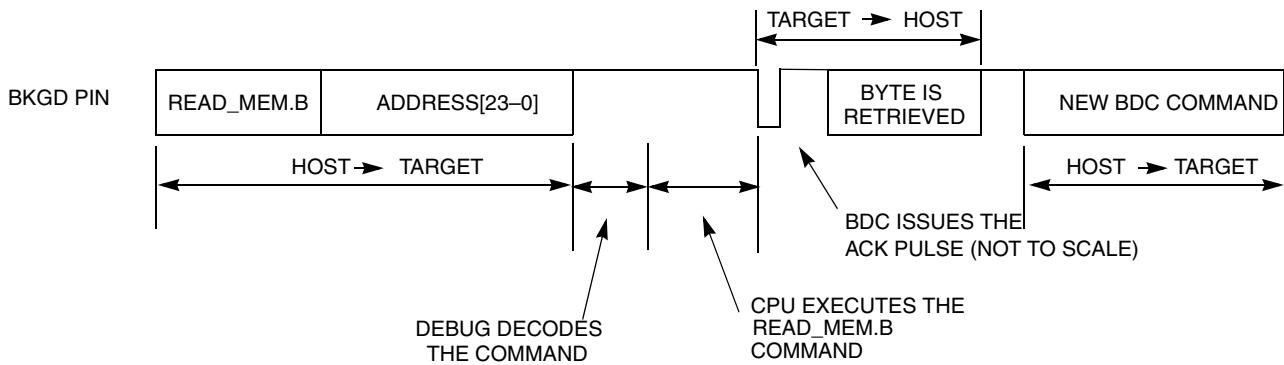
**NOTE**

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 23-21 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 23-21. 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 23-21](#) 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 23.4.1.7, “Hardware Handshake Abort Procedure.”](#)

### 23.4.1.7 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 23.4.1.5.1, “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\_BYTE 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 23-22](#) 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 23-22](#) signal timing is not drawn to scale.

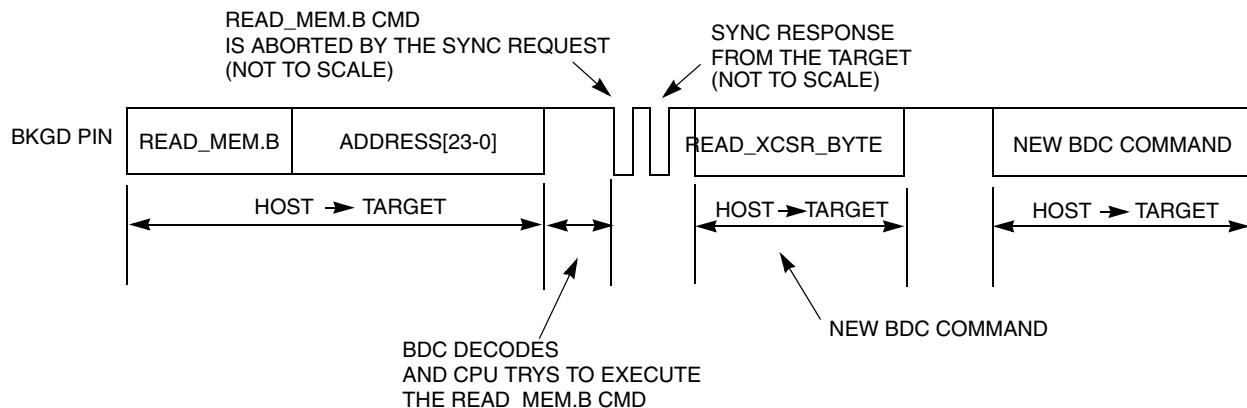


Figure 23-22. ACK Abort Procedure at the Command Level

Figure 23-23 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.

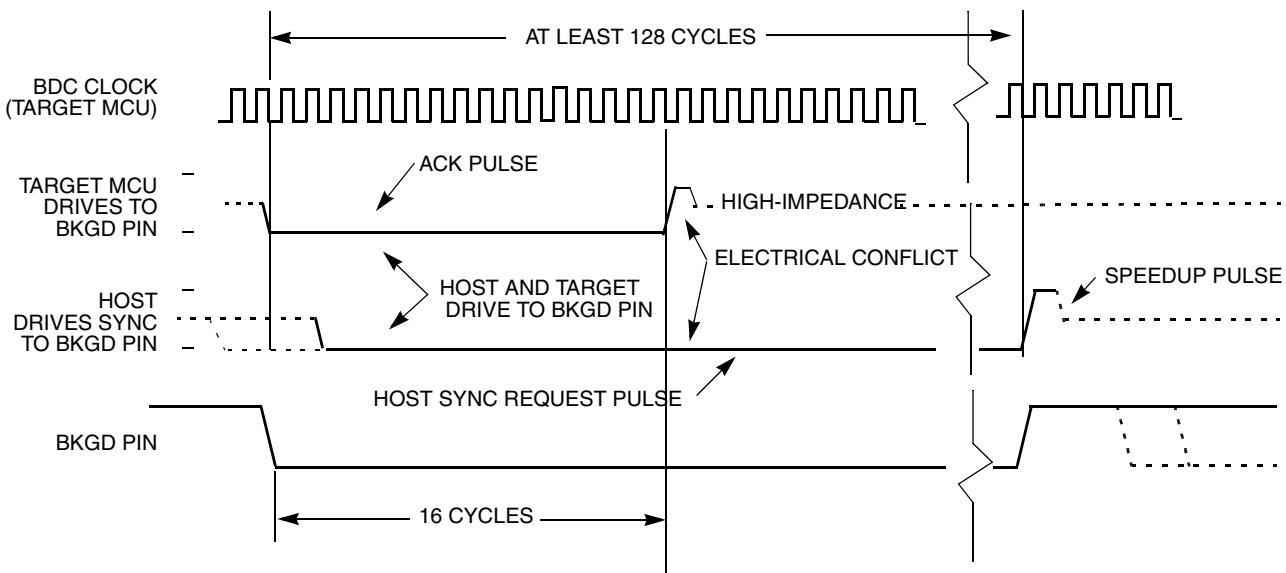


Figure 23-23. ACK Pulse and SYNC Request Conflict

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 23-25](#) 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.

### 23.4.2 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.

### 23.4.3 Trace Support

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.

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.

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. Luckily, the PST compression technology was previously developed and included as part of the Version 5 ColdFire core (although different than the resulting V1 implementation).

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 ≠ 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 23-26](#).

**Table 23-26. 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.

**Table 23-26. CF1 Debug Processor Status Encodings (continued)**

PST[4:0]	Definition
0x08–0x0B	Indicates the number of data bytes to be displayed as DDATA on subsequent processor clock cycles. This marker value is driven as the PST one processor clock cycle before the data is displayed on DDATA. 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 displayed as DDATA on subsequent processor clock cycles. This marker value is driven as the PST one processor clock cycle before the address is displayed on DDATA. 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
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.

### 23.4.3.1 Begin Execution of Taken Branch (PST = 0x05)

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, which 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 would be a JMP (A0) instruction. [Figure 23-24](#) shows the PSTB entries that indicate a JMP (A0) execution, assuming the CSR 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 23-24. Example JMP Instruction Output in PSTB**

PST of 0x05 indicates a taken branch and the marker value 0x0D indicates a 2-byte address. Thus, 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 23.4.3.2, “PST Trace Buffer \(PSTB\),”](#) for entry descriptions explaining the 2-bit prefix before each address nibble.

### 23.4.3.2 PST Trace Buffer (PSTB)

As PST and DDATA values are captured and loaded in the trace buffer, each entry is six bits in size so the type of the entry can easily be determined when post-processing the PSTB. See [Figure 23-25](#).

	5	4	3	2	1	0
PSTB[PST]	0			PST[4:0]		
Data PSTB[DDATA]	1	R/W			Data[3:0]	
Address PSTB[DDATA]	1	0			Address[3:0] <sup>1</sup>	
Reset:	—	—	—	—	—	—

<sup>1</sup> Depending on which nibble is displayed (as determined by CSR[9:8]), address[3:0] sequentially displays four bits of the real CPU address [16:1] or [24:17].

**Figure 23-25. V1 PST/DDATA Trace Buffer Entry Format**

### 23.4.3.3 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, then negates the IRQ, and performs a software IACK and then exits. This example is presented here because it exercises a considerable set of the PST/DDATA capabilities.

```

_isr:
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    IGCR0.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 the display of 2-byte branch addresses. Peripheral bus read and write operands are being traced. 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    IGCR0.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.

### 23.4.3.4 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:

$$\text{PST} = 0x01, \{\text{PST} = 0x0[89B], \text{DDATA} = \text{operand}\}$$

where the {...} definition is optional operand information defined by the setting of the CSR, and [...] indicates 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.

#### 23.4.3.4.1 User Instruction Set

[Table 23-27](#) 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 23-27. 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}
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}
bclr.{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}

**Table 23-27. PST/DDATA Specification for User-Mode Instructions (continued)**

<b>Instruction</b>	<b>Operand Syntax</b>	<b>PST/DDATA</b>
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}
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

**Table 23-27. PST/DDATA Specification for User-Mode Instructions (continued)**

<b>Instruction</b>	<b>Operand Syntax</b>	<b>PST/DDATA</b>
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
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}

**Table 23-27. PST/DDATA Specification for User-Mode Instructions (continued)**

Instruction	Operand Syntax	PST/DDATA
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).

### 23.4.3.4.2 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 23-28](#).

**Table 23-28. PST/DDATA Specification for Supervisor-Mode Instructions**

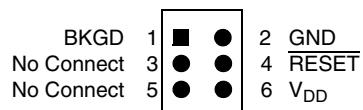
Instruction	Operand Syntax	PST/DDATA
halt		PST = 0x1F, PST = 0x1F
move.l	Ay,USP	PST = 0x01
move.l	USP,Ax	PST = 0x01
move.w	SR,Dx	PST = 0x01
move.w	{Dy,#<data>},SR	PST = 0x01, {PST = 0x03}
movec.l	Ry,Rc	PST = 0x01
rte		PST = 0x07, {PST = 0x0B, DD = source operand}, {PST = 0x03}, {PST = 0x0B, DD = source 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.

### 23.4.4 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, RESET, and sometimes V<sub>DD</sub>. 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<sub>DD</sub> 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 23-26. Recommended BDM Connector**



# Appendix A

## Revision History

This appendix lists major changes between versions of the MCF51JM128RM document.

### A.1 Changes Between Rev. 1 and Rev. 2

Table A-1. MCF51JM128 Rev 1 to Rev. 2 Changes

Chapter	Description
Overview	Updated the table <b>MCF51JM128 Series Package Availability</b>
Pins and Connections	Updated the figure <b>Basic System Connections</b> Updated the section <b>Power</b>
Mode	Updated the section Table 3-1 and 3-3
Memory	Updated the figure <b>MCF51JM128 Series Memory Maps</b> Updated Figure 4-4 and 4-6 Updated the second-level section <b>Security</b>
Resets, Interrupts, and General System Control	Updated the tables <b>ColdFire Vector Exception Table</b> and <b>ColdFire V1[Level][Priority within Level] Matrix Interrupt Assignments</b>
Version 1 ColdFire Interrupt Controller (CF1_INTC)	Updated the table <b>INTC_SFRC Field Descriptions</b> Updated the table <b>INTC_CFRC Field Descriptions</b>
Universal Serial Bus, OTG Capable Controller	In the section <b>Frame Number Register Low/High (FRM_NUML, FRM_NUMH)</b> , updated the introductory paragraph

