Xtrinsic FXLC95000CL Intelligent Motion-Sensing Platform

Hardware Reference Manual

Document Number: FXLC95000CLHWRM
Rev 0.6, May 2013

freescale
Xtrinsic

Preliminary

Contents

Section number **Title** **Page**

Chapter 1
About This Document

Chapter 2
Introduction

Chapter 3
Pins and Connections (PINS)

Chapter 4
Memory Maps (MEMORY)

Chapter 5
Slave Interface (SLAVE)

Chapter 6
Inter-Integrated Circuit (I2C)

Chapter 7
Queued Serial Peripheral Interface (QSPI)

Chapter 8
Operational Phases and Modes of Operation (OP_PHASES)

Chapter 9
System Integration Module (SIM)

Chapter 10
Analog Front End (AFE)

Chapter 11
On-Chip Oscillator (CLKGEN)

Chapter 12
Flash Memory Controller (FLASH)

Chapter 13
Port Controls (PC)

Chapter 14
Rapid General Purpose Input/Output Module (RGPIO)

Chapter 15
Pin Interrupt Function (IRQ)

Chapter 16
Read Only Memory (ROM)

Chapter 17
ColdFire v1 Interrupt Controller (CF1_INTC)

Chapter 18
Programmable Delay Block (PDB)

Chapter 19
Modulo Timer 16-Bit (MTIM16)

Chapter 20
Timer/Pulse-Width Modulator (TPM)

Chapter 21
ColdFire v1 Core (CF1_CORE)

Chapter 22
ColdFire v1 Debug (CF1_DEBUG)

Chapter 1
About This Document

1.1 Purpose

This reference manual describes the features, architecture, and programming model of the FXLC95000CL device, an intelligent, motion-sensing platform, which encompasses a three-axis accelerometer and a ColdFire V1 MCU.

1.2 Audience

This document is primarily for system architects and software application developers who are using or considering use of the FXLC95000CL in a system.

1.3 Terminology and Conventions

This section defines the terminology, abbreviations, and other conventions used throughout this document.

1.3.1 Terminology

Term	Meaning
ADC	Analog-to-Digital Converter
BCD	Binary Coded Decimal
BDM	Background Debug Module
CI	Command Interpreter
CSR	ColdFire Configuration Status Register
FOPT	Flash Options Register

Table continues on the next page...

Term	Meaning
hash	A cryptographic hash function is a deterministic procedure that takes an arbitrary block of data and returns a fixed-size bit string, the (cryptographic) hash value, such that an accidental or intentional change to the data will change the hash value.
PC	Program Counter
POR	Power-on-Reset
shared secret	In cryptography, a shared secret is a piece of data only known to the parties involved in a secure communication. The shared secret can be a password, a passphrase, a big number or an array of randomly chosen bytes.
SSP	Supervisor Stack Pointer
VBR	Vector Base Register. A register in the ColdFire memory map which controls the location of the exception vector table.

1.3.2 Conventions

Notational conventions are:

cleared/set	When a bit takes the value 0, it is said to be cleared; when it takes a value of 1, it is said to be set.
MNEMONICS	Mnemonics which may represent command names, defined macros, constants, enumeration values are shown as, for example, `CI_DEV_INFO`.
`programming domain entity`	Entities such as functions, data structures are shown as, for example `device_info_t`.
0x	Prefix to denote a hexadecimal number
h	Suffix to denote a hexadecimal number
nibble	A 4-bit data unit
byte	An 8-bit data unit
word	A 16-bit data unit
longword	A 32-bit data unit

CAUTION, *NOTE*, and *TIP* statements are used in this manual to emphasize critical, important, and useful information. The statements are defined below.

CAUTION

A CAUTION statement indicates a situation that could have unexpected or undesirable side effects or could be dangerous to the deployed application or system.

Note

A NOTE statement is used to point out important information.

Tip

A Tip statement is used to point out useful information.

1.3.3 Register figure conventions

The conventions for the register reset values are:

--	The bit is unused or reserved. It is undefined at reset and could be read as 1 or 0. When writing to this bit, write a 0 unless otherwise specified.
u	The bit is unaffected by reset.
[*signal_name*]	Reset value is determined by the polarity of the indicated signal.

The following register fields are used:

R	0		Field shading indicates a reserved bit field in a memory-mapped register and always read as 0.
W			

R	1		Field shading indicates a reserved bit field in a memory-mapped register and always read as 1.
W			

R	FIELDNAME		Indicates a read/write bit.
W			

R	FIELDNAME		Field shading indicates a read-only bit field in a memory-mapped register.
W			

R			Field shading indicates a write-only bit field in a memory-mapped register.
W	FIELDNAME		

R	FIELDNAME		Write 1 to clear: indicates that writing a 1 to this bit field clears it.
W	w1c		

R	0		Indicates a self-clearing bit.
W	FIELDNAME		

1.4 Related Documentation

The FXLC95000CL device's features and operations are described in a variety of reference manuals, user guides, and application notes.

To find the most-current versions of these documents:

1. Go to the Freescale homepage at freescale.com.
2. In the Keyword search box at the top of the page, enter the device number FXLC95000CL.
3. In the Refine Your Result pane on the left, click on the Documentation link.

1.5 Referenced documents

The following documents are referenced in this reference manual:

1. The I2C-Bus Specification Version 2.1, January 2000, Philips Semiconductors
2. ColdFire Family Programmer's Reference Manual, Freescale Semiconductor, CFPRM Rev. 3, 02/2005 http://www.freescale.com/files/dsp/doc/ref_manual/CFPRM.pdf
3. Version 1 ColdFire White Paper, Freescale Semiconductor, Document V1CFWP Rev.0, 02/2006 http://freescale.com/files/microcontrollers/doc/white_paper/V1CFWP.pdf
4. Wikipedia entry for "Semaphore": http://en.wikipedia.org/wiki/Semaphore_(programming).html.
5. Wikipedia entry for "Cryptographic hash function" last modified Dec. 12, 2011, http://en.wikipedia.org/wiki/Cryptographic_hash_function.html.
6. *ITU-T V.41 Recommendation: Code-Independent Error Control System,* available at http://www.itu.int/publications/index.html.
7. *ITU-T X.25 Recommendation: Interface between Data Terminal Equipment (DTE) and Data Circuit-terminating Equipment (DCE) for terminals operating in the packet mode and connected to public data networks by dedicated circuit,* available at http://www.itu.int/publications/index.html.
8. *ITU-T T.30 Recommendation: Procedures for document facsimile transmission in the general switched telephone network,* available at http://www.itu.int/publications/index.html.

Chapter 2
Introduction

2.1 Overview

The FXLC95000CL Intelligent, Motion-Sensing Platform is a breakthrough device with the integration of a 3-axis MEMS accelerometer and a 32-bit ColdFire MCU that enables autonomous, high-precision sensing solutions with local computing and sensors management capability in an open, easy to use, architecture.

The FXLC95000CL hardware is user-programmable to create an intelligent high-precision, flexible, motion-sensing platform. The user's firmware, together with the hardware device, can make system-level decisions required for sophisticated applications, such as gesture recognition, pedometer, and e-compass tilt compensation and calibration.

The FXLC95000 platform can act as an intelligent sensing hub and a highly configurable decision engine. Using the Master I^2C or SPI module, the FXLC95000 platform can manage secondary sensors such as pressure sensors, magnetometers, and gyroscopes. The embedded microcontroller allows sensor integration, initialization, calibration, data compensation, and computation functions to be added to the platform, thereby off-loading those functions from the host processor. Total system power consumption is significantly reduced because the application processor stays powered down for longer periods of time.

The FXLC95000CL device is programmed and configured with CodeWarrior Development Studio for Microcontroller (Eclipse IDE). This standard, integrated development environment (IDE) enables customers to quickly implement custom embedded algorithms and features to exactly match their application needs.

The following figure shows the salient components of FXLC95000CL, as well as the internal buses, clock domains and signals. Each block is labeled with a number—the table below provides links to each block's topics.

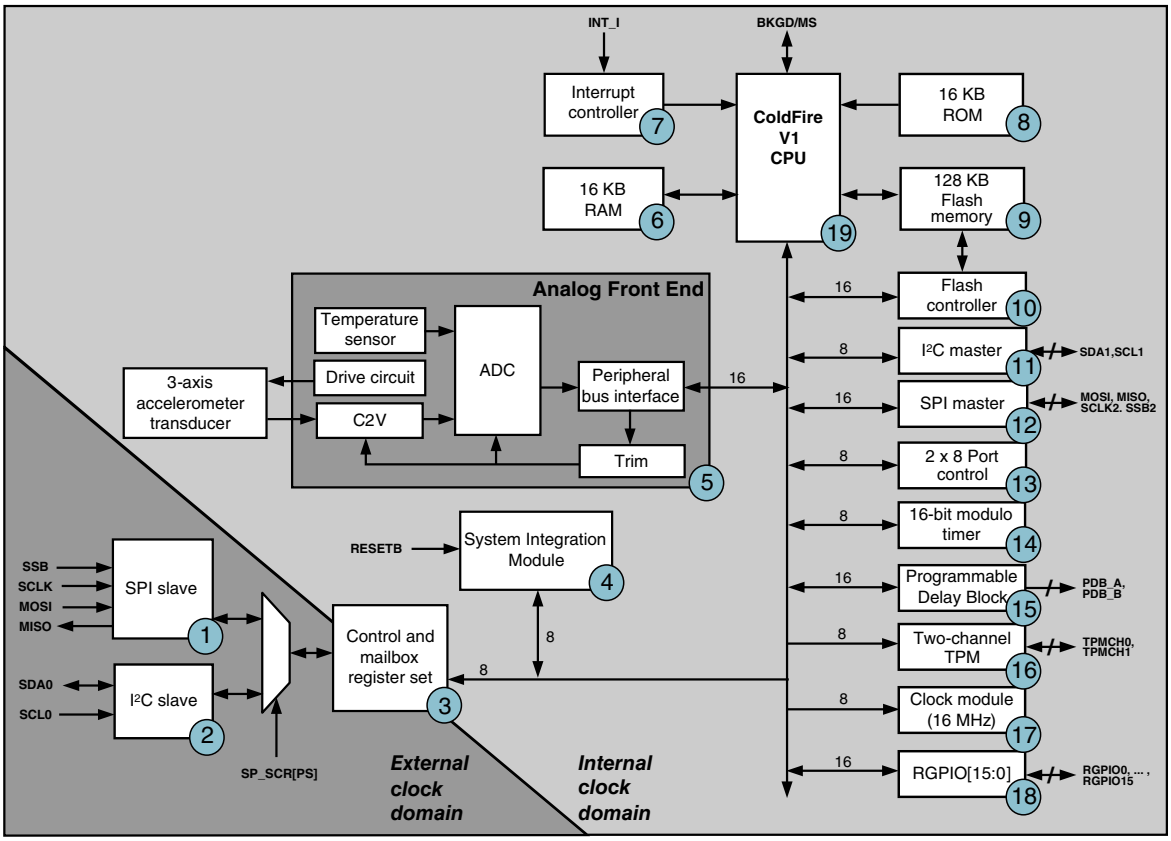


Figure 2-1. Block diagram of the FXLC95000CL

Block Diagram Number	Name	Topics
1	SPI slave	SPI features and limitations
2	I²C slave	I²C features and limitations
3	Control and mailbox register set	• Slave memory map and register definition • Oscillator control • Operating mode control
4	System integration module	System Integration Module (SIM) overview
5	Analog Front End	AFE architecture and theory of operation
6	16 KB RAM	RAM addressing and data
7	Interrupt controller	ColdFire v1 Interrupt Controller Introduction
8	16 KB ROM	Boot ROM
9	128 KB Flash memory	Flash memory overview
10	Flash controller	FLASH memory map and register descriptions
11	I²C master	Inter-Integrated Circuit (I²C) Introduction
12	SPI master	Queued Serial Peripheral Interface (QSPI) Introduction
13	2 x 8 Port control	FXLC95000CL port control customizations
14	16-bit modulo timer	Modulo Timer 16-Bit (MTIM16) Introduction
15	Programmable delay block	Programmable Delay Block (PDB) Introduction

Table continues on the next page...

Block Diagram Number	Name	Topics
16	Two-channel TPM	Timer/Pulse-Width Modulator (TPM) Introduction
17	Clock module (16 MHz)	On-Chip Oscillator (CLKGEN) Introduction
18	RGPIO[15:0]	Rapid General Purpose Input/Output Module (RGPIO) Introduction

2.2 Hardware features

The hardware features of the FXLC95000CL include the following:

- Three accelerometer operating ranges:
  - ±2 *g*: Suits most user-interaction (mouse) motions and free fall.
  - ±4 *g*: Covers most regular human dynamics (walking, jogging).
  - ±8 *g*: Detects most abrupt activities (gaming).
- Integrated temperature sensor.
- Master SPI and I^2C interfaces operating at rates up to 4 Mbit/s and 400 Kbit/s, respectively, to communicate with external sensors.
- Slave SPI and I^2C interfaces operating at rates up to 2 Mbit/s dedicated to communication with a host processor. Default value of the seven-bit I^2C address is 0x4C and can be customized by user firmware.
- 10-, 12-, 14-, and 16-bit trimmed analog-to-digital converter (ADC) data formats available.
- 1.8 V core and analog supply voltage.
- 1.71 V to 3.6 V I/O supply.
- 32-bit ColdFire V1 central processing unit (CPU) with multiply-accumulate (MAC) unit.
- Extensive set of power-management features and low-power modes.
- Single-wire background debug mode (BDM) pin interface.
- 128 KB (32 KB x 4) flash memory.
- 16 KB (4 KB x 4) random access memory (RAM).
- ROM-based flash controller and slave port, command line interpreter.
- Two channel timer with input capture, output capture, or edge-aligned (or center-aligned) pulse-width modulation (PWM).
- Programmable delay block for scheduling events relative to start of frame.
- Modulo timer for scheduling periodic events.
- Minimal external component requirements.
- 24-lead, 3 mm by 5 mm by 1 mm, RoHS-compliant package.
- –40°C to +85°C operating temperature.

Chapter 3
Pins and Connections (PINS)

3.1 Pins and Connections (PINS)

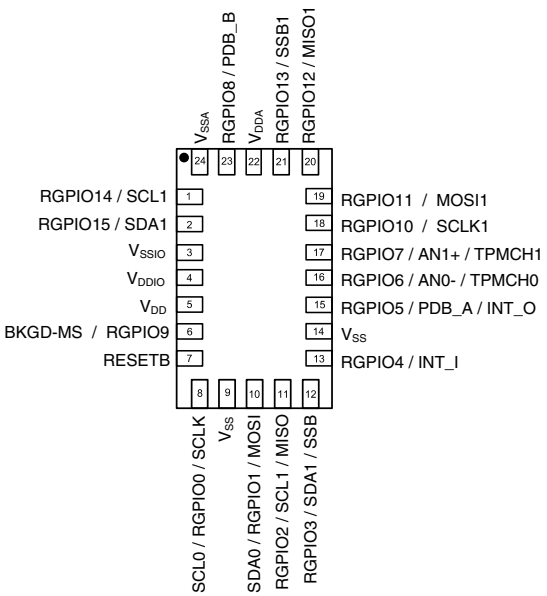


Figure 3-1. Device pinout (top view)

Table 3-1. Pin functions

Pin #	Default Pin Function[1]	Pin Function #2	Pin Function #3	Description
1	SCL1[2]	RGPIO14		Master I²C Clock / RGPIO14
2	SDA1[3]	RGPIO15		Master I²C Data / RGPIO15
3	V_{SSIO}			I/O ground
4	V_{DDIO}			I/O power supply
5	V_{DD}			Digital power supply
6	BKGD/MS	RGPIO9		Background debug - Mode select / RGPIO9
7	RESETB[4]			Active low reset with internal, pullup resistor
8	SCL0	RGPIO0	SCLK	Serial clock for slave I²C / RGPIO0 / Serial clock for slave SPI

Table continues on the next page...

Table 3-1. Pin functions (continued)

Pin #	Default Pin Function[1]	Pin Function #2	Pin Function #3	Description
9	V_{SS}			Digital ground
10	SDA0	RGPIO1	MOSI	Serial data for slave I^2C / RGPIO1 / SPI Master Output Slave Input
11	RGPIO2	SCL1	MISO	RGPIO2 / Serial clock for master I^2C / SPI Master Input Slave Output
12	RGPIO3	SDA1	SSB	RGPIO3 / Serial data for master I^2C / SPI slave select
13	RGPIO4	INT_I		RGPIO4 / Interrupt input
14	V_{SS}			Must be connected to GND externally
15	RGPIO5	PDB_A	INT_O	RGPIO5 / PDB_A / Interrupt output
16	RGPIO6	AN0-	TPMCH0	RGPIO6 / ADC Input 0 / TPM Channel 0
17	RGPIO7	AN1+	TPMCH1	RGPIO7 / ADC Input 1 / TPM Channel 1
18	SCLK1	RGPIO10		master queued SPI clock / RGPIO10
19	MOSI1	RGPIO11		master queued SPI Master Output Slave Input / RGPIO11
20	MISO1	RGPIO12		master queued SPI Master Input Slave Output / RGPIO12
21	SSB1	RGPIO13		master queued SPI slave select / RGPIO13
22	V_{DDA}			Analog power
23[5]	RGPIO8	PDB_B		RGPIO8 / PDB_B
24	V_{SSA}			Analog ground

1. Default Pin Function 1 represents the reset state of the device. Pin functions may be changed via the SIM pin mux-control registers. (See SIM memory map/register definitions). Drive strength and pullup controls are programmed by the port control registers.
2. SCL1 is available for use on pin (RGPIO14) only when SIM_PMCR0[A2] is not equal to "01". That setting would enable it for pin 11 (RGPIO2).
3. SDA1 is available for use on pin (RGPIO15) only when SIM_PMCR0[A3] is not equal to "01". That setting would enable it for pin 12 (RGPIO3).
4. RESETB defaults to input only, but can be configured as an open-drain, bidirectional pin.
5. GPIO8/PDB_B = LOW at startup indicates that SPI should be used as slave instead of the I^2C module.

3.1.1 Pin function description

Descriptions of the pin functions available on this device are provided in this section.

Sixteen of the device pins are multiplexed with Rapid GPIO (RGPIO) functions. The Default Pin Function column of Table 3-1 lists which function is active when the device exits the reset state. Freescale or customer firmware can use the Pin Mux Control registers in the System Integration Module (SIM) to change pin assignments for these pins after reset.

V_{DDIO} and V_{SSIO}

I/O power and ground. V_{DDIO} ranges from 1.71 V to 3.6 V for this device. The device will not load the I^2C bus if V_{DDIO} is not connected. Parasitic paths to supply this power domain from other pins is not recommended.

V_{DD} and V_{SS}

Digital power and ground. V_{DD} is nominally 1.8 V for this device. Parasitic paths to supply this power domain from other pins is not recommended.

V_{DDA} and V_{SSA}

Analog power and ground. V_{DDA} is nominally 1.8 V for this device. It is recommended that this supply voltage be filtered to remove any digital noise that may be present on the supply.

RESETB

The RESETB pin is an open-drain, bidirectional pin. At power up, it is configured strictly as an input pin. Setting RCSR[DR] (Reset Control & Status Register “Drive Reset” bit) to one will cause the RESET function to become bidirectional. Using this feature, FXLC95000CL can reset external devices whenever it is reset for any purpose other than power-on-reset.

Slave I^2C: SDA0, SCL0

Slave I^2C data and clock signals. FXLC95000CL may be controlled via this serial port or via the slave SPI interface. At reset, SDA0 and SCL0 are open-drain, bidirectional in input mode, with the pullup resistor disabled.

Master I^2C: SDA1, SCL1

Master I^2C data and clock signals. Because the FXLC95000CL contains a 32-bit ColdFire V1 CPU, it is fully capable of mastering other devices in the system via this serial port.

State at reset: active. SCL1 and SDA1 are configured on pins 1 and 2, respectively. The alternate functionality on these pins is RGPIO14 and RGPIO15.

Analog-to-Digital Conversion: AN0, AN1

The on-chip ADC can be used to perform a differential analog-to-digital conversion based upon the voltage present across pins AN0(-) and AN1(+). Conversions for these pins are at the same Sample Data Rate (SDR) as the MEMS transducer signals.

State at reset: Inactive. AN[1:0] are secondary functions on RGPIO[7:6], which own the pins at reset.

Rapid General Purpose I/O: RGPIO[15:0]

The ColdFire V1 CPU has a feature called “Rapid GPIO” or RGPIO. This is a 16-bit input/output port with single-cycle write, set, clear, and toggle functions available to the CPU. The FXLC95000CL brings out all 16 bits of that port as pins of the device.

State at reset:

- RGPIO[15:14]: inactive. SDA1 and SCL1 own the pin at reset.
- RGPIO[13:10, 8:2]: Pin mux registers for these bits are configured as RGPIO. Pullups are disabled. RGPIO functionality can be enabled via RGPIO_ENB[13:10, 8:2].
- RGPIO[9]: Inactive. BKGD/MS owns the pin at reset
- RGPIO[1:0]: inactive. SDA0 and SCL0 own the pin at reset.

Configuration details:

- RGPIO[15:14] are configured as Master I^2C port at reset when RGPIO_ENB[15:14]=00 and PMCR[A3]=PMCR[A2]=00 or 10. They can only be configured as RGPIO when PMCR[A3]=PMCR[A2]=01. RGPIO_ENB[15:14] must also be set to 11 for them to assume RGPIO functionality.
- RGPIO_ENB[13:10] are used to configure RGPIO[13:10].
- Pin function selections are made via the SIM pin mux registers for RGPIO[9:0].

Interrupts: INT_I

This input pin may be used to wake the CPU from a deep-sleep mode. It can be programmed to trigger on either rising or falling edge or high or low level. This pin operates as a level 7 (high priority) interrupt.

Interrupts: INT_O

RGPIO5 (pin 11) can be configured to function as an interrupt output pin. This interrupt can be asserted via software when a command response packet has been stored on the slave port mailboxes and is ready for the host to read. The host will see the interrupt and can read the data from the FXLC95000CL platform. The FXLC95000CL will automatically clear the interrupt once it recognizes that the response packet is being transmitted. This clearing action occurs while the packet is being read and prevents the host from falsely recognizing the same interrupt after the packet read is complete.

State at reset: Pin muxing is set to RGPIO5 mode.

Debug/Mode Control: BKGD/MS

At power-up, this pin operates as Mode Select. If low during power-up, the CPU will boot into debug halt mode. If high, the CPU will boot normally and run code. After power-on reset, this pin operates as a bidirectional, single-wire Background Debug port. CodeWarrior uses the Background Debug port to download code into on-chip RAM and flash, and for debugging that code using breakpoints and single stepping.

State at reset: Mode Select (MS).

MS = 1'b0, at exit from reset → boot to debug halt mode.
MS = 1'b1, at exit from reset → boot to run mode.

State after reset: BKGD. The BKGD pin is a bidirectional, pseudo-open-drain pin used for communications with a debug environment.

For details, see Coldfire V1 debug.

Programmable Delay Block: PDB_A, PDB_B

These are the two outputs of the programmable delay block (PDB). Normally, the PDB is used to schedule internal events at some fixed interval(s) relative to start of either the analog or digital phase. By bringing the PDB outputs to these pins, it becomes possible for the FXLC95000CL to initiate some external event, also relative to start of analog or digital phase. For more information, refer to the FXLC95000CL Hardware Reference Manual.

Timer: TPMCH0 and TPMCH1

These pins are the outputs for a general modulo 16 timer and general input/output capture (TPM) and pulse width modulation (PWM) functions.

Slave SPI Interface: SCLK, MOSI, MISO, SSB

Slave SPI clock, master-output slave-input, master-input slave-output, and slave-select signals. The FXLC95000CL may be controlled via this serial port or via the slave I^2C interface.

State at reset: In reset, these pins are configured according to I^2C and RGPIO[3:2] functions listed above. The pin may be reconfigured for SPI use as part of the boot process.

Master SPI Interface: SCLK1, MOSI1, MISO1, SSB1

Master SPI clock, master-output slave-input, master-input slave-output, and slave-select signals.

State at reset: In reset, these pins are configured as RGPIO[13:10] functions listed above.

3.2 System connections

The FXLC95000CL platform offers the choice of connecting to a host processor through either an I^2C or SPI interface. It can also act as a master controller for I^2C or SPI peripherals and analog sensors.

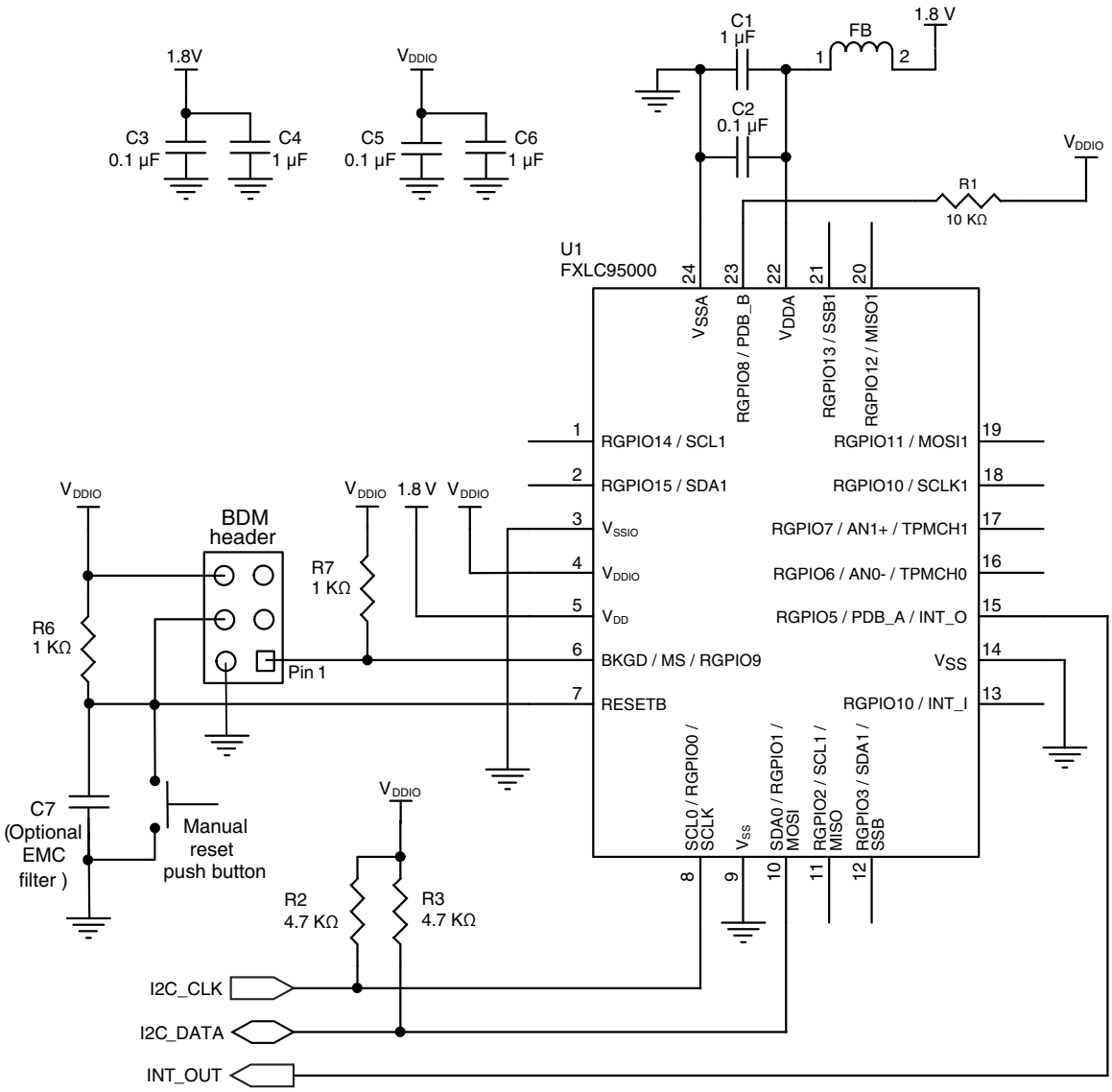
3.2.1 Power supply considerations

- An internal circuit powered by V_{DDA} provides the FXLC95000CL with a power-on-reset signal. For this signal to be properly recognized, it is important that V_{DD} is powered up before or simultaneously with V_{DDA}.
- The voltage potential difference between V_{DD} and V_{DDA} must not exceed ±0.1 V. The simplest way to accomplish this is to power both pins from the same voltage source.
- When using the same voltage source, some digital noise might reach the analog section. To prevent this, connect a small inductor or ferrite bead in serial with both the V_{DDA} and V_{SSA} traces. Additionally, two ceramic capacitors (of approximately 1 µF, and 100 nF, respectively) can be used to efficiently bypass the power and ground of both digital and analog supply rails.
- V_{DDIO} must rise up before or simultaneously with V_{DDA}/V_{DD}.

3.2.2 FXLC95000CL as an intelligent slave

I^2C pullup resistors, a ferrite bead, and a few bypass capacitors are all that are required to attach this device to a host platform. The basic configuration of the I^2C interface is shown in Figure 3-2.

The voltage level on pin 23 (RGPIO8) selects the slave-port format: I^2C or SPI. The RGPIO pins can also be programmed to generate interrupts to the host platform, in response to the occurrence of application events. In this case, the pins should be routed to the external interrupt pins of the host processor.

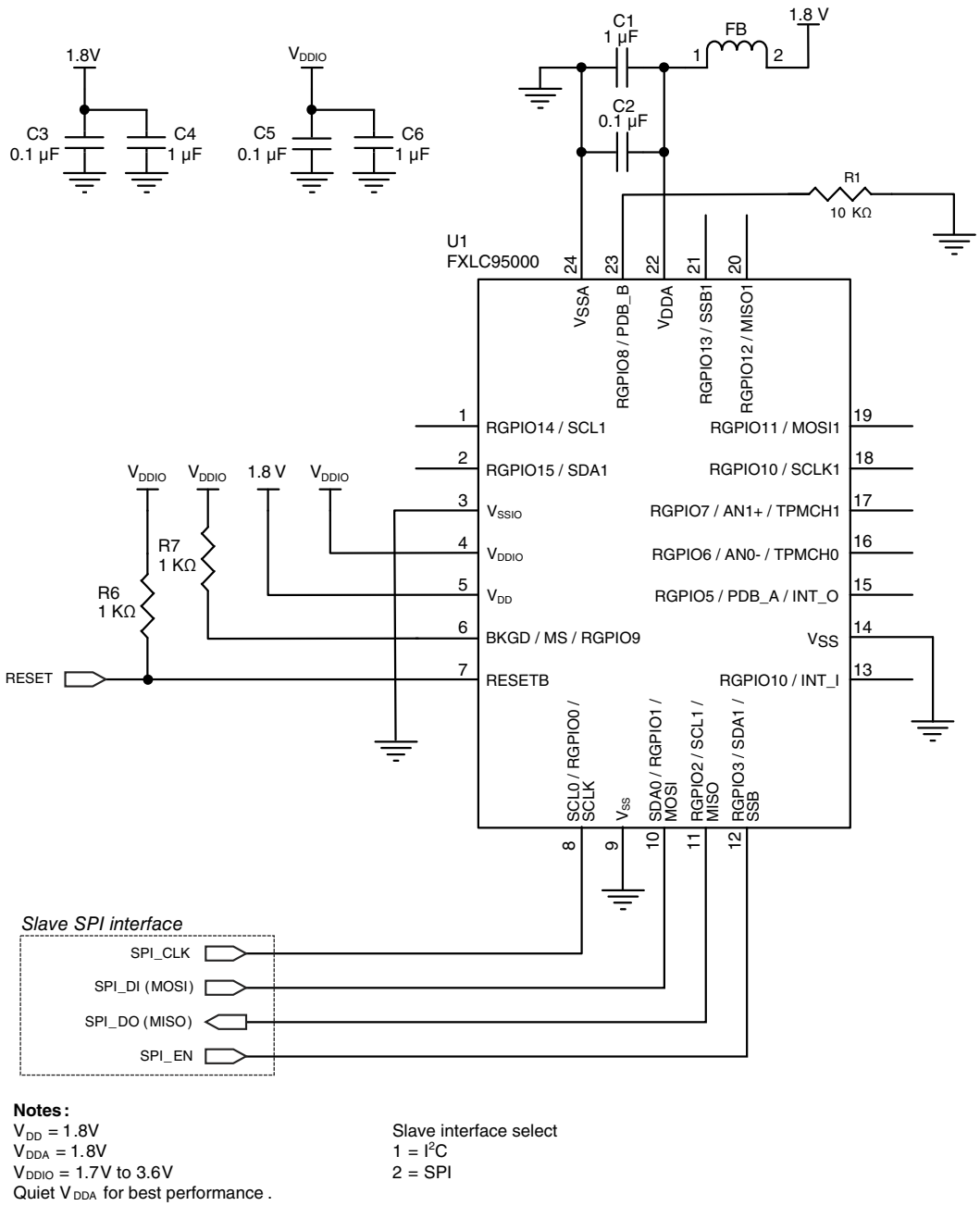


Notes:
V_{DD} = 1.8V
V_{DDA} = 1.8V
V_{DDIO} = 1.71V to 3.6V
Quiet V_{DDA} for best performance.

Pn = RGPIOn
(n from 0 to 15)

Figure 3-2. FXLC95000CL as a slave (I^2C interface)

The basic configuration of the SPI interface is shown in Figure 3-3. The RGPIO pins can also be programmed to generate interrupts to the host platform, in response to the occurrence of application events. In this case, the pins should be routed to the external interrupt pins of the host processor.



Notes:
V_{DD} = 1.8V
V_{DDA} = 1.8V
V_{DDIO} = 1.7V to 3.6V
Quiet V_{DDA} for best performance .

Slave interface select
1 = I^2C
2 = SPI

Figure 3-3. FXLC95000CL as a slave (SPI interface)

3.2.3 FXLC95000CL as a sensor hub

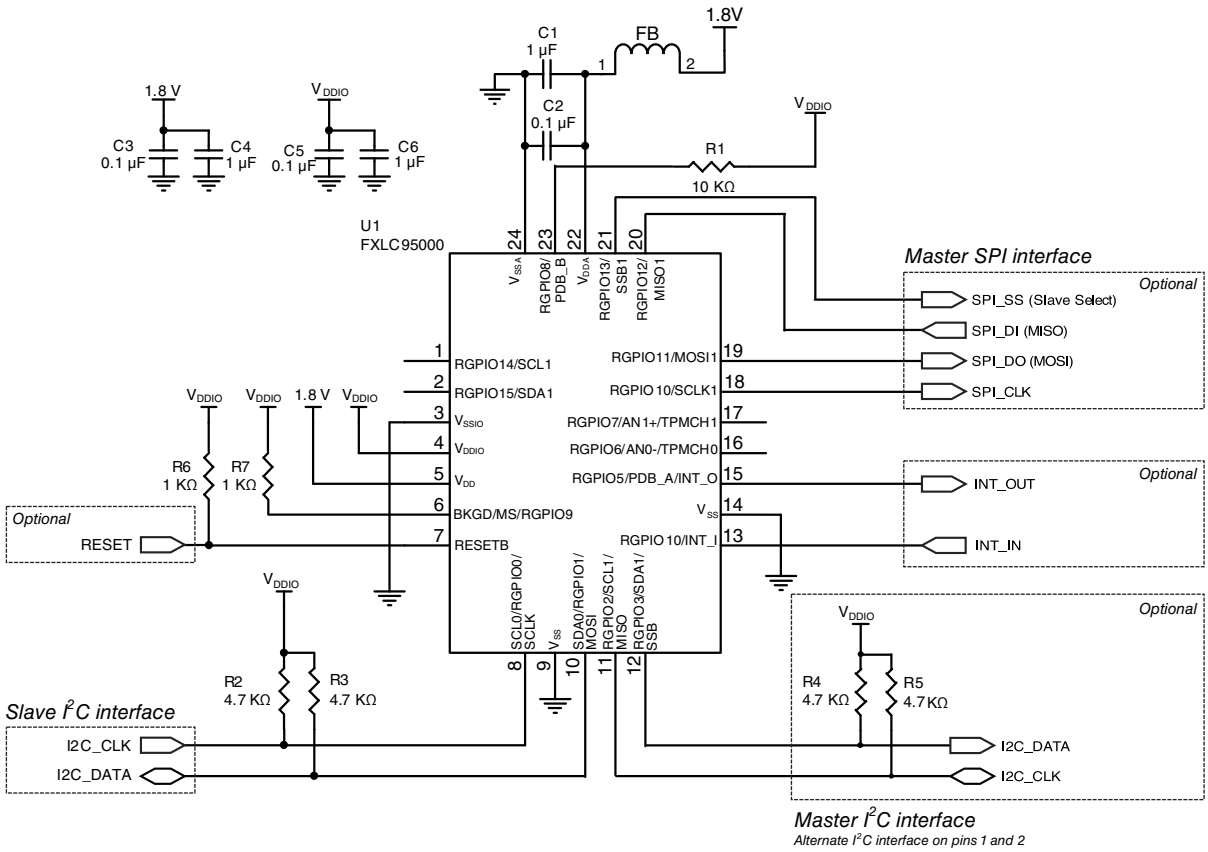
The FXLC95000CL device includes a powerful 32-bit ColdFire V1 CPU associated with an ample amount of RAM and flash memory, a master SPI and I^2C bus, and external differential analog inputs. These are the key hardware components that transform FXLC95000CL into an efficient and versatile sensor hub. The FXLC95000CL Xtrinsic Intelligent Sensing Platform can interface and manage almost any type of sensor, digital

or analog, such as pressure sensors, magnetometers, gyroscopes, and humidity sensors. The system supports external sensors interfacing to FXLC95000CL concurrently, via a combination of master SPI and master I^2C interfaces, and external differential analog inputs.

Besides FXLC95000CL rich connectivity, the 32-bit core and hardware Multiply Accumulator (MAC) provide the processing power to collect, manipulate and fuse all sensors measurement locally and make appropriate decisions to optimize overall system power consumption.

For example, FXLC95000CL can be programmed to operate effectively as a power controller for handheld units by enabling the host platform to put itself to sleep, with confidence that the FXLC95000CL will issue a wake-up request when an external event requires the host's attention. Figure 3-4 shows the FXLC95000CL being used in this sensor hub configuration. Note the simple connections. Only a few bypass capacitors, a ferrite bead, and pullup resistors for the I^2C buses are required.

- Slave I^2C interface is dedicated to communication with the host processor. Interrupt output line INT_O can be involved as well.
- Master SPI, Master I^2C, AN0/AN1 and interrupt input line INT_I are available to interface a variety of external sensors



Notes:

V_{DD} = 1.8 V
V_{DDA} = 1.8 V
V_{DDIO} = 1.7 V to 3.6 V
Quiet V_{DDA} for best performance.

Slave interface select
1 = I^2C
2 = SPI

Figure 3-4. FXLC95000CL as a sensor hub (I^2C interface)

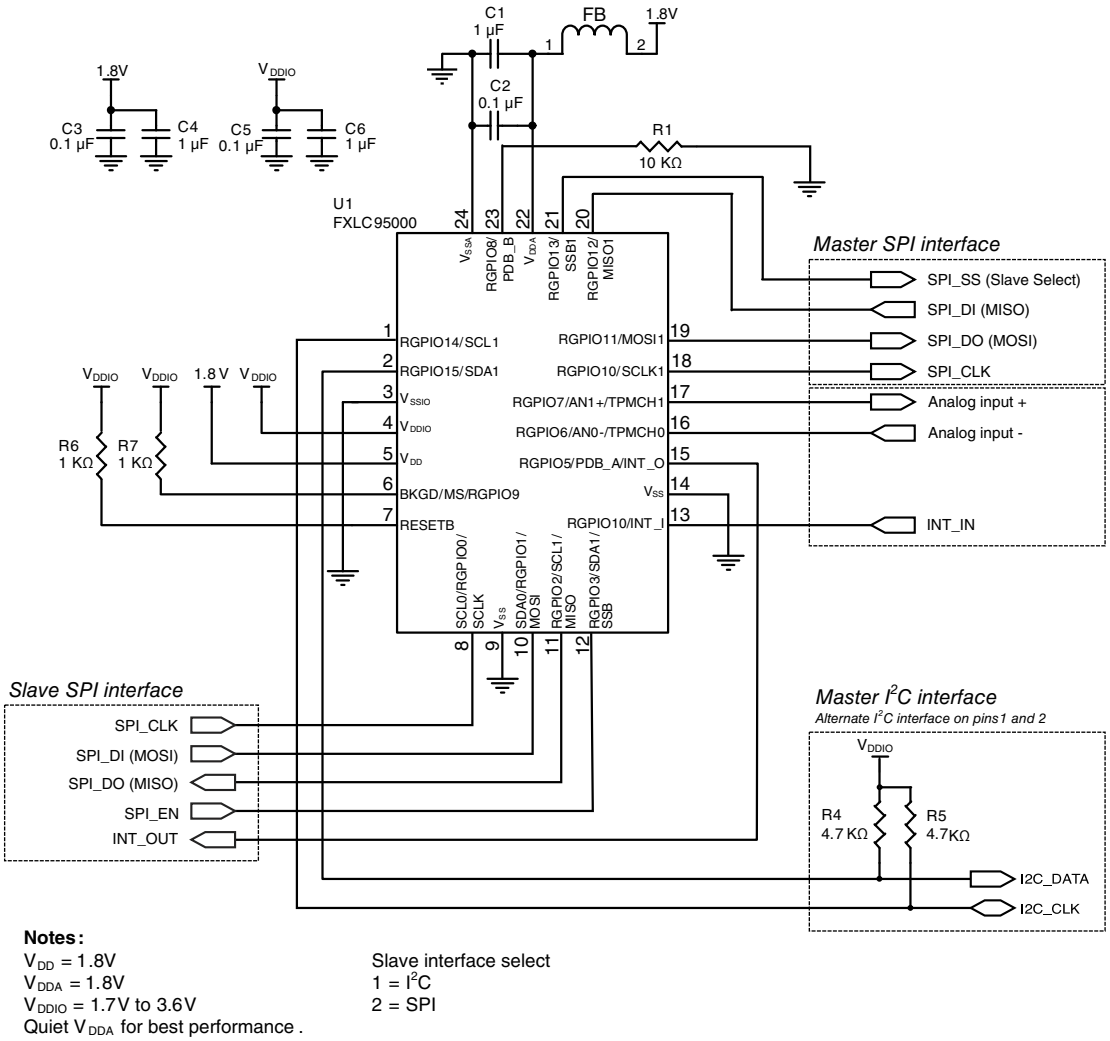


Figure 3-5. FXLC95000CL as a sensor hub (SPI interface)

3.2.4 RESETB pin

Figure 3-2 shows a complete circuit where a Reset event can be generated by:

- An external, manual reset button
- The Background Debug Mode interface
- The V_{DD} main supply

An external, pullup resistor is necessary to reduce and better control the RESETB voltage-settling time. An optional shunt capacitor to ground can be added to that node, to reduce EMC and noise susceptibility. With the shunt capacitor, the maximum RC time constant has to be strictly bounded. (For details, see Reset outputs.)

At power-up, the RESETB pin is configured as an input pin, but it also can be programmed as bidirectional. Using the bidirectional feature, the FXLC95000CL can reset external devices for any purpose other than power-on-reset. When using the RESETB pin output drive capability, the allowed upper limit for the RC time constant is reduced to only microseconds.

3.2.5 Background/Mode Select (BKGD/MS)

Figure 3-2 shows the connection to the BKGD/MS pin when an in-circuit debug capability is desired.

In this configuration, the background header also takes control of the RESETB line. This could result in parasitic capacitance from the BDM connector and its ribbon cable, which may increase RESETB settling time. This situation must be considered in the target implementation.

Chapter 4
Memory Maps (MEMORY)

4.1 High-level memory map

The memory map for the FXLC95000CL is based on the generic ColdFire V1 memory configuration. The FXLC95000CL memory covers the entire V1 address space - some segments are allocated to ROM, RAM, Flash, RGPIO and slave peripherals and others are unimplemented.

Table 4-1. FXLC95000CL memory map

Address range	Generic V1 ColdFire memory usage	Address range	FXLC95000CL memory usage
0x(00)00_0000	Allocated to on-chip flash memory	0x(00)00_0000	128K Bytes flash memory
		...	
		0x(00)01_FFFF	
		0x(00)02_0000	Unimplemented
...		...	
0x(00)2F_FFFF		0x(00)2F_FFFF	
0x(00)30_0000	Allocated to on-chip ROM	0x(00)30_0000	16K Bytes ROM
...		...	
0x(00)3F_FFFF		0x(00)30_3FFF	
0x(00)40_0000	Optional off-chip expansion	0x(00)30_4000	Unimplemented
...		...	
0x(00)7F_FFFF		0x(00)7F_FFFF	
0x(00)80_0000	Allocated to on-chip RAM	0x(00)80_0000	16K Bytes RAM
...		...	
0x(00)9F_FFFF		0x(00)80_3FFF	
0x(00)A0_0000	Optional off-chip expansion	0x(00)80_4000	Unimplemented
...		...	
0x(00)BF_FFFF		0x(00)BF_FFFF	
0x(00)C0_0000	ColdFire Rapid GPIO	0x(00)C0_0000	ColdFire Rapid GPIO
...		...	
0x(00)C0_000F		0x(00)C0_000F	

Table continues on the next page...

Table 4-1. FXLC95000CL memory map (continued)

Address range	Generic V1 ColdFire memory usage	Address range	FXLC95000CL memory usage
0x(00)C0_0010	Unimplemented	0x(00)C0_0010	Unimplemented
...		...	
0x(FF)FF_7FFF		0x(FF)FF_7FFF	
0x(FF)FF_8000	Slave peripherals	0x(FF)FF_8000	Slave peripherals
...		...	
0x(FF)FF_FFFF		0x(FF)FF_FFFF	

The left-most map in Table 4-1 is the generic, high-level, memory map applicable to the V1 ColdFire family. Memory map areas shown for RAM, ROM and flash are maximum supported for the family. Lesser amounts of all three will usually be included on specific orderable devices. The memory map for the FXLC95000CL is shown on the right.

The slave peripherals section of the memory map is further broken down as shown in Table 4-2. FXLC95000CL microcontrollers include off-platform, 8-bit and 16-bit peripheral buses. The bus bridges from the ColdFire system bus to off-platform buses are capable of serializing 32-bit accesses into two 16-bit accesses or 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.

The off-platform 8- and 16-bit interfaces operate at the same speed as the CPU.

CPU accesses to those parts of the memory map marked as "Unimplemented" in Table 4-1 result in an illegal address reset if CPUCR[ARD] = 0 or an address error exception if CPUCR[ARD] = 1.

Table 4-2. High-level peripheral memory map

Peripheral	Description	Instance name	Native bus width	Base address
RGPIO	Rapid general-purpose I/O	RGPIO	16	0x(00)C0_0000
Slave I^2C	Slave I^2C	SI2C	8	0x(FF)FF_8000
IIC	Inter-integrated IC	MI2C	8	0x(FF)FF_8040
SIM	System integration module	SIM	8	0x(FF)FF_8060
CLKGEN	CLKGEN	CK	8	0x(FF)FF_8080
MTIM16	16-bit modulo timer	MTIM	8	0x(FF)FF_80A0
IRQ	External interrupt module	IRQ	8	0x(FF)FF_80C0
Port control module	Port I/O control module 0	PC0	8	0x(FF)FF_80E0
Port control module	Port I/O control module 1	PC1	8	0x(FF)FF_8100
TPM	Two-channel, timer/pulse-width modulator	TPM	8	0x(FF)FF_8120
PDB	Programmable delay block	PDB	16	0x(FF)FF_EC00
Flash controller	Flash controller	FC	16	0x(FF)FF_EC20

Table continues on the next page...

Table 4-2. High-level peripheral memory map (continued)

Peripheral	Description	Instance name	Native bus width	Base address
AFE	Analog front end	AFE	16	0x(FF)FF_EC40
QSPI	Queued serial peripheral interface	QSPI	16	0x(FF)FF_EC80
INTC	V1 ColdFire interrupt controller	INTC	8	0x(FF)FF_FFC0

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

4.2 Read-after-write sequence and required serialization of memory operations

In some situations, a write to a peripheral must be completed fully before a subsequent action can occur. Examples of such situations include:

- Exiting an interrupt service routine (ISR)
- Changing a mode
- Configuring a function

In these situations, application software must perform a read-after-write sequence to guarantee the required serialization of the memory operations:

1. Write the peripheral register.
2. Read the written peripheral register to verify the write.
3. Continue with subsequent operations.

NOTE

One factor contributing to these situations is processor write buffering. The processor architecture has a programmable configuration bit to disable write buffering: CPUCR[BWD]. However, disabling buffered writes is likely to degrade system performance much more than simply performing the required memory serialization for the situations that truly require it.

4.3 Memory alignment issues

Regions within the memory map are subject to restrictions with regard to the types of CPU accesses allowed.

ColdFire has a big endian byte-addressable memory architecture, so the most-significant byte of each address is the lowest-numbered one, as shown in Table 4-3. Multi-byte operands (such as 16-bit words and 32-bit long-words) are referenced using an address pointing to the most-significant (first) byte.

Table 4-3. ColdFire memory organization

31	24	23	16	15	8	7	0
Longword 0x(00)00_0000							
Word 0x(00)00_0000				Word 0x(00)00_0002			
Byte 0x(00)00_0000		Byte 0x(00)00_0001		Byte 0x(00)00_0002		Byte 0x(00)00_0003	
Longword 0x(00)00_0004							
Word 0x(00)00_0004				Word 0x(00)00_0006			
Byte 0x(00)00_0004 ...		Byte 0x(00)00_0005 ...		Byte 0x(00)00_0006 ...		Byte 0x(00)00_0007 ...	
...		...		...		...	
Longword 0x(FF)FF_FFFC							
Word 0x(FF)FF_FFFC				Word 0x(FF)FF_FFFE			
Byte 0x(FF)FF_FFFC		Byte 0x(FF)FF_FFFD		Byte 0x(FF)FF_FFFE		Byte 0x(FF)FF_FFFF	

Memory access restrictions are outlined in Table 4-4. 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 4-4. V1 ColdFire memory map access constraints for regions

Base address	Region	Read			Write		
		Byte	Word	Long	Byte	Word	Long
0x(00)00_0000	Flash	x	x	x	-	-	x
0x(00)30_0000	ROM	x	x	x	-	-	-
0x(00)80_0000	RAM	x	x	x	x	x	x
0x(00)C0_0000	Rapid GPIO	x	x	x	x	x	x
0x(FF)FF_8000	8-bit peripherals	x	x	x	x	x	x
0x(FF)FF_EC00	16-bit peripherals	-	x	x	-	x	x

4.4 Memory maps for on-chip peripherals

The following tables summarize register bitfields for on-chip peripherals. For further details, see the chapter of the specific peripheral.

4.4.1 Rapid General Purpose Input/Output (RGPIO) register summary

The section of memory at 0x(00)C0_0000 is assigned for use by the ColdFire v1 Rapid GPIO module. See RGPIO for the module details.

Table 4-5. Rapid General Purpose Input/Output (RGPIO) detailed memory map

Address	Register	Bit 15/7	Bit 14/6	Bit 13/5	Bit 12/4	Bit 11/3	Bit 10/2	Bit 9/1	Bit 8/0
(00)C0_0000	RGPIO_DIR	DIR[15:8] (Read/Write)							
		DIR[7:0] (Read/Write)							
(00)C0_0002	RGPIO_DATA	DATA[15:8] (Read/Write)							
		DATA[7:0] (Read/Write)							
(00)C0_0004	RGPIO_ENB	ENB[15:8] (Read/Write)							
		ENB[7:0] (Read/Write)							
(00)C0_0006	RGPIO_CLR	CLR[15:8] (Write only)							
		CLR[7:0] (Write only)							
(00)C0_0006	RGPIO_DATA	DATA[15:8] (Read only)							
		DATA[7:0] (Read only)							
(00)C0_0008	RGPIO_DIR	DIR[15:8] (Read only)							
		DIR[7:0] (Read only)							
(00)C0_000A	RGPIO_SET	SET[15:8] (Write only)							
		SET[7:0] (Write only)							
(00)C0_000A	RGPIO_DATA	DATA[15:8] (Read only)							
		DATA[7:0] (Read only)							
(00)C0_000C	RGPIO_DIR	DIR[15:8] (Read only)							
		DIR[7:0] (Read only)							
(00)C0_000E	RGPIO_TOG	TOG[15:8] (Write only)							
		TOG[7:0] (Write only)							
(00)C0_000E	RGPIO_DATA	DATA[15:8] (Read only)							
		DATA[7:0] (Read only)							

4.4.2 Slave Interface (SLAVE) register summary

The following table summarizes the register bitfields for the SLAVE registers.

Table 4-6. Slave interface (SLAVE) detailed memory map

Address	Register	Bit 31/23 15/7	Bit 30/22 14/6	Bit 29/21 13/5	Bit 28/20 12/4	Bit 27/19 11/3	Bit 26/18 10/2	Bit 25/17 9/1	Bit 24/16 8/0
(FF)FF_8000	SP_MB0	DATA							
(FF)FF_8001	SP_MB1	DATA							
(FF)FF_8002	SP_MB2	DATA							
(FF)FF_8003	SP_MB3	DATA							
(FF)FF_8004	SP_MB4	DATA							
(FF)FF_8005	SP_MB5	DATA							
(FF)FF_8006	SP_MB6	DATA							
(FF)FF_8007	SP_MB7	DATA							
(FF)FF_8008	SP_MB8	DATA							
(FF)FF_8009	SP_MB9	DATA							
(FF)FF_800A	SP_MB10	DATA							
(FF)FF_800B	SP_MB11	DATA							
(FF)FF_800C	SP_MB12	DATA							
(FF)FF_800D	SP_MB13	DATA							
(FF)FF_800E	SP_MB14	DATA							
(FF)FF_800F	SP_MB15	DATA							
(FF)FF_8010	SP_MB16	DATA							
(FF)FF_8011	SP_MB17	DATA							
(FF)FF_8012	SP_MB18	DATA							
(FF)FF_8013	SP_MB19	DATA							
(FF)FF_8014	SP_MB20	DATA							
(FF)FF_8015	SP_MB21	DATA							
(FF)FF_8016	SP_MB22	DATA							
(FF)FF_8017	SP_MB23	DATA							
(FF)FF_8018	SP_MB24	DATA							
(FF)FF_8019	SP_MB25	DATA							
(FF)FF_801A	SP_MB26	DATA							
(FF)FF_801B	SP_MB27	DATA							
(FF)FF_801C	SP_MB28	DATA							
(FF)FF_801D	SP_MB29	DATA							
(FF)FF_801E	SP_MB30	DATA							
(FF)FF_801F	SP_MB31	DATA							
(FF)FF_8020	SP_MUTEX0	0	0	0	0	0	0	SSTS	
(FF)FF_8021	SP_MUTEX1	0	0	0	0	0	0	SSTS	
(FF)FF_8022	SP_ADDR	0	ADDR						
(FF)FF_8023	SP_SCR	EN	PS	ACTIVE	CW	RIE	WIE	WWUP	
(FF)FF_8024	SP_WSTS0	D31	D30	D29	D28	D27	D26	D25	D24

Table continues on the next page...

Table 4-6. Slave interface (SLAVE) detailed memory map (continued)

Address	Register	Bit 31/23 15/7	Bit 30/22 14/6	Bit 29/21 13/5	Bit 28/20 12/4	Bit 27/19 11/3	Bit 26/18 10/2	Bit 25/17 9/1	Bit 24/16 8/0
(FF)FF_8025	SP_WSTS1	D23	D22	D21	D20	D19	D18	D17	D16
(FF)FF_8026	SP_WSTS2	D15	D14	D13	D12	D11	D10	D9	D8
(FF)FF_8027	SP_WSTS3	D7	D6	D5	D4	D3	D2	D1	D0
(FF)FF_8028	SP_RSTS0	D31	D30	D29	D28	D27	D26	D25	D24
(FF)FF_8029	SP_RSTS1	D23	D22	D21	D20	D19	D18	D17	D16
(FF)FF_802A	SP_RSTS2	D15	D14	D13	D12	D11	D10	D9	D8
(FF)FF_802B	SP_RSTS3	D7	D6	D5	D4	D3	D2	D1	D0
(FF)FF_802C	SP_MTOR0	0	TOSTS	EN	MTE				
(FF)FF_802D	SP_MTOR1	0	TOSTS	EN	MTE				
(FF)FF_802E	SP_OIC	0	0	0	0	0	POL	CLR	SET_IN_O

4.4.3 Master I²C (MI2C) register summary

The following table summarizes the register bitfields for the on-chip peripheral I²C.

Table 4-7. Master I²C (MI2C) detailed memory map

Address	Register	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
(FF)FF_8040	IIC_A1	AD7	AD6	AD5	AD4	AD3	AD2	AD1	0
(FF)FF_8041	IIC_F	MULT		ICR					
(FF)FF_8042	IIC_C1	IICEN	IICIE	MST	TX	TXAK	RSTA	WUEN	0
(FF)FF_8043	IIC_S	TCF	IAAS	BUSY	ARBL	0	SRW	IICIF	RXAK
(FF)FF_8044	IIC_D	DATA							
(FF)FF_8045	IIC_C2	GCAEN	ADEXT	0	0	0	AD10	AD9	AD8
(FF)FF_8046	IIC_FLT	0	0	0	FLT4	FLT3	FLT2	FLT1	FLT0

4.4.4 System Integration Module (SIM) register summary

The following table summarizes the register bitfields for the on-chip peripheral SIM.

Table 4-8. System Integration Module (SIM) memory map

Address	Register	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
(FF)FF_8060	STOPCR	0	0	0	SIM_CLK_EN	FC	SC	NC	SCtoFC

Table continues on the next page...

Table 4-8. System Integration Module (SIM) memory map (continued)

Address	Register	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
(FF)FF_8061	FCSR	0	0	A_EN	SFEIE		FE	SFDIE	SF
(FF)FF_8062	RCSR	0	DR	ASR	SW	ILOP	ILAD	PIN	POR
(FF)FF_8063	SIM_TR	TP1				TP0			
(FF)FF_8064	PCESFC0	0	T2	T1	T0	IRQ	AFE	PCTRL	FLSH
(FF)FF_8065	PCESFC1	0	0	0	0	0	0	MI2C	SLAVE
(FF)FF_8066	PCESSC0	0	T2	T1	T0	IRQ	AFE	PCTRL	FLSH
(FF)FF_8067	PCESSC1	0	0	0	0	0	0	MI2C	SLAVE
(FF)FF_8068	PCERUN0	0	T2	T1	T0	IRQ	AFE	PCTRL	FLSH
(FF)FF_8069	PCERUN1	0	0	0	0	0	0	MI2C	SLAVE
(FF)FF_806A	PMCR0	A9	A8	A7		A6		0	A4
(FF)FF_806B	PMCR1	A3		A2		A1		A0	
(FF)FF_806C	PMCR2	0	0	0	0	0	0	A5	

4.4.5 On-Chip Oscillator (CLKGEN) register summary

The CLKGEN module is organized as a memory-mapped peripheral on the 8-bit IP Bus. For register details, see CLKGEN.

Table 4-9. On-chip oscillator (CLKGEN) memory map

Address	Register	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
(FF)FF_8080	CK_OSCTRL	0	FFCEN	FFSEN	FLE				
(FF)FF_8081	Reserved	Reserved							
(FF)FF_8082	Reserved	Reserved							
(FF)FF_8083	Reserved	Reserved							
(FF)FF_8084	CK_TRIMLM	TRIML[15:8]							
(FF)FF_8085	CK_TRIMLL	TRIML[7:0]							
(FF)FF_8086	CK_TRIMHM	TRIMH[15:8]							
(FF)FF_8087	CK_TRIMHL	TRIMH[7:0]							

4.4.6 Modulo Timer 16-bit (MTIM16) register summary

The following table summarizes the register-bit fields for the on-chip peripheral MTIM16.

Table 4-10. Modulo Timer 16-bit (MTIM16) memory map

Address	Register	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
(FF)FF_80A0	MTIM_SC	TOF	TOIE	TRST	TSTP	0	0	0	0
(FF)FF_80A1	MTIM_CLK	0	0	CLKS		PS			
(FF)FF_80A2	MTIM_CNTH	CNTH							
(FF)FF_80A3	MTIM_CNTL	CNTL							
(FF)FF_80A4	MTIM_MODH	MODH							
(FF)FF_80A5	MTIM_MODL	MODL							

4.4.7 Pin Interrupt (IRQ) register summary

The following table summarizes the register bitfields for the on-chip peripheral IRQ.

Table 4-11. Pin Interrupt (IRQ) memory map

Address	Register	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
(FF)FF_80C0	IRQSC	0	IRQ_PDD	IRQ_EDG	IRQ_PE	IRQ_F	IRQ_ACK	IRQ_IE	IRQ_MOD

4.4.8 Port Control 0 (PC0) register summary

The following table summarizes the register bitfields for the on-chip peripheral Port Controls.

Table 4-12. Port Control 0 (PC0) memory map

Address	Register	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
(FF)FF_80E0	PC0_PE	PE7	PE6	PE5	PE4	PE3	PE2	PE1	PE0
(FF)FF_80E1	PC0_SE	SE7	SE6	SE5	SE4	SE3	SE2	SE1	SE0
(FF)FF_80E2	PC0_DS	DS7	DS6	DS5	DS4	DS3	DS2	DS1	DS0
(FF)FF_80E3	PC0_IFE	IFE7	IFE6	IFE5	IFE4	IFE3	IFE2	IFE1	IFE0

4.4.9 Port Control 1 (PC1) register summary

The following table summarizes the register bitfields for the on-chip peripheral Port Controls.

Table 4-13. Port Control 1 (PC1) memory map

Address	Register	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
(FF)FF_8100	PC1_PE	PE7	PE6	PE5	PE4	PE3	PE2	PE1	PE0
(FF)FF_8101	PC1_SE	SE7	SE6	SE5	SE4	SE3	SE2	SE1	SE0
(FF)FF_8102	PC1_DS	DS7	DS6	DS5	DS4	DS3	DS2	DS1	DS0
(FF)FF_8103	PC1_IFE	IFE7	IFE6	IFE5	IFE4	IFE3	IFE2	IFE1	IFE0

4.4.10 Timer Pulse-Width Modulator (TPM) register summary

The following table summarizes the register bitfields for the on-chip peripheral TPM.

Table 4-14. Timer Pulse-Width Modulator (TPM) memory map

Address	Register	Bit 15/7	Bit 14/6	Bit 13/5	Bit 12/4	Bit 11/3	Bit 10/2	Bit 9/1	Bit 8/0
(FF)FF_8120	TPM_SC	TOF	TOIE	CPWMS	CLKSB | CLKSA		PS		
(FF)FF_8121	TPM_CNTH	Bit 15	14	13	12	11	10	9	Bit 8
(FF)FF_8122	TPM_CNTL	Bit 7	6	5	4	3	2	1	Bit 0
(FF)FF_8123	TPM_MODH	Bit 15	14	13	12	11	10	9	Bit 8
(FF)FF_8124	TPM_MODL	Bit 7	6	5	4	3	2	1	Bit 0
(FF)FF_8125	TPM_C0SC	CH0F	CH0IE	MS0B	MS0A	ELS0B	ELS0A	0	0
(FF)FF_8126	TPM_C0VH	Bit 15	14	13	12	11	10	9	Bit 8
(FF)FF_8127	TPM_C0VL	Bit 7	6	5	4	3	2	1	Bit 0
(FF)FF_8128	TPM_C1SC	CH1F	CH1IE	MS1B	MS1A	ELS1B	ELS1A	0	0
(FF)FF_8129	TPM_C1VH	Bit 15	14	13	12	11	10	9	Bit 8
(FF)FF_812A	TPM_C1VL	Bit 7	6	5	4	3	2	1	Bit 0

4.4.11 Programmable Delay Block (PDB) register summary

The following table summarizes the register bitfields for the on-chip peripheral PDB.

Table 4-15. Programmable Delay Block (PDB) memory map

Address	Register	Bit 15/7	Bit 14/6	Bit 13/5	Bit 12/4	Bit 11/3	Bit 10/2	Bit 9/1	Bit 8/0
(FF)FF_EC00	PDB_CSR	PRESCALER			SB	SA	IENB	IENA	BOS[1]
		BOS[0]	AOS		CONT	SWTRIG	TRIGSEL		EN
(FF)FF_EC02	PDB_DELAYA	DELAYA[15:8]							
		DELAYA[7:0]							
(FF)FF_EC04	PDB_DELAYB	DELAYB[15:8]							
		DELAYB[7:0]							

Table continues on the next page...

Table 4-15. Programmable Delay Block (PDB) memory map (continued)

Address	Register	Bit 15/7	Bit 14/6	Bit 13/5	Bit 12/4	Bit 11/3	Bit 10/2	Bit 9/1	Bit 8/0
(FF)FF_EC06	PDB_MOD	MOD[15:8]							
		MOD[7:0]							
(FF)FF_EC08	PDB_COUNT	COUNT[15:8]							
		COUNT[7:0]							

4.4.12 Flash Memory Controller (FLASH) register summary

The Flash controller registers may only be accessed when the CPU is in Supervisor mode. See FLASH.

Table 4-16. Flash memory controller (FLASH) memory map

Address	Register	Bit 15/7	Bit 14/6	Bit 13/5	Bit 12/4	Bit 11/3	Bit 10/2	Bit 9/1	Bit 8/0
(FF)FF_EC20	FLASH_FOPT	CSR1	0	BF	0	0	MECFB	CHECKB	
		1	1	PW	PROTB	1	OVRF	SSW	SSC

There is a nonvolatile register area, which consists of a reserved block of 4 bytes in flash memory at 0x(00)01_FFFC - 0x(00)01_FFFF. The byte at 0x(00)01_FFFF (NVOPT) is allocated to flash protection and security functions. Additionally, the byte at 0x(00)01_FFFE (NVBOPT) is used to initialize boot options for the device. See Security for the configuration options for these two bytes.

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

4.4.13 Analog Front End (AFE) register summary

The Analog Front End (AFE) registers may only be accessed when the CPU is in Supervisor mode. See Analog Front End (AFE).

Table 4-17. Analog Front End (AFE) memory map

Address	Peripheral	Register	Bit 15/7	14/6	13/5	12/4	11/3	10/2	9/1	8/0
(FF)FF_EC40	AFE	AFE_CSR	FS		C4S		CM		Reserved	RES1
			ST	0	0	0	CCIEN	COCO	SWTRIG	TT

4.4.14 Queued Serial Peripheral Interface (QSPI) register summary

The following table summarizes the register-bit fields for the on-chip Queued Serial Peripheral Interface (QSPI).

Table 4-18. Queued Serial Peripheral Interface (QSPI) memory map

Address	Register	Bit 15/7	Bit 14/6	Bit 13/5	Bit 12/4	Bit 11/3	Bit 10/2	Bit 9/1	Bit 8/0
(FF)FF_E0B0	QSPI_SPSCR	SPR[2:0]			DSO	ERRIE	MODFEN	SPRIE	SPMSTR
		CPOL	CPHA	SPE	SPTIE	SPRF	OVRF	MODF	SPTE
(FF)FF_E0B1	QSPI_SPDSR	WOM	0		BD2X	SSB_IN	SSB_DATA	SSB_ODM	SSB_AUTO
		SSB_DDR	SSB_STRB	SSB_OVER	SPR3	DS[3:0]			
(FF)FF_E0B2	QSPI_SPDRR	R15	R14	R13	R12	R11	R10	R9	R8
		R7	R6	R5	R4	R3	R2	R1	R0
(FF)FF_E0B3	QSPI_SPDTR	T15	T14	T13	T12	T11	T10	T9	T8
		T7	T6	T5	T4	T3	T2	T1	T0
(FF)FF_E0B4	QSPI_SPFIFO	0	TFCNT			0	RFCNT		
		0	TFWM		0	RFWM		0	FIFO_ENA
(FF)FF_E0B5	QSPI_SPWAIT	0			WAIT				
		WAIT							

4.4.15 ColdFire V1 Interrupt Controller (CF1_INTC) register summary

The 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; all memory locations are accessed as 8-bit (byte) operands. This 64-byte space includes the program-visible interrupt controller registers, as well as the space used for interrupt-acknowledge (IACK) cycles.

The following table summarizes the CF1_INTC user-accessible peripheral registers and control bits.

Table 4-19. ColdFire v1 Interrupt Controller (CF1_INTC) memory map

Address	Register	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
(FF)FF_FFD0	INTC_FRC	0	LVL1	LVL2	LVL3	LVL4	LVL5	LVL6	LVL7
(FF)FF_FFD8	INTC_PL6P7	0	0	REQN					
(FF)FF_FFD9	INTC_PL6P6	0	0	REQN					
(FF)FF_FFDB	INTC_WCR	ENB	0	0	0	0	MASK		
(FF)FF_FFDE	INTC_SFRC	0	0	SET					
(FF)FF_FFDF	INTC_CFRC	0	0	CLR					
(FF)FF_FFE0	INTC_SWIACK	0	VECN						
(FF)FF_FFE4	INTC_LVL1IACK	0	VECN						
(FF)FF_FFE8	INTC_LVL2IACK	0	VECN						
(FF)FF_FFEC	INTC_LVL3IACK	0	VECN						
(FF)FF_FFF0	INTC_LVL4IACK	0	VECN						
(FF)FF_FFF4	INTC_LVL15ACK	0	VECN						
(FF)FF_FFF8	INTC_LVL6IACK	0	VECN						
(FF)FF_FFFC	INTC_LVL7IACK	0	VECN						

4.5 Interrupt vector table

The FXLC95000CL has a default interrupt vector table that can be used to respond to many interrupts and exception conditions during operation. Interrupt enables and interrupt sources for many of the entries in the vector table are available via bitfields in various device registers.

Table 4-20 summarizes the default vector map for this device.

Table 4-20. Interrupt vector table

Vector name	Vector number	Vector offset	INT level	Priority within level	Stacked program counter	Assignment	Interrupt enable	Interrupt source
	0	0		N/A	-	Initial supervisor stack pointer		
	1	0x004		N/A	-	Initial program counter		
	2-63			N/A	-	Reserved for internal CPU exceptions (see Table 21-14)		
			7	7-5		Reserved		
irq	64	0x100	7	4	Next	IRQ	IRQ_SC[IRQIE]	IRQ_SC[IRQF]
frame_err	65	0x104	7	3	Next	SIM Frame Error	SIM_FCSR[SFEIE]	SIM_FCSR[FE]

Table continues on the next page...

Table 4-20. Interrupt vector table (continued)

Vector name	Vector number	Vector offset	INT level	Priority within level	Stacked program counter	Assignment	Interrupt enable	Interrupt source
N/A	66	0x108	7	2	Next	Expansion		
N/A	67	0x10C	7	1	Next	Expansion		
			6	7		Reserved for remapped vector #1		
			6	6		Reserved for remapped vector #2		
N/A	68	0x110	6	5	Next	Expansion		
N/A	69	0x114	6	4	Next	Expansion		
tpm1ovf	70	0x118	6	3	Next	TPM[OVRF]	TPM_1SC[TOIE]	TPM_1SC[TOF]
tpm1ch0	71	0x11C	6	2	Next	TPM[CH0]	TPM_1C0SC[CH0IE]	TPM_1C0SC[CH0F]
tpm1ch1	72	0x120	6	1	Next	TPM[CH1]	TPM_1C1SC[CH1IE]	TPM_1C1SC[CH1F]
N/A	73	0x124	5	7	Next	Expansion		
N/A	74	0x128	5	6	Next	Expansion		
mtim_ovfl	75	0x12C	5	5	Next	MTIM Overflow	MTIM_SC[TOIE]	MTIM_SC[TOF]
pdb_a	76	0x130	5	4	Next	Programmable delay A	PDB_CSR[IENA]	PDB_CSR[SA]
pdb_b	77	0x134	5	3	Next	Programmable delay B	PDB_CSR[IENB]	PDB_CSR[SB]
N/A	78	0x138	5	2	Next	Expansion		
N/A	79	0x13C	5	1	Next	Expansion		
N/A	80	0x140	4	7	Next	Expansion		
N/A	81	0x144	4	6	Next	Expansion		
sp_wake	82	0x148	4	5	Next	Slave port wake-up	SP_SCR[WIE]	Slave port write status registers
N/A	83	0x14C	4	4	Next	Expansion		
N/A	84	0x150	4	3	Next	Expansion		
N/A	85	0x154	4	2	Next	Expansion		
N/A	86	0x158	4	1	Next	Expansion		
N/A	87	0x15C	3	7	Next	Expansion		
N/A	88	0x160	3	6	Next	Expansion		
N/A	89	0x164	3	5	Next	Expansion		
sp_to_0	90	0x168	3	4	Next	Mutex zero timeout	SP_MTOR0[EN]	SP_MTOR0[STS]
sp_to_1	91	0x16C	3	3	Next	Mutex one timeout	SP_MTOR1[EN]	SP_MTOR1[STS]
N/A	92	0x170	3	2	Next	Expansion		
N/A	93	0x174	3	1	Next	Expansion		
N/A	94	0x178	2	7	Next	Expansion		

Table continues on the next page...

Table 4-20. Interrupt vector table (continued)

Vector name	Vector number	Vector offset	INT level	Priority within level	Stacked program counter	Assignment	Interrupt enable	Interrupt source
start_of _frame	95	0x17C	2	6	Next	Start of frame (phase D)	SIM_FCSR[SFDIE]	SIM_FCSR[SF]
conversion _ complete	96	0x180	2	5	Next	AFE conversion complete interrupt	AFE_CSR[CCIEN]	AFE_CSR[COCO]
N/A	97	0x184	2	4	Next	Expansion		
N/A	98	0x188	2	3	Next	Expansion		
N/A	99	0x18C	2	2	Next	Expansion		
N/A	100	0x190	2	1	Next	Expansion		
master _i2c	101	0x194	1	7	Next	Master I2C	Complete 1-byte transfer (TCF) Interrupt Match of received calling address (IAAS) Interrupt Arbitration Lost (ARBL)	I2C_C1[IICIE]
N/A	102	0x198	1	6	Next	Expansion		
L7swi	103	0x19C	7	0	Next	Level-7 software interrupt		
L6swi	104	0x1A0	6	0	Next	Level-6 software interrupt		
L5swi	105	0x1A4	5	0	Next	Level-5 software interrupt		
L4swi	106	0x1A8	4	0	Next	Level-4 software interrupt		
L3swi	107	0x1AC	3	0	Next	Level-3 software interrupt		
L2swi	108	0x1B0	2	0	Next	Level-2 software interrupt		
L1swi	109	0x1B4	1	0	Next	Level-1 software interrupt		
N/A	110	0x1B8	1	5	Next	Expansion		
N/A	111	0x1BC	1	4	Next	Expansion		
N/A	112	0x1C0	1	3	Next	Expansion		
N/A	113	0x1C4	1	2	Next	Expansion		
N/A	114	0x1C8	1	1	Next	Expansion		
N/A	115	0x1CC	N/A	N/A	Next	Reserved		
N/A	...		...	...	Next	Reserved		
N/A	255	0x3FC	N/A	N/A	Next	Reserved		

Error exceptions arising from user-mode attempts to access supervisor-only memory and registers, will result in a soft-reset of the device being performed by the "access error" exception handler specified at Vector #2 of the exception table.

4.6 RAM addressing and data

When using RAM memory in FXLC95000CL, users should consider the size, addressing mode and data initialization and retention.

The FXLC95000CL includes 16 KB of static RAM. RAM is most efficiently accessed using the A5-relative addressing mode (address register indirect with displacement mode). Any single bit in this area can be accessed with the bit manipulation instructions (such as BCLR and BSET).

At system start up, the contents of RAM are uninitialized. RAM data is unaffected by any reset, as long as the supply voltage does not drop below the minimum value for RAM retention (VRAM, in the data sheet).

Chapter 5
Slave Interface (SLAVE)

5.1 Introduction

The FXLC95000CL MCU-based, motion-sensing platform from Freescale can communicate with a host processor using either I^2C or SPI interfaces.

The selection of the operating mode between I^2C and SPI is initialized at startup. If RGPIO8 is found to be low during the boot process, then SPI mode will be programmed; see Slave Port Status and Control Register.

Both SPI and I^2C slave modules are on a clock domain that is separate from the clock domain of the rest of the device. The SPI and I^2C slave modules can be used during all modes of operation except deep sleep ($STOP_{NC}$). In deep sleep mode, the SPI and I^2C slave modules can provide a wake-up interrupt signal to exit deep sleep mode. When there is an address match or ssb_deassertion, the slave can provide a wake-up interrupt.

The slave interface is reset when the chip is reset. I^2C and SPI communications are not possible during reset (specifically sim_chip_resetb) assertion.

The conceptual architecture of this interface is shown in Figure 5-1.

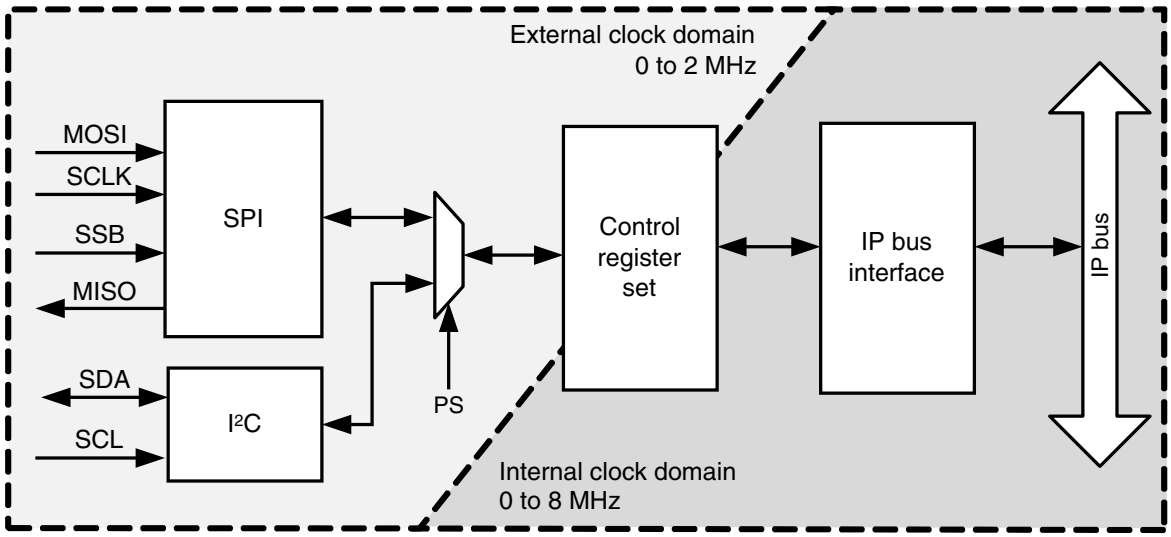


Figure 5-1. Combination SPI and I²C slave port

5.1.1 I²C features and limitations

If RGPIO8 is found to be high during the boot process, then I²C mode will be programmed.

The FXLC95000CL architecture supports Inter Integrated (I2C) communication for data transfer. The I2C interface is used to communicate with an external master device such as the host microcontroller. The signal from RGPIO8 selects the slave-port format: I2C or SPI. In addition, the RGPIO pins can be programmed to generate interrupts to host platforms. When RGPIO8 is found low during the boot process, then the SPI mode will be programmed; otherwise, I2C mode will be selected.

Figure 5-2 shows typical connection of master or slave devices on an I²C bus. The two shared, external pull-up resistors are the only external components required for proper operation of the open-drain outputs.

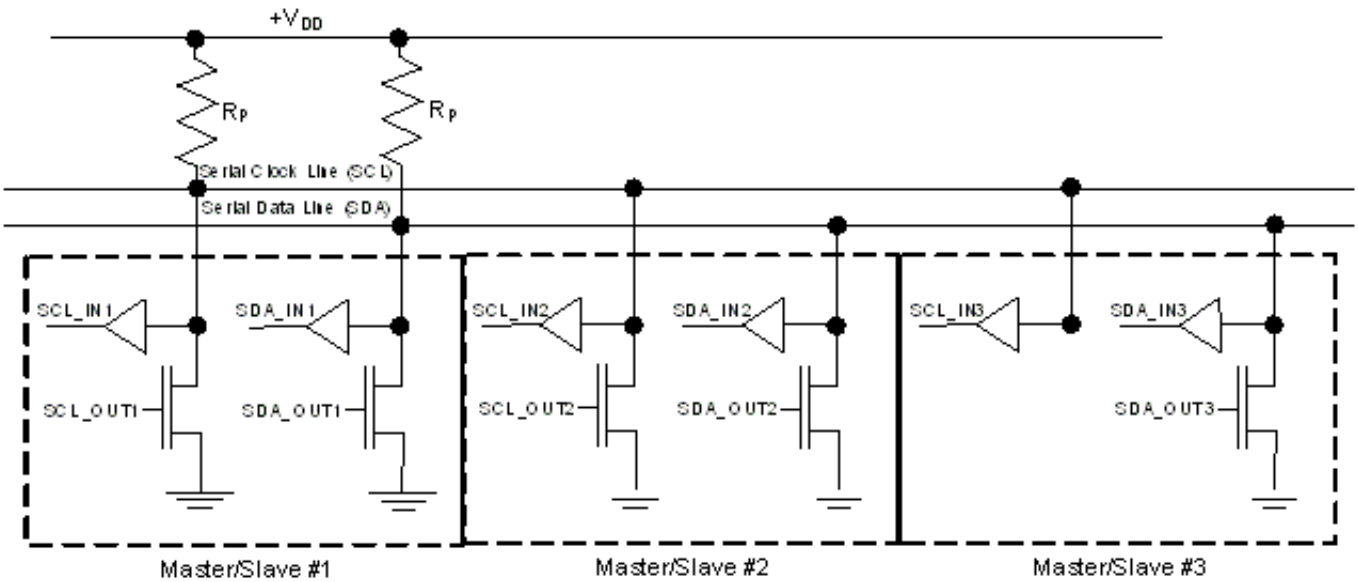


Figure 5-2. I²C wired-AND/open-drain bus

5.1.1.1 I²C features

The I²C slave port includes these distinctive features:

- Compatible with I²C bus standard
- 32 general-purpose, eight-bit mailbox registers:
  - Visible to both CPU and master of the slave I²C interface
  - Programmable to any desired function
- 32-bit read buffer supports definition of 16- and 32-bit variables in the shared mailbox space
- Two hardware semaphores are available for strict management of data-coherency issues
- Write status registers enable easy tracking of writes by the I²C communications occuring independently of CPU mode. The module is externally clocked and can operate in all modes.
- Configurable I²C device address
- 2-Mbps maximum data-transfer rate
- Configurable wake-up behavior
- Register address auto-increments between accesses. It wraps from Mailbox #31 back to Mailbox #0.

5.1.1.2 I²C limitations

This module offers a subset of the features available in the full standard. In particular, the module is subject to the following limitations:

- Seven-bit addressing only
- Maximum SCL frequency of operation equals 2 Mbps
- General call is not supported
- "Start Byte" is not supported
- Slave-only operation
- Input filters are not implemented
- Use of standard, digital-I/O impose loading limitations on the bus

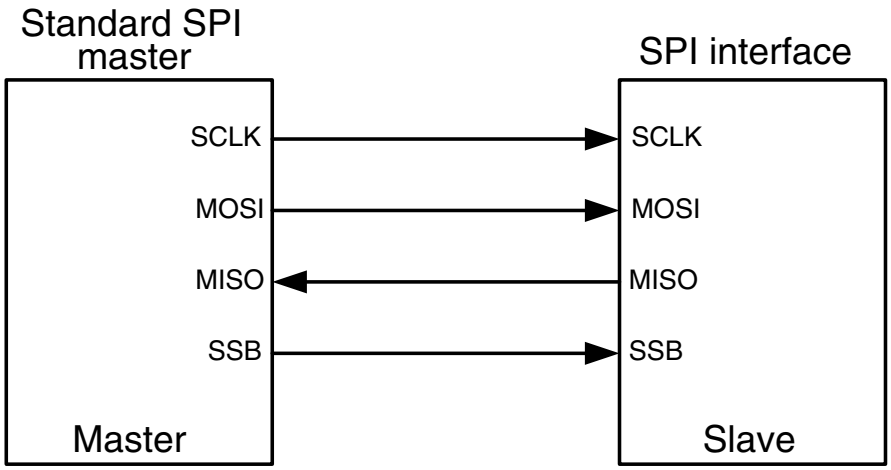
Ignoring spike protection and hold-time differences, the I²C standard states that "the only difference between Hs-Mode slave devices and F/S-mode slave devices is the speed at which they operate." Thus, the FXLC95000CL slave I²C port can be used with Hs-mode devices operating up to 2 Mbps.

5.1.2 SPI features and limitations

The FXLC95000CL architecture also supports Serial Peripheral Interface (SPI) communication for digital communications. The SPI is used for synchronous, serial communication between a master device and one or more slave devices. See Figure 5-3 for an example of how to configure one master with one FXLC95000CL device.

The FXLC95000CL is always operated as a slave device. Typically, the master device would be the host microcontroller, which would drive the clock (SCLK) and chip-select (SSB) signals. (In a SPI interface, those 2 signals (SCLK, SSB) are driven (controlled) only by the master device.)

The "slave device" behavior of FXLC 95000 is specific to the Slave port (which can be either I²C or SPI interface). On the other hand, FXLC95000 can use its additional I²C and SPI interfaces/pins to control external sensors, then acting as a master device.



Standard SPI pins

Pin name	Function
SCLK	SPI clock
SSB	Slave select(active low)
MOSI	Master out, slave in
MISO	Master in, slave out

Platform's SPI pins

Pin name	Function
SCLK	SPI clock
SSB	Slave select(active low)
MOSI	Master out,slave in
MISO	Master in,slave out

Figure 5-3. Dedicated connection to SPI host

The SPI interface consists of two control lines and two data lines: SSB, SCLK, MOSI and MISO. The SSB pin, also known as "Slave Select" (active low), is the slave, device-enable mechanism which is controlled by the SPI master. SSB is driven low at the start of a transmission and driven high at the end of a transmission.

SCLK is the SPI clock that is also controlled by the SPI master. MISO and MOSI are the master-in, slave-out and the master-out, slave-in pins.

Figure 5-4 illustrates the standard mechanism by which a master controls two or more slave devices that share the SPI bus. In this scenario, the master uses two general-purpose I/O pins to signal which of the two slaves is enabled at any point in time.

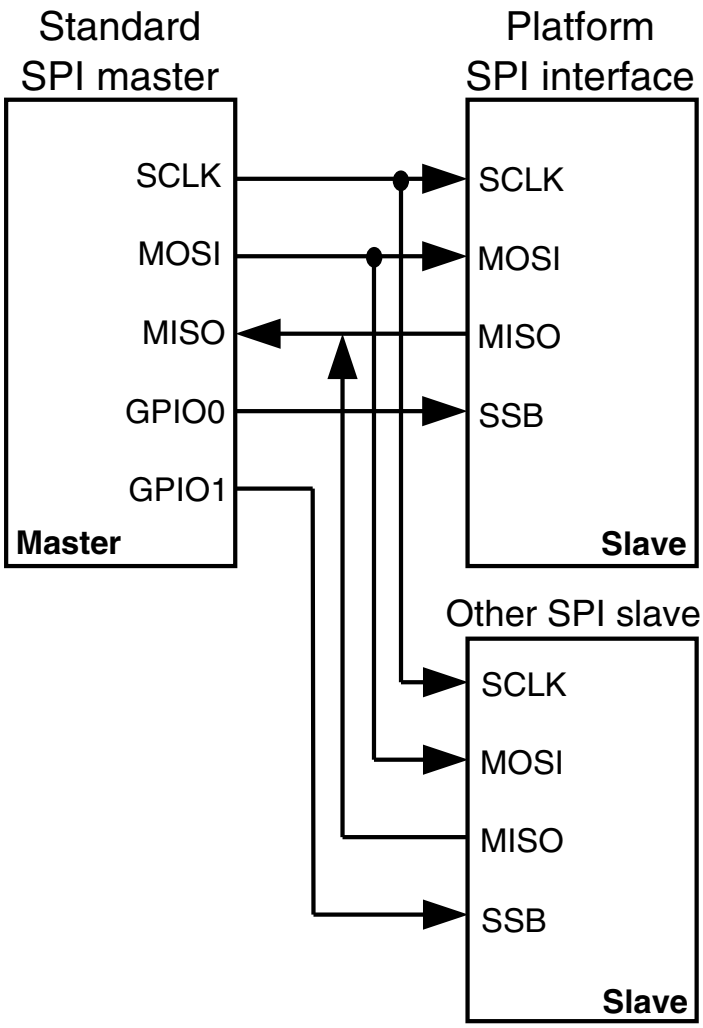


Figure 5-4. Shared connection to SPI host

5.1.2.1 SPI features

The SPI slave port includes these distinctive features:

- Compatible with SPI interfaces found on many microcontrollers
- 32 general-purpose, eight-bit mailbox registers:
  - Visible to both CPU and master of the SPI master
  - Can be programmed for any desired function
- 32-bit read buffer supports definition of 16- and 32-bit variables in the shared mailbox space
- Two hardware semaphores are available for strict management of data coherency issues

- Write status registers enable easy tracking of writes by the SPI communications occur independently of CPU mode. The module is externally clocked and can operate in all modes.
- 2-Mbps maximum data-transfer rate
- Configurable wake-up behavior
- Register address auto-increments between accesses. It wraps from Mailbox #31 back to Mailbox #0.

5.1.2.2 SPI limitations

- Fixed clock polarity: The MOSI and MISO data lines are driven at the falling edge of the SCLK and should be captured at the rising edge of the SCLK.
- Clock phase is fixed (See SPI serial protocol and timing). Data is shifted MSB first, LSB last.
- SSB should not be strobed between bytes of a multi-byte transfer.
- There is no explicit protocol checking. Invalid results may occur if an invalid sequence is received by the slave interface.

5.2 Slave port module memory map

The slave port module is organized as a memory-mapped peripheral on the eight-bit IP bus.

Table 5-1 presents one view of the module memory map.

Table 5-1. Module memory map

Register Name	Address	Visibility	Function
SP_MB0	0x(FF)FF_8000	CPU/I²C	Mailbox Register 0
SP_MB1	0x(FF)FF_8001	CPU/I²C	Mailbox Register 1
...	...	...	...
SP_MB30	0x(FF)FF_800E	CPU/I²C	Mailbox Register 30
SP_MB31	0x(FF)FF_800F	CPU/I²C	Mailbox Register 31
MUTEX0	0x(FF)FF_8020	CPU/I²C	Binary Semaphore (Mutex) Register 0
MUTEX1	0x(FF)FF_8021	CPU/I²C	Binary Semaphore (Mutex) Register 1
SP_ADDR	0x(FF)FF_8022	CPU only	Slave I²C Address Register

Table continues on the next page...

Table 5-1. Module memory map (continued)

Register Name	Address	Visibility	Function
SP_SCR	0x(FF)FF_8023	CPU only	Slave Port Status and Control Register
SP_WSTS0	0x(FF)FF_8024	CPU only	Write Status Register 0
SP_WSTS1	0x(FF)FF_8025	CPU only	Write Status Register 1
SP_WSTS2	0x(FF)FF_8026	CPU only	Write Status Register 2
SP_WSTS3	0x(FF)FF_8027	CPU only	Write Status Register 3
SP_RSTS0	0x(FF)FF_8028	CPU only	Read Status Register 0
SP_RSTS1	0x(FF)FF_8029	CPU only	Read Status Register 1
SP_RSTS2	0x(FF)FF_802A	CPU only	Read Status Register 2
SP_RSTS3	0x(FF)FF_802B	CPU only	Read Status Register 3
SP_MTOR0	0x(FF)FF_802C	CPU only	Mutext Timeout Register 0
SP_MTOR1	0x(FF)FF_802D	CPU only	Mutext Timeout Register 2
SP_OIC	0x(FF)FF_802E	CPU only	Output Interrupt Control Register

5.3 Data read/write coherency issues

The FXLC95000CL platform handles data coherency issues using two complementary mechanisms; read buffering and semaphores. A four-byte read buffer guarantees that naturally aligned two- and four-byte variables are self-consistent, when read by the master and when two binary semaphores (mutex operators) are available for applications requiring more rigorous control of shared resources.

Mailbox registers are shared between the FXLC95000CL and an external master. Because both master and slave have the ability to read and write these registers, data coherency issues can be handled via read buffering and semaphores.

By its nature, the I^2C interface is based on a byte-wide protocol. This presents a number of challenges when dealing with shared 16-bit and 32-bit data.

Since the SPI interface is designed to support a protocol similar to I^2C, it also presents the same restrictions.

From the perspective of the FXLC95000CL platform, the Version 1 ColdFire will automatically serialize one 16-bit access into two sequential byte accesses. It will also serialize one long word (32-bit) access into four byte-wide sequential accesses. Thus, four eight-bit registers can be written with a single 32-bit write instruction. Serialized accesses work best on addresses aligned to the size of the operand being written.

From the perspective of an external master, four sequential byte writes by the CPU happen quickly. Depending on the frequency of operation, the byte writes may take less time than a single bit in the I^2C or SPI data stream. However it is easy to see that a problem could still occur if the master is reading byte number two of a four-byte variable when the CPU decides to update the variable. Bytes zero and one (already read by the master) would correspond to the two most-significant bytes of the previous value. Byte number three would correspond to the least-significant byte of the next value and byte number two could go either way, depending on precisely when the CPU write occurs with respect to the master read operation.

From an even broader perspective, how does the master know when it is OK to read or write a register that the CPU is updating on some regular basis?

The FXLC95000CL has two complementary mechanisms to address these problems:

1. A four-byte read buffer guarantees that naturally aligned two- and four-byte variables are self-consistent when read by the master.
2. Two binary semaphores (mutex operators) are available for applications requiring more rigorous control of shared resources.

5.3.1 Read buffer

FXLC95000CL guarantees that naturally aligned 2-byte and 4-byte variables are self-consistent when mailboxes are read by the master. To prevent data coherency issues during multi-byte reads, it is important to know legal and illegal allocation of variables to mailbox locations.

When any byte is read by the master, the entire 4-byte region in which that byte resides will be cached in a 4-byte, line buffer. Reads of subsequent bytes will be done from the buffer, ensuring that the master sees consistent data in multiple-byte variables.

This process is best seen by way of example. The Version 1 ColdFire CPU uses big-endian addressing. Referring to Table 5-2, the 4-byte read buffer can be used to make simultaneous reads of the mailboxes in any row of the table.

Table 5-2. Mailbox memory map

MSB Address	MSB			LSB
0x00	SP_MB0	SP_MB1	SP_MB2	SP_MB3
0x04	SP_MB4	SP_MB5	SP_MB6	SP_MB7
0x08	SP_MB8	SP_MB9	SP_MB10	SP_MB11
0x0C	SP_MB12	SP_MB13	SP_MB14	SP_MB15
0x10	SP_MB16	SP_MB17	SP_MB18	SP_MB19
0x14	SP_MB20	SP_MB21	SP_MB22	SP_MB23
0x18	SP_MB24	SP_MB25	SP_MB26	SP_MB27
0x1C	SP_MB28	SP_MB29	SP_MB30	SP_MB31

An example of the legal allocation of variables to these locations is shown in Table 5-3.

Table 5-3. Valid mailbox organization

MSB Address	MSB			LSB
0x00	Variable 1 (MSB)	(MSB+1)	(MSB+2)	Variable 1 (LSB)
0x04	Variable 2 (MSB)	(MSB+1)	(MSB+2)	Variable 2 (LSB)
0x08	Variable 3 (MSB)	(MSB+1)	(MSB+2)	Variable 3 (LSB)
0x0C	Variable 4 (MSB)	(LSB)	Variable 5 (MSB)	(LSB)
0x10	Variable 6 (MSB)	(LSB)	Variable 7 (MSB)	(LSB)
0x14	Variable 8 (MSB)	(LSB)	Variable 9 (MSB)	(LSB)
0x18	Variable 10	Variable 11	Variable 12	Variable 13
0x1C	Variable 14	Variable 15	Variable 16	Variable 17

In Table 5-3, variables 1, 2 and 3 are 32-bit variables. Variables 4 through 9 are 16-bit variables and variables 10 through 17 are 8-bit variables. All variables are guaranteed to have self-consistent values when read by the sensor master.

An invalid allocation of variables would have variables spanning rows as shown in Table 5-4.

Table 5-4. Invalid mailbox organization

MSB Address	MSB			LSB
0x00	Variable 1 (MSB)	(LSB)	Variable 2 (MSB)	(MSB+1)
0x04	Variable 2 (MSB+2)	(LSB)	Variable 3 (MSB)	(MSB+1)
0x08	Variable 3 (MSB+2)	(LSB)	Variable 4 (MSG)	(LSB)
0x0C	Variable 5 (MSB)	(LSB)	Variable 6 (MSB)	(MSB+1)
0x10	Variable 6 (MSB+2)	(LSB)	Variable 7 (MSB)	(LSB)
0x14	Variable 8 (MSB)	(LSB)	Variable 9	Variable 10 (MSB)
0x18	Variable 10 (LSB)	Variable 11	Variable 12	Variable 13
0x1C	Variable 14	Variable 15	Variable 16	Variable 17

Table 5-4 shading shows the improperly aligned variables. Depending on when reads and writes occur, it is possible that the master and CPU would see inconsistent values.

The 4-byte read buffer is cleared when an I^2C STOP condition occurs or when the SPI SSB signal is deasserted, depending upon which port is in use. Contents are replaced whenever the register address increments from one row to the next.

5.3.2 Binary semaphore (mutex) operation

The FXLC95000CL includes two semaphore registers that can be used by its CPU and system master to negotiate ownership of shared assets. Atomic actions assure that the CPU and master can unambiguously negotiate ownership of any asset.

These can be mailbox registers or any other shared item. These registers can be read by only one of the two parties at any given instant in time. Simultaneous attempts will be serialized by the module.

Each semaphore register has several possible actions associated with it, as shown in Table 5-5.

Table 5-5. Semaphore actions

Semaphore Content	Action	Side Effect
0x00	Read 0x00	Set semaphore = 1. Reader now has ownership of shared asset.
0x01	Read 0x01	None. Slave host has ownership of the semaphore.
0x02	Read 0x02	None. CPU has ownership of the semaphore.
0x00	Write any value (Normally, this is only done by the current owner)	Set semaphore = 0 - No action
0x01 or 0x02		Set semaphore = 0 - Ownership has been relinquished

The actions in Table 5-5 are atomic, assuring that the CPU and master can unambiguously negotiate ownership of any asset.

Operation of the two semaphore registers is identical, enabling simultaneous negotiations for two different sets of shared assets.

A classic problem with operation of semaphores occurs when the owner of a shared asset fails to relinquish control, resulting in a "lockout" situation. For this reason, each semaphore has an optional "time-out" register. If enabled, a countdown timer is initiated to the specified value whenever the semaphore is set. The counter then begins counting down. When it hits zero, an interrupt is issued. The CPU should then clear the semaphore.

Each timer is stopped whenever the associated semaphore is cleared.

5.4 Slave memory map and register definition

SLAVE memory map

Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
FFFF_8000	Slave port mailbox registers n (SLAVE_SP_MBn)	8	R/W	00h	5.4.1/72
FFFF_8020	Slave port binary semaphore (mutex) register n (SLAVE_SP_MUTEXn, n=0 or 1)	8	R/W	00h	5.4.2/73
FFFF_8022	Slave port I2C address register (SLAVE_SP_ADDR)	8	R/W	4Ch	5.4.3/74
FFFF_8023	Slave port status and control register (SLAVE_SP_SCR)	8	R/W	00h	5.4.4/74
FFFF_8024	Slave port write status register 0 (SLAVE_SP_WSTS0)	8	R	00h	5.4.5/76
FFFF_8025	Slave port write status register 1 (SLAVE_SP_WSTS1)	8	R	00h	5.4.6/77
FFFF_8026	Slave port write status register 2 (SLAVE_SP_WSTS2)	8	R	00h	5.4.7/78
FFFF_8027	Slave port write status register 3 (SLAVE_SP_WSTS3)	8	R	00h	5.4.8/78
FFFF_8028	Slave port read status register 0 (SLAVE_SP_RSTS0)	8	R	00h	5.4.9/79
FFFF_8029	Slave port read status register 1 (SLAVE_SP_RSTS1)	8	R	00h	5.4.10/80
FFFF_802A	Slave port read status register 2 (SLAVE_SP_RSTS2)	8	R	00h	5.4.11/81
FFFF_802B	Slave port read status register 3 (SLAVE_SP_RSTS3)	8	R	00h	5.4.12/82
FFFF_802C	Slave port mutext timeout register n (SLAVE_SP_MTORn)	8	R/W	00h	5.4.13/82
FFFF_802E	Slave port output interrupt control register (SLAVE_SP_OIC)	8	R/W	00h	5.4.14/83

5.4.1 Slave port mailbox registers n (SLAVE_SP_MBn)

The slave port is the main communication channel between the system master and the FXLC95000CL. SP_MB0 through SP_MB31 are bidirectional mailboxes and can be software-configured to support any number of applications. Each mailbox can be read/ written by both the CPU and the serial interface master.

The write status registers (SP_WSTS0, 1, 2, and 3) can be used to notify the CPU that the master has written to one or more mailbox registers. Whenever any of the mailbox registers has been written, the bit corresponding to that register in the SP_WSTS*x* registers will be set. The slave port can be configured to generate an interrupt when this occurs.

SP_MB31 and SP_MB15 can be programmed via the SP_SCL[WWUP] bits to operate as a special "wake-up" register that will wake the CPU from STOP mode when written. This operation can be extended to *all* of the SP_MB*n* registers or none, as desired.

Reads by both CPU and the serial interface master are non-blocking and can occur simultaneously. CPU writes may be delayed a cycle to allow master writes time to complete.

NOTE

Simultaneous writes represent a hazard condition. Developers are encouraged to restrict write access to either CPU or sensor master on a register-by-register basis.

Address: FFFF_8000h base + 0h offset = FFFF_8000h

Bit	7	6	5	4	3	2	1	0
Read Write	DATA							
Reset	0	0	0	0	0	0	0	0

SLAVE_SP_MBn field descriptions

Field	Description
DATA	See detailed description above.

5.4.2 Slave port binary semaphore (mutex) register n (SLAVE_SP_MUTEXn, n=0 or 1)

SP_MUXTEX0 and SP_MUXTEX1 are identical in form and operation. For additional information regarding operation of the semaphore registers, see Binary semaphore (mutex) operation.

Address: FFFF_8000h base + 20h offset = FFFF_8020h

Bit	7	6	5	4	3	2	1	0
Read	R						SSTS	
Write								
Reset	0	0	0	0	0	0	0	0

SLAVE_SP_MUTEXn, n=0 or 1 field descriptions

Field	Description
7–2 R	Reserved - RO bits
SSTS	Semaphore Status A write of any value to this register sets STS to 0. There is no hardware lock to ensure that the current owner of the semaphore is the one to unlock it. It is the responsibility of the software to enforce this practice.

Table continues on the next page...

SLAVE_SP_MUTEXn, n=0 or 1 field descriptions (continued)

Field	Description
	00 A read value of 00 returned in this field indicates that the reader now has ownership of the shared asset. STS will subsequently read as 01 or 10 until relinquished. 01 A read value of 01 returned in this field indicates that the semaphore has previously been claimed by the host controlling the slave port. 10 A read value of 10 returned in this field indicates that the semaphore has previously been claimed by the CPU.

5.4.3 Slave port I2C address register (SLAVE_SP_ADDR)

The seven-bit address for this port can be specified by writing to this register (SLAVE_SP_ADDR), which should be configured in advance of any I^2C traffic on the slave port unless the default value (0x4C) is used.

This register has no function when a SPI interface is used. In that case, the register is *reserved.*

Address: FFFF_8000h base + 22h offset = FFFF_8022h

Bit	7	6	5	4	3	2	1	0
Read Write	Reserved				ADDR			
Reset	0	1	0	0	1	1	0	0

SLAVE_SP_ADDR field descriptions

Field	Description
7 R	This field is reserved. Reserved
ADDR	See detailed description above.

5.4.4 Slave port status and control register (SLAVE_SP_SCR)

Address: FFFF_8000h base + 23h offset = FFFF_8023h

Bit	7	6	5	4	3	2	1	0
Read Write	EN	PS	ACTIVE/ CSR	STOP_EN	RIE	WIE	WUP	
Reset	0	0	0	0	0	0	0	0

SLAVE_SP_SCR field descriptions

Field	Description
7 EN	Slave Port Enable This bit it initialized to 1 by the boot manager at startup.

Table continues on the next page...

SLAVE_SP_SCR field descriptions (continued)

Field	Description
	EN should be switched high only when the slave port bus is known to be inactive. 0 Slave port is not enabled. Use this only for the instance where this device has no host controller. 1 Either SPI or I^2C slave port is enabled (based on choice of PS bit).
6 PS	Port Select On devices that support only one of the SPI or I^2C interfaces, this register bit is read-only. On the FXLC95000CL , the PS bit is initialized by the boot manager at startup. If RGPIO8 is found to be low during the boot process (immediately following any reset), the PS is set to 1 (SPI mode). Otherwise, the I^2C mode is assumed. The PS bit can be changed at later times as long as the slave port is disabled. **NOTE:** Normally, both EN and PS are "set and forget" controls. Indiscriminately switching between modes may result in loss of data unless extreme care is taken at the system level. 0 The I^2C interface has been selected for use as slave port. 1 The SPI interface has been selected for use as slave port.
5 ACTIVE/CSR	ACTIVE RO Slave port is active The function of this bit is dependent on the state of the PS bit. This bit is not applicable with EN is 0. CSR WO Clear Read and Write Status Registers This bit always reads as 0. Write a 1 to clear all four write status registers and all four read status registers. 0 Clear Read and Write Status Registers 1 Clear SP_WSTS0, 1, 2 and 3 and SP_RSTS0, 1, 2 and 3.
4 STOP_EN	Interrupt STOP Enable 0 Interrupts are not enabled during STOP. Assertion of interrupts is deferred until the device exits STOP mode. 1 Interrupts are generated in STOP mode.
3 RIE	Read Interrupt Enable The read status registers record master writes on a register-by-register basis. Write a 1 to the CSR bit to clear interrupts enabled by RIE. 0 Read interrupt is not enabled. 1 Enable CPU interrupt when one or more of the SP_MB*n* registers have been read by the system master. Interrupt operation is further qualified by the WUP bits.
2 WIE	Write Interrupt Enable The write status registers record master writes on a register-by-register basis. Write a 1 to the CSR bit to clear interrupts enabled by WIE. 0 Write interrupt is not enabled. 1 Enable CPU interrupt when one or more of the SP_MB*n* registers have been written by the system master. Interrupt operation is further qualified by the WUP bits.
WUP	Wake-up Configuration

Table continues on the next page...

SLAVE_SP_SCR field descriptions (continued)

Field	Description
	The ability of the write-interrupt function to interrupt the CPU is defined by the WUP bits, in conjunction with RIE, WIE and STOP_EN. This field is applicable only when RIE and/or WIE have been set to 1. Interrupts are deferred until STOP mode is exited if STOP_EN is equal to 0. 00 Generate interrupt (subject to RIE, WIE and STOP_EN) on any mailbox access. Interrupt generation occurs at the end of the packet transmission. Only one interrupt is generated, even if multiple mailboxes are accessed. 01 Generate interrupt (subject to RIE, WIE and STOP_EN) only on access to Mailbox 15. Interrupt generation occurs immediately upon completion of the access to Mailbox 15. 10 Generate interrupt (subject to RIE, WIE and STOP_EN) only on access to Mailbox 31. Interrupt generation occurs immediately upon completion of the access to Mailbox 31. 11 Reserved

5.4.5 Slave port write status register 0 (SLAVE_SP_WSTS0)

The write status registers are used to track write activity by the system master on the slave bus. Each of the 32 bits in these registers corresponds to exactly one of the SP_MB*n* registers. D0 maps to SP_MB0, D1 to SP_MB1 and so forth. The "D" bits are set whenever the corresponding register is written by the system master.

Software running on the CPU can read these registers to determine which mailboxes have been updated by the master.

Address: FFFF_8000h base + 24h offset = FFFF_8024h

Bit	7	6	5	4	3	2	1	0
Read	D31	D30	D29	D28	D27	D26	D25	D24
Write								
Reset	0	0	0	0	0	0	0	0

SLAVE_SP_WSTS0 field descriptions

Field	Description
7 D31	Maps to SP_MB31 Bit sets when SP_MB31 is written by the system master.
6 D30	Maps to SP_MB30 Bit sets when SP_MB30 is written by the system master.
5 D29	Maps to SP_MB29 Bit sets when SP_MB29 is written by the system master.
4 D28	Maps to SP_MB28 Bit sets when SP_MB28 is written by the system master.
3 D27	Maps to SP_MB27 Bit sets when SP_MB27 is written by the system master.

Table continues on the next page...

SLAVE_SP_WSTS0 field descriptions (continued)

Field	Description
2 D26	Maps to SP_MB26 Bit sets when SP_MB26 is written by the system master.
1 D25	Maps to SP_MB25 Bit sets when SP_MB25 is written by the system master.
0 D24	Maps to SP_MB24 Bit sets when SP_MB24 is written by the system master.

5.4.6 Slave port write status register 1 (SLAVE_SP_WSTS1)

Address: FFFF_8000h base + 25h offset = FFFF_8025h

Bit	7	6	5	4	3	2	1	0
Read	D23	D22	D21	D20	D19	D18	D17	D16
Write								
Reset	0	0	0	0	0	0	0	0

SLAVE_SP_WSTS1 field descriptions

Field	Description
7 D23	Maps to SP_MB23 Bit sets when SP_MB23 is written by the system master.
6 D22	Maps to SP_MB22 Bit sets when SP_MB22 is written by the system master.
5 D21	Maps to SP_MB21 Bit sets when SP_MB21 is written by the system master.
4 D20	Maps to SP_MB20 Bit sets when SP_MB20 is written by the system master.
3 D19	Maps to SP_MB19 Bit sets when SP_MB19 is written by the system master.
2 D18	Maps to SP_MB18 Bit sets when SP_MB18 is written by the system master.
1 D17	Maps to SP_MB17 Bit sets when SP_MB17 is written by the system master.
0 D16	Maps to SP_MB16 Bit sets when SP_MB16 is written by the system master.

5.4.7 Slave port write status register 2 (SLAVE_SP_WSTS2)

Address: FFFF_8000h base + 26h offset = FFFF_8026h

Bit	7	6	5	4	3	2	1	0
Read	D15	D14	D13	D12	D11	D10	D9	D8
Write								
Reset	0	0	0	0	0	0	0	0

SLAVE_SP_WSTS2 field descriptions

Field	Description
7 D15	Maps to SP_MB15 Bit sets when SP_MB15 is written by the system master.
6 D14	Maps to SP_MB14 Bit sets when SP_MB14 is written by the system master.
5 D13	Maps to SP_MB13 Bit sets when SP_MB13 is written by the system master.
4 D12	Maps to SP_MB12 Bit sets when SP_MB12 is written by the system master.
3 D11	Maps to SP_MB11 Bit sets when SP_MB11 is written by the system master.
2 D10	Maps to SP_MB10 Bit sets when SP_MB10 is written by the system master.
1 D9	Maps to SP_MB9 Bit sets when SP_MB9 is written by the system master.
0 D8	Maps to SP_MB8 Bit sets when SP_MB8 is written by the system master.

5.4.8 Slave port write status register 3 (SLAVE_SP_WSTS3)

Address: FFFF_8000h base + 27h offset = FFFF_8027h

Bit	7	6	5	4	3	2	1	0
Read	D7	D6	D5	D4	D3	D2	D1	D0
Write								
Reset	0	0	0	0	0	0	0	0

SLAVE_SP_WSTS3 field descriptions

Field	Description
7 D7	Maps to SP_MB7

Table continues on the next page...

SLAVE_SP_WSTS3 field descriptions (continued)

Field	Description
	Bit sets when SP_MB7 is written by the system master.
6 D6	Maps to SP_MB6 Bit sets when SP_MB6 is written by the system master.
5 D5	Maps to SP_MB5 Bit sets when SP_MB5 is written by the system master.
4 D4	Maps to SP_MB4 Bit sets when SP_MB4 is written by the system master.
3 D3	Maps to SP_MB3 Bit sets when SP_MB3 is written by the system master.
2 D2	Maps to SP_MB2 Bit sets when SP_MB2 is written by the system master.
1 D1	Maps to SP_MB1 Bit sets when SP_MB1is written by the system master.
0 D0	Maps to SP_MB0 Bit sets when SP_MB0 is written by the system master.

5.4.9 Slave port read status register 0 (SLAVE_SP_RSTS0)

The read status registers are used to track read activity by the system master on the slave bus. Each of the 32 bits in these registers corresponds to exactly one of the SP_MB*n* registers. D0 maps to SP_MB0, D1 to SP_MB1 and so forth. The "D" bits are set whenever the corresponding register is read by the system master.

Software running on the CPU can read these registers to determine which mailboxes have been inspected by the master.

Address: FFFF_8000h base + 28h offset = FFFF_8028h

Bit	7	6	5	4	3	2	1	0
Read	D31	D30	D29	D28	D27	D26	D25	D24
Write								
Reset	0	0	0	0	0	0	0	0

SLAVE_SP_RSTS0 field descriptions

Field	Description
7 D31	Maps to SP_MB31 Bit sets when SP_MB31 is read by the system master.
6 D30	Maps to SP_MB30 Bit sets when SP_MB30 is read by the system master.

Table continues on the next page...

SLAVE_SP_RSTS0 field descriptions (continued)

Field	Description
5 D29	Maps to SP_MB29 Bit sets when SP_MB29 is read by the system master.
4 D28	Maps to SP_MB28 Bit sets when SP_MB28 is read by the system master.
3 D27	Maps to SP_MB27 Bit sets when SP_MB27 is read by the system master.
2 D26	Maps to SP_MB26 Bit sets when SP_MB26 is read by the system master.
1 D25	Maps to SP_MB25 Bit sets when SP_MB25 is read by the system master.
0 D24	Maps to SP_MB24 Bit sets when SP_MB24 is read by the system master.

5.4.10 Slave port read status register 1 (SLAVE_SP_RSTS1)

Address: FFFF_8000h base + 29h offset = FFFF_8029h

Bit	7	6	5	4	3	2	1	0
Read	D23	D22	D21	D20	D19	D18	D17	D16
Write								
Reset	0	0	0	0	0	0	0	0

SLAVE_SP_RSTS1 field descriptions

Field	Description
7 D23	Maps to SP_MB23 Bit sets when SP_MB23 is read by the system master.
6 D22	Maps to SP_MB22 Bit sets when SP_MB22 is read by the system master.
5 D21	Maps to SP_MB21 Bit sets when SP_MB21 is read by the system master.
4 D20	Maps to SP_MB20 Bit sets when SP_MB20 is read by the system master.
3 D19	Maps to SP_MB19 Bit sets when SP_MB19 is read by the system master.
2 D18	Maps to SP_MB18 Bit sets when SP_MB18 is read by the system master.
1 D17	Maps to SP_MB17 Bit sets when SP_MB17 is read by the system master.

Table continues on the next page...

SLAVE_SP_RSTS1 field descriptions (continued)

Field	Description
0 D16	Maps to SP_MB16 Bit sets when SP_MB16 is read by the system master.

5.4.11 Slave port read status register 2 (SLAVE_SP_RSTS2)

Address: FFFF_8000h base + 2Ah offset = FFFF_802Ah

Bit	7	6	5	4	3	2	1	0
Read	D15	D14	D13	D12	D11	D10	D9	D8
Write								
Reset	0	0	0	0	0	0	0	0

SLAVE_SP_RSTS2 field descriptions

Field	Description
7 D15	Maps to SP_MB15 Bit sets when SP_MB15 is read by the system master.
6 D14	Maps to SP_MB14 Bit sets when SP_MB14 is read by the system master.
5 D13	Maps to SP_MB13 Bit sets when SP_MB13 is read by the system master.
4 D12	Maps to SP_MB12 Bit sets when SP_MB12 is read by the system master.
3 D11	Maps to SP_MB11 Bit sets when SP_MB11 is read by the system master.
2 D10	Maps to SP_MB10 Bit sets when SP_MB10 is read by the system master.
1 D9	Maps to SP_MB9 Bit sets when SP_MB9 is read by the system master.
0 D8	Maps to SP_MB8 Bit sets when SP_MB8 is read by the system master.

5.4.12 Slave port read status register 3 (SLAVE_SP_RSTS3)

Address: FFFF_8000h base + 2Bh offset = FFFF_802Bh

Bit	7	6	5	4	3	2	1	0
Read	D7	D6	D5	D4	D3	D2	D1	D0
Write								
Reset	0	0	0	0	0	0	0	0

SLAVE_SP_RSTS3 field descriptions

Field	Description
7 D7	Maps to SP_MB7 Bit sets when SP_MB7 is read by the system master.
6 D6	Maps to SP_MB6 Bit sets when SP_MB6 is read by the system master.
5 D5	Maps to SP_MB5 Bit sets when SP_MB5 is read by the system master.
4 D4	Maps to SP_MB4 Bit sets when SP_MB4 is read by the system master.
3 D3	Maps to SP_MB3 Bit sets when SP_MB3 is read by the system master.
2 D2	Maps to SP_MB2 Bit sets when SP_MB2 is read by the system master.
1 D1	Maps to SP_MB1 Bit sets when SP_MB1is read by the system master.
0 D0	Maps to SP_MB0 Bit sets when SP_MB0 is read by the system master.

5.4.13 Slave port mutext timeout register n (SLAVE_SP_MTORn)

The two semaphores are equipped with identical timeout registers of the format shown below. SP_MTOR0 controls time-out functions for Semaphore 0 and SP_MTOR1 controls time-out functions for Semaphore 1.

The Mutext timers are based on the standard peripheral clock. They are used to limit semaphore operations within the scope of a single frame. They will not increment if the slave port peripheral clock is disabled during STOP modes. Nor will they be triggered if a host write occurs to the semaphores while the peripheral clock is stopped. Peripheral clock operation in STOP modes is controlled via the peripheral-clock-enable registers in the SIM.

It is recommended that timeout periods extending beyond one frame be implemented in software as part of the start of frame interrupt, rather than being implemented through use of the time-out registers.

Address: FFFF_8000h base + 2Ch offset = FFFF_802Ch

Bit	7	6	5	4	3	2	1	0
Read	R	TOSTS	EN	MTE				
Write								
Reset	0	0	0	0	0	0	0	0

SLAVE_SP_MTORn field descriptions

Field	Description
7 R	Reserved
6 TOSTS	Mutext Timeout Status Each semaphore has an optional "time-out" register. If enabled, a countdown timer is initiated to the specified value whenever the semaphore is set. The counter then begins counting down. When it hits zero, an interrupt is issued. The CPU should then clear the semaphore, which will also clear this field. Each timer is stopped whenever the associated semaphore is cleared. 0 Mutext time-out expiration is not asserted. 1 Mutext time-out expiration interrupt has been asserted.
5 EN	Mutext 0 Timeout Enable 0 Mutext timeout is not enabled. 1 Mutext timer and interrupt are enabled.
MTE	Mutext Timeout Exponent The length of the timeout = 128 x 2^{MTE} x $P_{osc\text{-}high}$.

5.4.14 Slave port output interrupt control register (SLAVE_SP_OIC)

RGPIO5 (Pin 11) can be reprogrammed to function as an interrupt output pin. This interrupt is intended to be asserted when a command-response packet has been stored in the slave-port mailboxes and is ready for the host to read. The host will see the interrupt and proceed to read the data.

Once the FXLC95000CL recognizes that the response packet is being transmitted, it will automatically clear the interrupt. The clearing action occurs while the packet is still underway. This prevents the host from falsely recognizing the same interrupt after the packet is complete.

The hardware will clear the interrupt on one of the following:

- SSB = 0 (SPI mode)
- Slave port I^2C address matches address in packet header (I^2C mode)

Address: FFFF_8000h base + 2Eh offset = FFFF_802Eh

Bit	7	6	5	4	3	2	1	0
Read Write	Reserved					POL	CLR	Read INT_O Write SET
Reset	0	0	0	0	0	0	0	0

SLAVE_SP_OIC field descriptions

Field	Description
7–3 R	This field is reserved. Reserved
2 POL	Output Polarity The POL bit should only be changed when CLR is being written as 1. This forces INT_O to its new de-asserted state. 0 Output is active high. (Interrupt asserted = high on output pin.) 1 Output is active low. (Interrupt asserted = low on output pin.)
1 CLR	Clear/De-assert (INT_O = POL) The CLR bit must be set =0 before setting SET=1. Do not set CLR=0 and SET=1 in the same instruction. 0 No action 1 De-assert INT_O (INT_O = POL)
0 Read INT_O Write SET	Output Interrupt or Set This bit reflects the current state of the INT_O function. The interrupt assertion state depends also on the POL field as shown in Table 5-21. SET Set/Assert (INT_O = NOT POL) 0 Writing 0 No action 1 Writing 1 Assert INT_O (INT_O = NOT POL)

5.4.15 Output interrupt details

The following table shows the current value of INT_O function.

Table 5-21. Interrupt logic value as a function of the POL bit

POL	INT_O	Assertion State
0	0	Not asserted
0	1	Asserted
1	0	Asserted
1	1	Not asserted

The following table defines INT_O behavior as a function of SET, CLR and the hardware-clear events.

Table 5-22. INT_O functional truth table

RESETB	SET	CLR	Hardware Clear Event	INT_O
0	N/A	0	N/A	POL = 0
1	Don't care	1	Don't care	POL
1	WRITE 1	0	0	NOT POL
1	No change	0	1	POL
1	No change	0	0	Previous INT_O

Figure 5-19 and Figure 5-20 show the I^{2}C slave port traffic versus interrupt assertion and de-assertion for both active high and active low interrupt outputs.

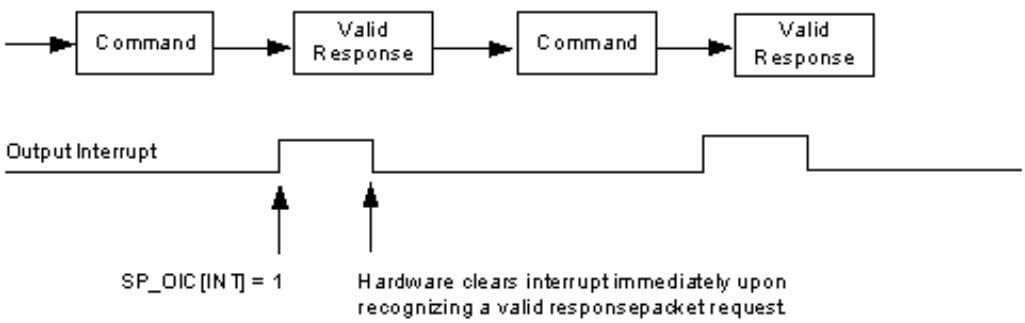


Figure 5-19. Output interrupt timing (POL = 0)

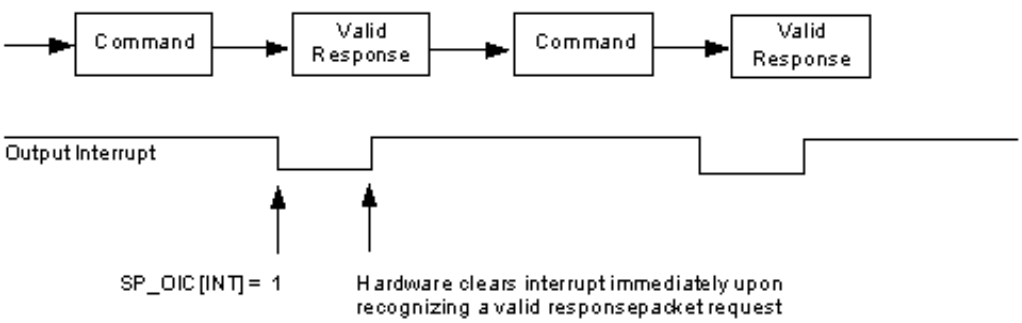


Figure 5-20. Output interrupt timing (POL = 1)

The interrupt output function is not the default function of Pin 11, which defaults to RGPIO5 (in input state with pull-up resistors disabled). It is important that the device receiving the interrupt has that interrupt input disabled until after the POL bit has been changed and SIM_PMCR2 has been reprogrammed to drive Pin 11 with the INT_O function. Alternately, an external pull-up/down resistor can be added to the pin to ensure the correct reset state. reproduces the SIM_PMCR2 settings required to program Pin 11 as INT_O. (See Pin Mux Control Register2 (SIM_PMCR2) for additional details.)

Table 5-23. Pin 11 mux controls

RESETB	SIM_PMCR2	PIN_11
0	0x00	RGPIO input
1	0x00	RGPIO
1	0x01	PDB Output A
1	0x10	INT_O
	0x11	Not defined

5.5 I²C serial protocol and timing

To send data to and receive data from FXLC95000CL when using I2C, the host must follow specific protocols and adhere to its timing characteristics.

5.5.1 Baud rates

I²C supports several ranges of operation. From the perspective of a slave device such as this module, the protocol is the same, regardless of speed. Differences in the I²C spec arise in terms of noise suppression, bit times and electrical drivers, but the logical behavior of the slave is consistent across the modes.

The FXLC95000CL utilizes an internal, 16 MHz CPU and peripheral clock, yielding an internal clock rate of 62.5 ns. At this speed, a minimum (60 ns) SCL, high signal cannot be reliably sampled in high-speed mode. Therefore, the communication rate for the I²C is limited to 2 MHz or less.

The FXLC95000CL data sheet summarizes timing options and requirements for the slave I²C module.

5.5.2 I²C serial-addressing

The FXLC95000CL operates as a slave that sends and receives data through an I²C, two-wire interface. The interface uses a serial data line (SDA) and a serial clock line (SCL) to achieve bi-directional communication between master(s) and slave(s). A master (typically a microcontroller) initiates all data transfers to and from the FXLC95000CL and generates the SCL clock that synchronizes the data transfer.

Table 5-24. SDA and SCL line operation

Pin	Directional attribute	Pull-up resistor
SDA	Input and open-drain output	typically 4.7 kΩ is required
SCL	Input only	typically 4.7 kΩ is required if there are multiple masters on the two-wire interface or if the master in a single-master system has an open-drain SCL output

5.5.3 I²C START, STOP and REPEATED START conditions

Each transmission consists of a START condition (Figure 5-21) sent by a master, followed by the device seven-bit slave address and a read/write bit, a register address byte, one or more data bytes and, finally, a STOP or REPEATED START bit.

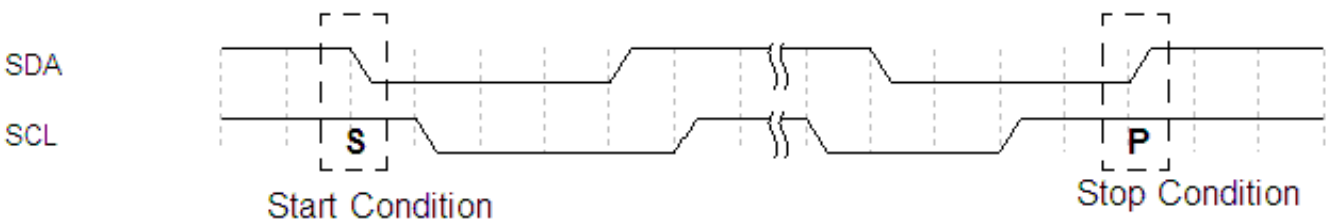


Figure 5-21. START (S) and STOP (P) conditions

Both SCL and SDA remain high when the interface is not busy. A master signals the beginning of a transmission with a START (S) condition by transitioning SDA from high to low while SCL is high. When the master has finished communicating with the slave, it issues a STOP (P) condition by transitioning SDA from low to high while SCL is high.

The bus is then free for another transmission. Alternately, instead of STOP, the master can continue to control the bus by transmitting a repeated START bit instead of the STOP bit. The repeated START condition is functionally identical to a START condition. See Figure 5-22.

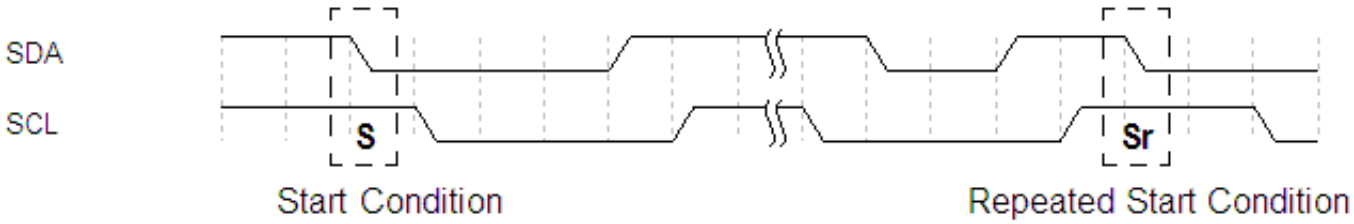


Figure 5-22. START (S) and REPEATED START (Sr) conditions

5.5.4 I²C bit transfer

One data bit is transferred during each clock pulse (Figure 5-23). The data on SDA must remain stable while SCL is high.

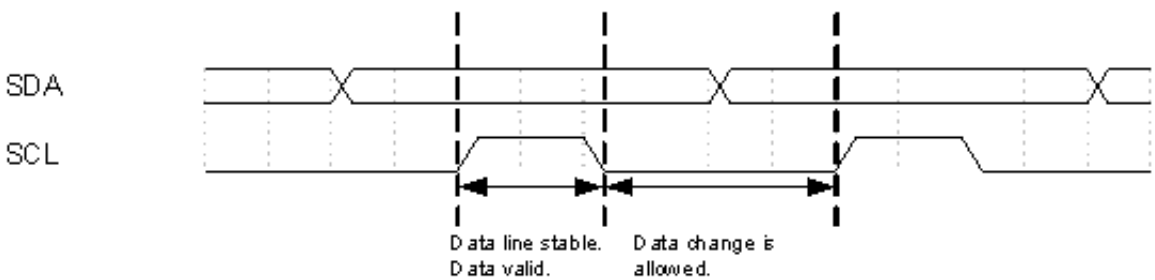


Figure 5-23. Bit transfer

5.5.5 I²C acknowledge

The acknowledge bit is the clocked ninth bit (Figure 5-24) that the recipient uses to handshake receipt of each byte of data. Thus each byte transferred effectively requires nine bits. The master generates the ninth clock pulse and the recipient pulls down SDA during the acknowledge clock pulse. That makes the SDA line stable low during the high period of the clock pulse. When the master is transmitting to the FXLC95000CL, then the FXLC95000CL generates the acknowledge bit because the FXLC95000CL is the recipient.

When the FXLC95000CL is transmitting to the master, then the master generates the acknowledge bit because the master is the recipient.

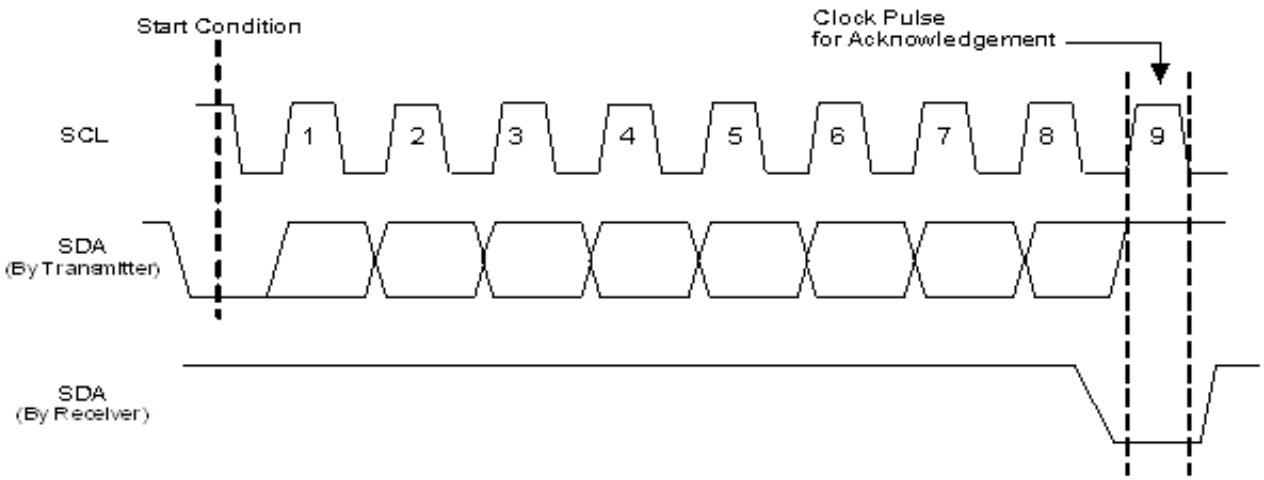


Figure 5-24. I²C acknowledge

5.5.6 I²C slave address

The SP_ADDR register must be programmed via software prior to start of any I²C communications unless the default value (0x4C) is used.

FXLC95000CL has a seven-bit long slave address (Figure 5-25). The bit following the seven-bit slave address (Bit 8) is the read/write bit, which is low for a write command and high for a read command. The device address for the FXLC95000CL is software-programmable via the SP_ADDR register. This value must be programmed prior to start of any I²C communications unless the default value (0x4C) is used.

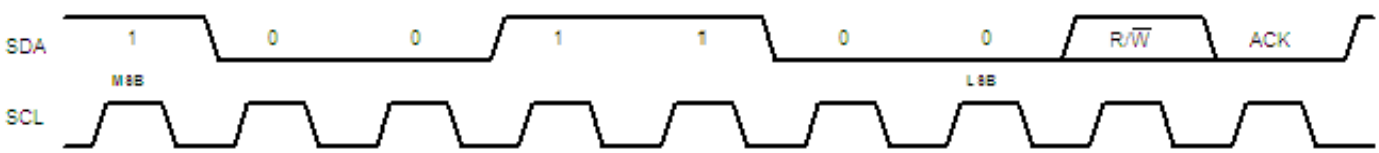


Figure 5-25. I²C slave address

The FXLC95000CL monitors the bus continuously, waiting for a START condition followed by its slave address. When it recognizes its slave address, it acknowledges that communication and is ready for continued communication.

5.5.7 I²C message format for writing

The figures in this section make use of the following notation:

- S Start Bit/Repeated Start Bit
- A Acknowledge Bit
- A Not-Acknowledge Bit
- P Stop Bit

A write to the FXLC95000CL comprises the transmission of the FXLC95000CL device's slave address with the read/write bit set to 0, followed by at least one byte of information. The first byte of information is the register address of the first internal register (0x00 to 0x21) that is to be updated.

If a STOP condition is detected after just the register address is received, the FXLC95000CL takes no action. This is shown in Figure 5-26.

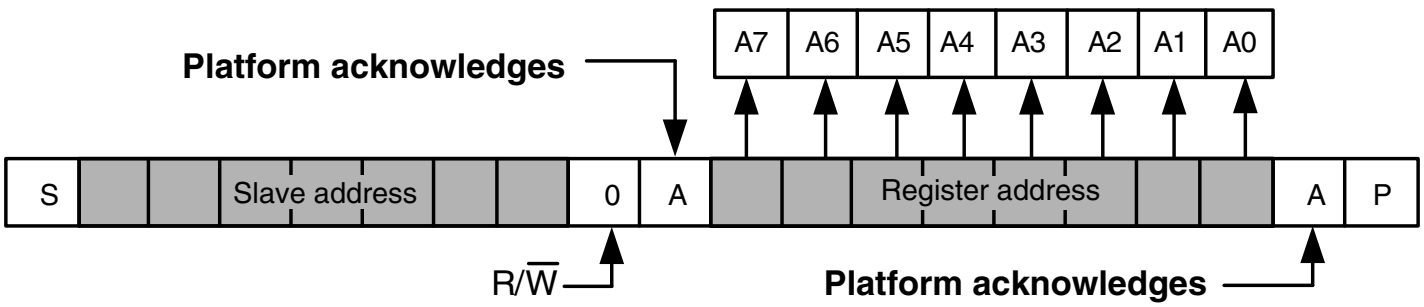


Figure 5-26. Minimal I²C command has no effect

The FXLC95000CL clears its internal register address pointer to register 0x00 when a STOP condition is detected, so a single-byte write has no net effect because the register address given in this first-and-only byte is replaced by 0x00 at the STOP condition.

The internal register address pointer is *not*, however, cleared on a repeated START condition. Figure 5-27 shows the simplest case where a single register value is read.

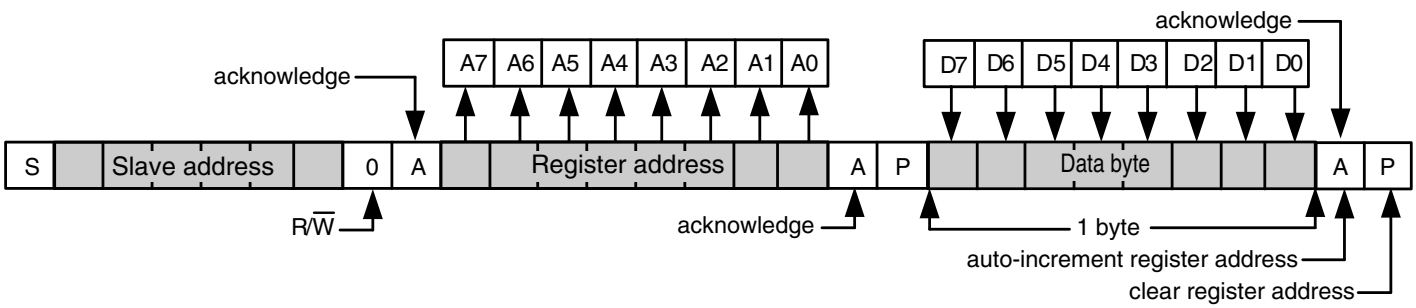


Figure 5-27. Writing one byte of information into slave I²C

Any bytes received after the register address are data bytes. The first data byte goes into the internal register of the FXLC95000CL that was designated by the register address. If multiple data bytes are transmitted before a STOP condition is detected, these bytes are

generally stored in subsequent FXLC95000CL internal registers because the register address increments after each access. This auto-increment feature works identically for read and write operations.

The address will "wrap" around to Mailbox Register 0 once the Mailbox Register 31 is accessed.

Figure 5-28 illustrates the case where three bytes of information are written into three consecutive, slave I^{2}C registers.

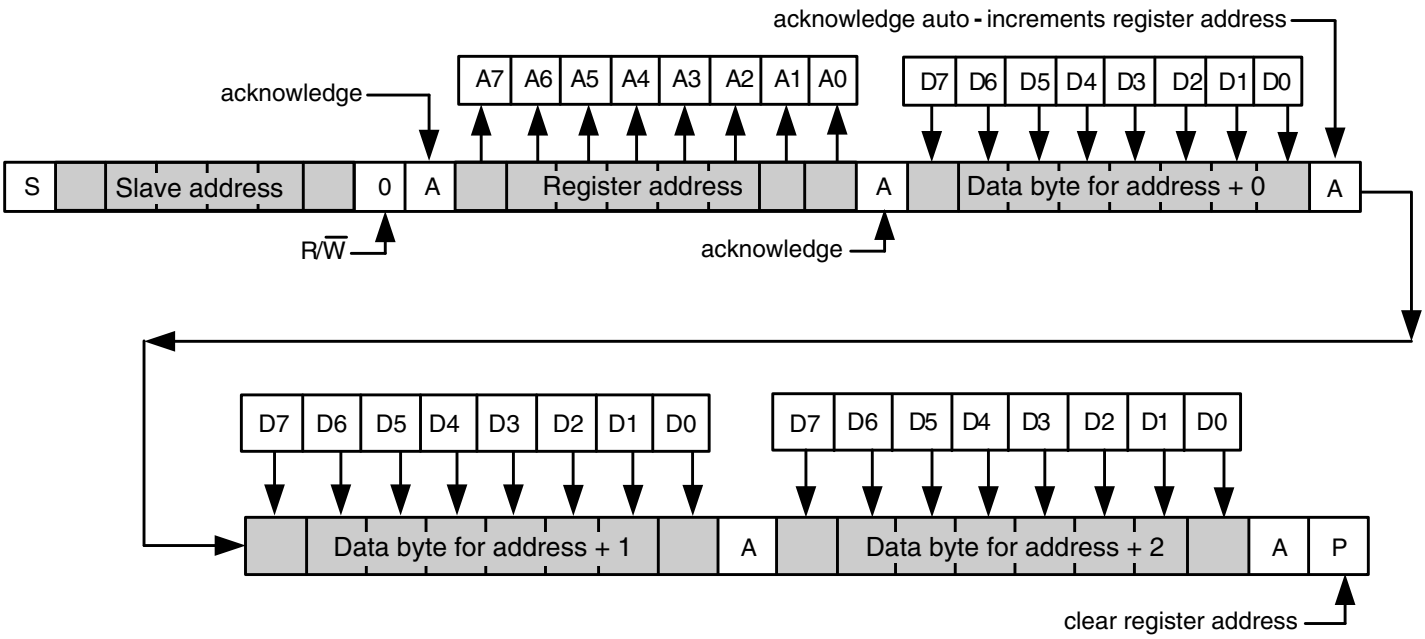


Figure 5-28. Three consecutive byte writes to the slave I^{2}C data registers

5.5.8 I^{2}C message format for reading

Read instructions by the master involve writing the register address and reading the contents of one or more registers. In the first part of the sequence, the bus master is placing information on the bus. In the second, the FXLC95000CL is placing information on the bus. The I^{2}C standard refers to this type of instruction as a "combined format" because the initial write of the device and register addresses must be followed by a read operation.

This is done by following the register address with a repeated START, a repeat of the device address (but now with RWB = 1), an acknowledgement and a data read.

The sequence outlined above is shown in Figure 5-29.

The FXLC95000CL is read using its internally stored register address as an address pointer, the same way the stored register address is used as address pointer for a write. The pointer generally auto-increments after each data byte is read using the same rules as for a write. The address will "wrap" around to Mailbox Register 0 once Mailbox Register 31 is accessed.

The master can now read "*n*" consecutive bytes from FXLC95000CL, with the first data byte being read from the register addressed by the initialized register address.

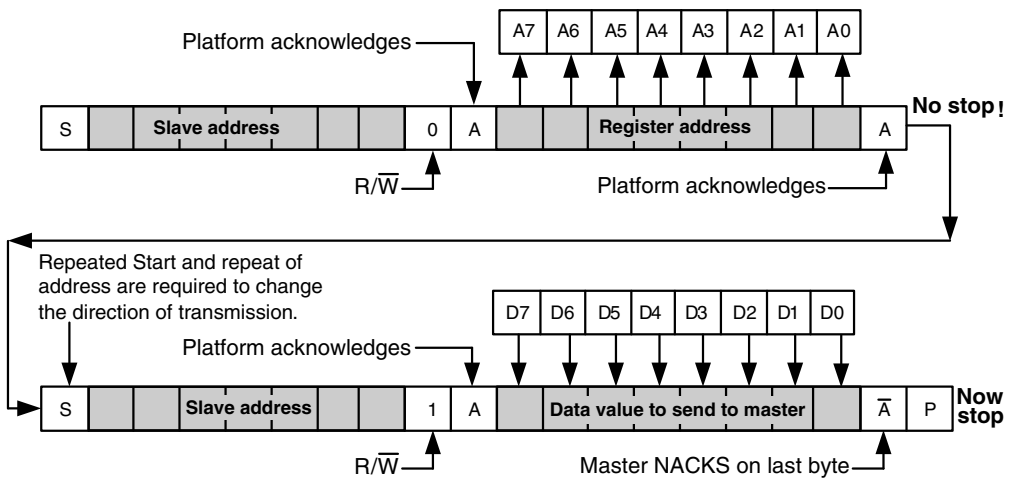


Figure 5-29. Read of a single byte of information

The I^{2}C standard specifies that a master-receiver must signal the end of transfer to a slave transmitter via generation of a NACK (rather than ACK) prior to the Stop bit. This is shown in Figure 5-29.

Figure 5-30 extends the read sequence to access three bytes of information using auto-incrementing of the address register.

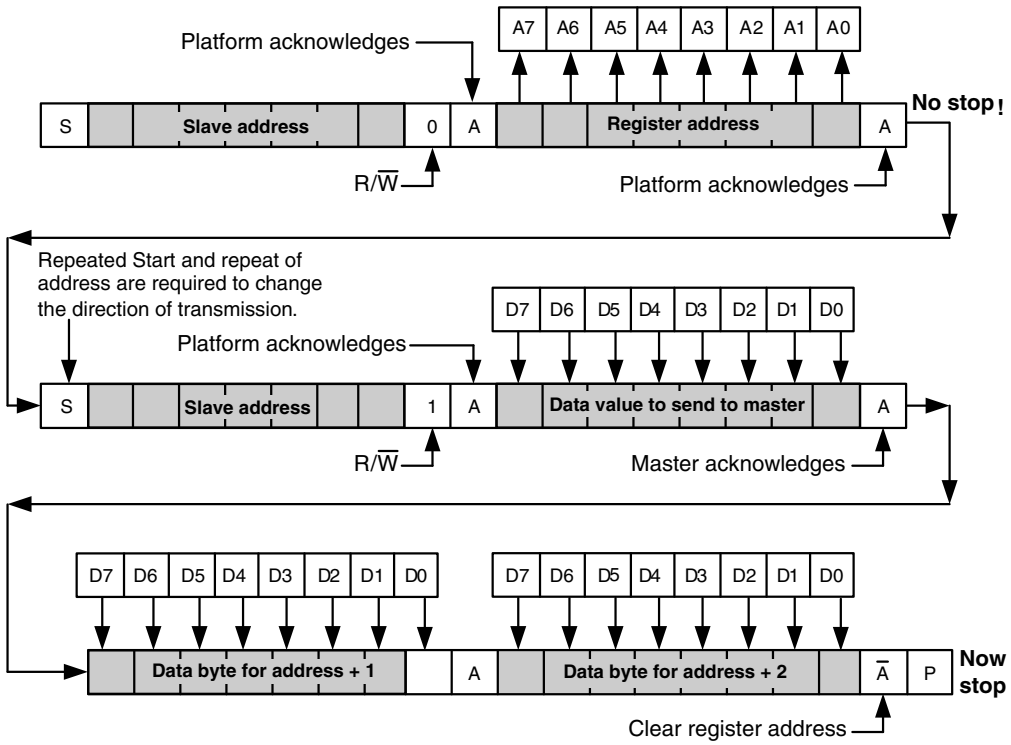


Figure 5-30. Repeated read of 3 bytes of information

5.6 SPI serial protocol and timing

To send data to and receive data from FXLC95000CL when using SPI, the host must follow specific protocols and adhere to its timing characteristics.

The SPI interface consists of two control lines (SSB, SCLK) and two data lines (MOSI, MISO).

- SSB (also known as "Slave Select", active low) is the slave device enable, and is controlled by the SPI master. SSB is driven low at the start of a transmission, and driven high at the end of a transmission.
- The SCLK control line is the SPI clock, and is also controlled by the SPI master.
- MISO is the SPI Data Input.
- MOSI is the SPI Data Output. The MISO and MOSI data lines are driven at the falling edge of the SCLK, and should be captured at the rising edge of the SCLK

Read and write register commands are completed in 16 clock pulses or in multiples of 8 clock pulses (in the case of a multiple-byte read/write).

5.6.1 SPI read operation

Figure 5-31 shows a read of a single register on the SPI port. A SPI read transfer requires that 2 bytes be transmitted on MOSI by the host. The first consists of:

- A 1-bit Read/Write signal (0 = Read, 1 = Write)
- A 6-bit address
- A 1-bit don't-care bit

The second byte on MOSI is discarded by the FXLC95000CL and is shown as "dc" (Don't Care).

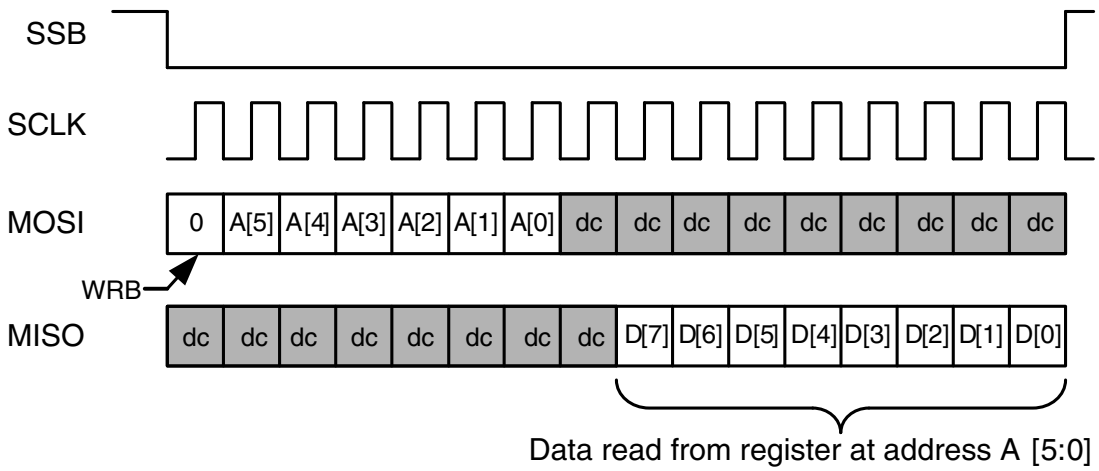


Figure 5-31. Single-byte read by host using SPI

At the beginning of the transmission, the FXLC95000CL does not know the type of transfer that is coming. The first byte of information on MISO is therefore useless and should be discarded.

Transmission is initiated when the host drives SSB low and the transmission terminates when SSB is driven back high.

Packet payloads are an integer number of bytes long. The first address to be read is transmitted in the first byte of information placed on MOSI by the master. Subsequent read addresses are automatically indexed by one.

Figure 5-32 illustrates the case where the payload is 3 bytes long.

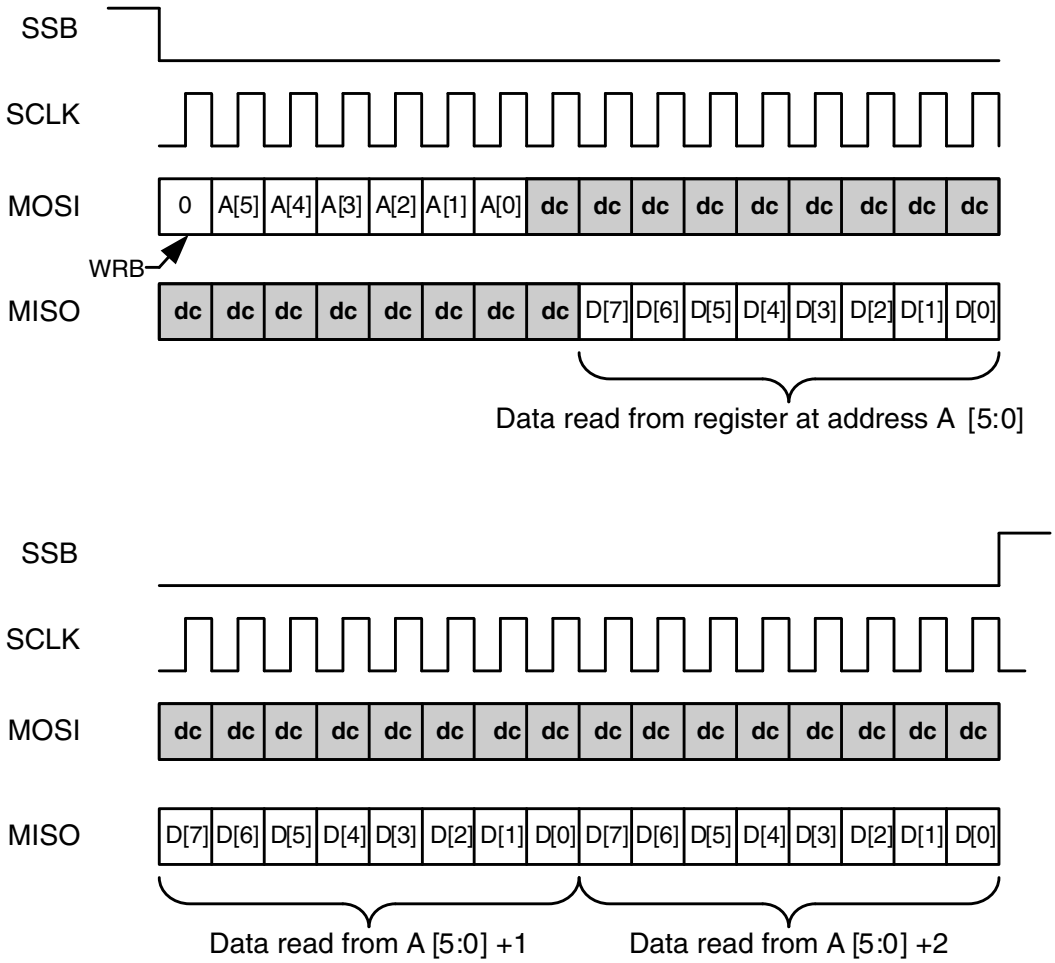


Figure 5-32. 3-byte read by host using SPI

5.6.2 SPI write operation

To write to one of the eight-bit registers, an eight-bit write command must be sent to the FXLC95000CL. The write command consists of an MSB (0 = read, 1 = write) to indicate a write is to be made to the FXLC95000CL register, followed by a 6-bit address and a 1-bit, don't-care bit.

The command should then be followed the 8-bit data transfer. See Figure 5-33 for the timing diagram for a single 8-bit data write.

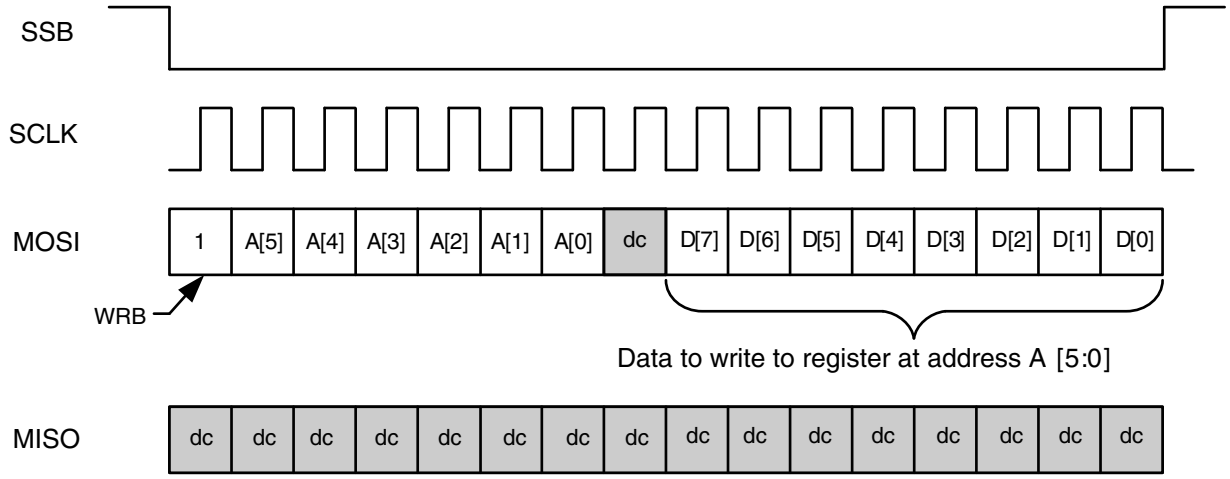


Figure 5-33. 1-byte write by host using SPI

Figure 5-34 illustrates the case where the host is writing 3 sequential bytes of information to the FXLC95000CL.

Note that the FXLC95000CL MISO line is not required for write operations. Data received on MISO by the master device should be ignored.

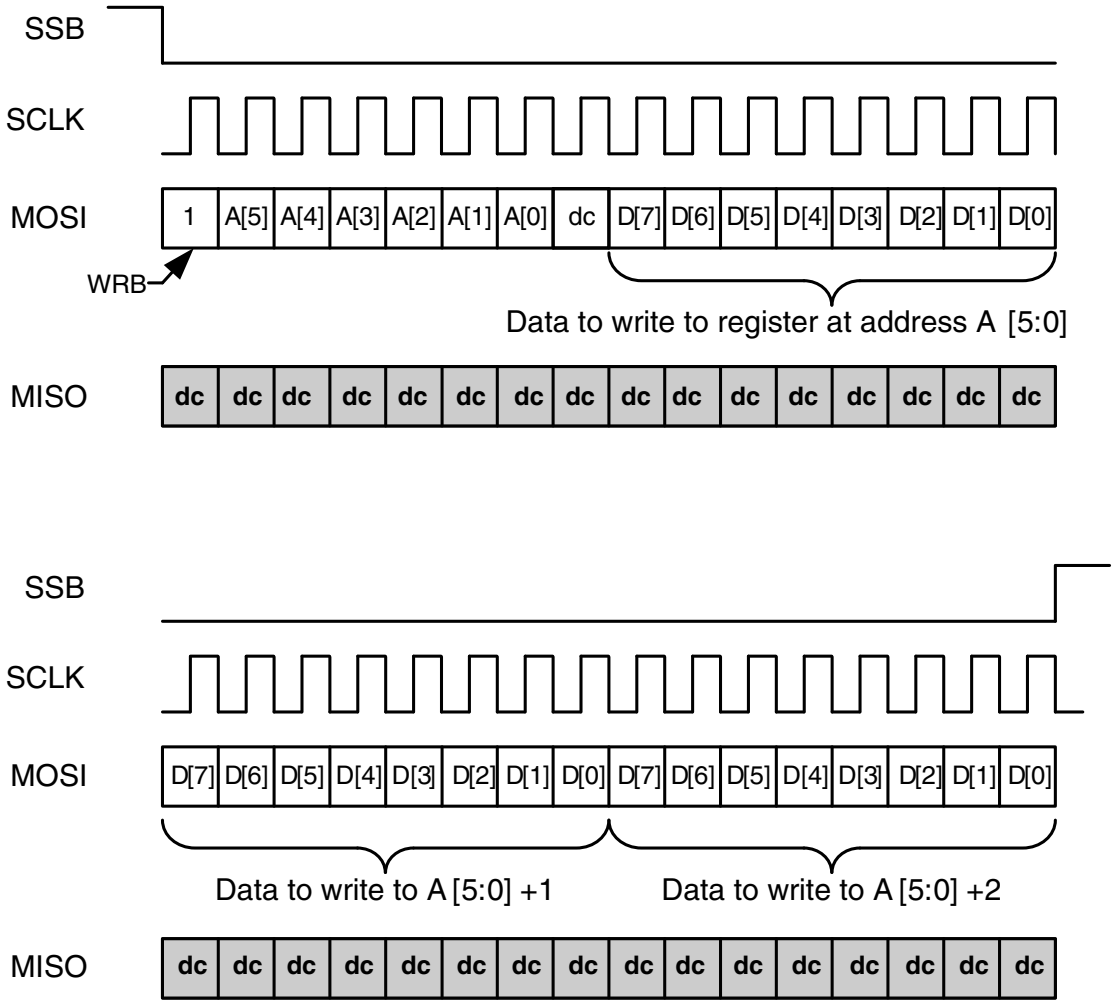


Figure 5-34. 3-byte write by host using SPI

5.7 Slave port module interrupts

The FXLC95000CL can be interrupted by various events from within the slave port module.

The events occurring in the slave port module that can generate interrupts to the CPU are:

- Completion of a read command packet
- Completion of a write command packet
- Access to mailbox 15
- Access to mailbox 31
- Semaphore time-out 1
- Semaphore time-out 2

The slave port can also generate an interrupt (a software-generated interrupt) to the CPU via the SP_OIC register.

5.7.1 Mailbox interrupt

The SP_SCR[WIE] register bit can be used to enable/disable the mailbox write interrupt. An interrupt will always be issued at the end of any write command during RUN mode and during STOP if SP_SCR[STOP_EN]=1.

The SP_SCR[RIE] register bit can be used to enable/disable the mailbox read interrupt. An interrupt will always be issued at the end of any read command during RUN mode and during STOP if SP_SCR[STOP_EN]=1.

SP_SCR[WUP] can be used to add interrupts on accesses to mailboxes 15 and 31. Again, STOP mode operation is controlled via SP_SCR[STOP_EN].

WUP[0] enables interrupt assertion upon a read of Mailbox #15 when SP_SCR[RIE]=1 or when a write of Mailbox #15 occurs when SP_SCR[WIE]=1.

WUP[1] enables interrupt assertion upon a read of Mailbox #31 when SP_SCR[RIE]=1 or when a write of Mailbox #31 occurs when SP_SCR[WIE]=1.

Table 5-25. Slave port interrupt operation

Operation	STOP_EN	WIE	RIE	WUP	Mode	Interrupt asserted?
Any read	–	–	0	–	–	No
Any read	0	–	1	00	RUN	At end of packet
Any read	0	–	1	00	STOP	When STOP exits
Any read	1	–	1	00	STOP	At end of packet
Any write	–	0	–	–	–	No
Any write	0	1	–	00	RUN	At end of packet
Any write	0	1	–	00	STOP	When STOP exits
Any write	1	1	–	00	STOP	At end of packet
Read from mailbox 15	–	–	1	01	RUN	When mailbox 15 is read
Write to mailbox 15	–	1	–	01	RUN	When mailbox 15 is written
Read from mailbox 31	–	–	1	10	RUN	When mailbox 31 is read
Write to mailbox 31	–	1	–	10	RUN	When mailbox 31 is written

5.7.2 Semaphore interrupts

Each of the two semaphores has its own time-out control register and time-out interrupt. The semaphore timers are intended to limit semaphore lockouts to one frame in length or less. Longer periods should be enforced via software in the start of frame interrupt service routine.

5.8 Reset operation

The slave interface is reset when the chip is reset. I^2C and SPI communications are not possible during reset (specifically sim_chip_resetb) assertion.

Chapter 6
Inter-Integrated Circuit (I2C)

6.1 Chip-specific details about Master I²C

Although the Inter-Integrated Circuit (IIC or I²C) module is the same master/slave I²C module found on many Freescale devices, some features may be implemented differently on the FXLC95000CL.

- The FXLC95000CL device already has a dedicated slave interface to communicate with a host processor, as described in Slave Interface. This additional I²C circuit will be used mainly as a master I²C for controlling and communicating with external peripherals (such as sensors). This capability enables the FXLC95000CL to function as an effective and autonomous sensor hub.
- The I²C module may continue to run in $STOP_{FC}$ mode, so long as PCESFC1[MI2C] is set to 1, because the system clock runs at only 62.5 kHz in $STOP_{SC}$ mode. The I²C module should not be used in $STOP_{SC}$ mode.
- Address matching causes wakeup when MCU is in $STOP_{FC}$ mode.
- The I2C is inactive in $STOP_{FC}$ mode for reduced power consumption, except that address matching is enabled in $STOP_{FC}$ mode.

6.1.1 SCL divider corrections

For ICR values of 00h to 0Fh, the following table lists FXLC95000CL-specific SCL divider values as a function of ICR and MULT values.

Table 6-1. FXLC95000CL-specific corrections for SCL divider

ICR field values(hex)	SCL divider values in Table 6-11	MULT=0 Actual SCL divider values	MULT=1 Actual SCL divider values	MULT=2 Actual SCL divider values
0	20	24	20	18
1	22	26	22	20
2	24	28	24	22

Table continues on the next page...

Table 6-1. FXLC95000CL-specific corrections for SCL divider (continued)

ICR field values(hex)	SCL divider values in Table 6-11	MULT=0 Actual SCL divider values	MULT=1 Actual SCL divider values	MULT=2 Actual SCL divider values
3	26	30	26	24
4	28	32	28	26
5	30	34	30	28
6	34	38	34	32
7	40	44	40	38
8	28	32	28	28
9	32	36	32	32
A	36	40	35	36
B	40	44	40	40
C	44	48	44	44
D	48	52	48	48
E	56	60	56	56
F	68	72	68	68

6.2 Introduction

The inter-integrated circuit (I^2C, I2C, or IIC) module provides a method of communication between a number of devices. The interface is designed to operate up to 100 kbit/s with maximum bus loading and timing. The I2C device is capable of operating at higher baud rates, up to a maximum of clock/20, with reduced bus loading. The maximum communication length and the number of devices that can be connected are limited by a maximum bus capacitance of 400 pF.

6.2.1 Features

The I2C module has the following features:

- Compatible with *The I^2C-Bus Specification*
- Multimaster 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 and detection
- Repeated START signal generation and detection

- Acknowledge bit generation and detection
- Bus busy detection
- General call recognition
- 10-bit address extension
- Programmable glitch input filter
- Low power mode wakeup on slave address match

6.2.2 Modes of operation

The I2C module's operation in various low power modes is as follows:

- 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 when the core is in Wait mode and can provide a wakeup interrupt.
- Stop mode: The module is inactive in Stop mode for reduced power consumption, except that address matching is enabled in Stop mode. The STOP instruction does not affect the I2C module's register states.

6.2.3 Block diagram

The following figure is a functional block diagram of the I2C module.

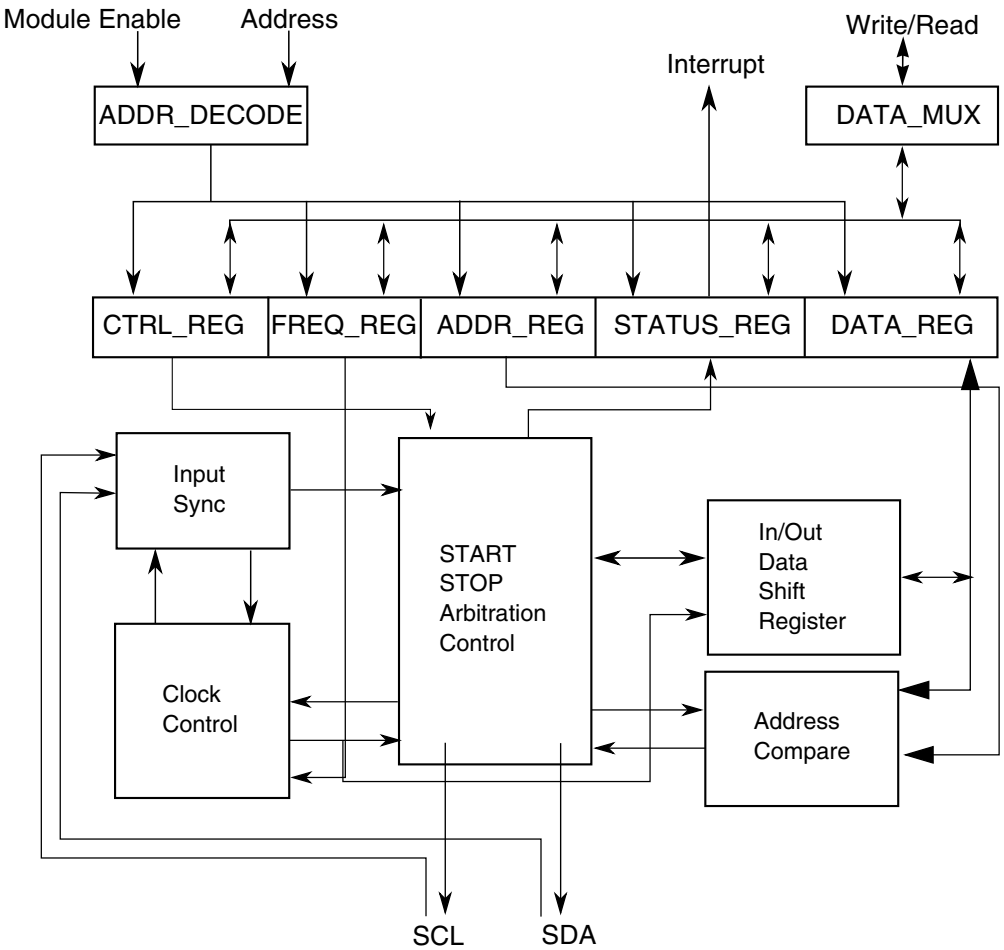


Figure 6-1. I2C Functional block diagram

6.3 I²C signal descriptions

The signal properties of I²C are shown in the following table.

Table 6-2. I²C signal descriptions

Signal	Description	I/O
SCL	Bidirectional serial clock line of the I²C system.	I/O
SDA	Bidirectional serial data line of the I²C system.	I/O

6.4 Memory map and register descriptions

This section describes in detail all I2C registers accessible to the end user.

I2C memory map

Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
FFFF_8040	I2C Address Register 1 (I2C_A1)	8	R/W	00h	6.4.1/105
FFFF_8041	I2C Frequency Divider register (I2C_F)	8	R/W	00h	6.4.2/105
FFFF_8042	I2C Control Register 1 (I2C_C1)	8	R/W	00h	6.4.3/106
FFFF_8043	I2C Status register (I2C_S)	8	R/W	80h	6.4.4/108
FFFF_8044	I2C Data I/O register (I2C_D)	8	R/W	00h	6.4.5/109
FFFF_8045	I2C Control Register 2 (I2C_C2)	8	R/W	00h	6.4.6/110
FFFF_8046	I2C Programmable Input Glitch Filter register (I2C_FLT)	8	R/W	00h	6.4.7/110

6.4.1 I2C Address Register 1 (I2C_A1)

This register contains the slave address to be used by the I2C module.

Address: FFFF_8040h base + 0h offset = FFFF_8040h

Bit	7	6	5	4	3	2	1	0
Read	AD[7:1]							0
Write								
Reset	0	0	0	0	0	0	0	0

I2C_A1 field descriptions

Field	Description
7–1 AD[7:1]	Address Contains the primary slave address used by the I2C module when it is addressed as a slave. This field is used in the 7-bit address scheme and the lower seven bits in the 10-bit address scheme.
0 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.

6.4.2 I2C Frequency Divider register (I2C_F)

Address: FFFF_8040h base + 1h offset = FFFF_8041h

Bit	7	6	5	4	3	2	1	0
Read	MULT		ICR					
Write								
Reset	0	0	0	0	0	0	0	0

I2C_F field descriptions

Field	Description
7–6 MULT	The MULT bits define the multiplier factor mul. This factor is used along with the SCL divider to generate the I2C baud rate.

Table continues on the next page...

I2C_F field descriptions (continued)

Field	Description
	00 mul = 1 01 mul = 2 10 mul = 4 11 Reserved

ICR	ClockRate
	
	Prescales the bus clock for bit rate selection. This field and the MULT field determine the I2C baud rate, the SDA hold time, the SCL start hold time, and the SCL stop hold time. For a list of values corresponding to each ICR setting, see I2C divider and hold values.
	
	The SCL divider multiplied by multiplier factor (mul) determines the I2C baud rate.
	
	`I2C baud rate = bus speed (Hz)/(mul × SCL divider)`
	
	The SDA hold time is the delay from the falling edge of SCL (I2C clock) to the changing of SDA (I2C data).
	
	`SDA hold time = bus period (s) × mul × SDA hold value`
	
	The SCL start hold time is the delay from the falling edge of SDA (I2C data) while SCL is high (start condition) to the falling edge of SCL (I2C clock).
	
	`SCL start hold time = bus period (s) × mul × SCL start hold value`
	
	The SCL stop hold time is the delay from the rising edge of SCL (I2C clock) to the rising edge of SDA (I2C data) while SCL is high (stop condition).
	
	`SCL stop hold time = bus period (s) × mul × SCL stop hold value`
	
	For example, if the bus speed is 8 MHz, the following table shows the possible hold time values with different ICR and MULT selections to achieve an I^2C baud rate of 100 kbit/s.

MULT	ICR	Hold times (µs)		
		SDA	SCL Start	SCL Stop
2h	00h	3.500	3.000	5.500
1h	07h	2.500	4.000	5.250
1h	0Bh	2.250	4.000	5.250
0h	14h	2.125	4.250	5.125
0h	18h	1.125	4.750	5.125

6.4.3 I2C Control Register 1 (I2C_C1)

Address: FFFF_8040h base + 2h offset = FFFF_8042h

Bit	7	6	5	4	3	2	1	0
Read	IICEN	IICIE	MST	TX	TXAK	0	WUEN	0
Write						RSTA		
Reset	0	0	0	0	0	0	0	0

I2C_C1 field descriptions

Field	Description
7 IICEN	I2C Enable Enables I2C module operation. 0 Disabled 1 Enabled
6 IICIE	I2C Interrupt Enable Enables I2C interrupt requests. 0 Disabled 1 Enabled
5 MST	Master Mode Select When the MST bit is changed from a 0 to a 1, a START signal is generated on the bus and master mode is selected. When this bit changes from a 1 to a 0, a STOP signal is generated and the mode of operation changes from master to slave. 0 Slave mode 1 Master mode
4 TX	Transmit Mode Select Selects the direction of master and slave transfers. In master mode this bit must be set according to the type of transfer required. Therefore, for address cycles, this bit is always set. When addressed as a slave this bit must be set by software according to the SRW bit in the status register. 0 Receive 1 Transmit
3 TXAK	Transmit Acknowledge Enable Specifies the value driven onto the SDA during data acknowledge cycles for both master and slave receivers. **NOTE:** SCL is held low until TXAK is written. 0 An acknowledge signal is sent to the bus on the following receiving byte 1 No acknowledge signal is sent to the bus on the following receiving data byte.
2 RSTA	Repeat START Writing a one to this bit generates a repeated START condition provided it is the current master. This bit will always be read as zero. Attempting a repeat at the wrong time results in loss of arbitration.
1 WUEN	Wakeup Enable The I2C module can wake the MCU from low power mode with no peripheral bus running when slave address matching occurs. 0 Normal operation. No interrupt generated when address matching in low power mode. 1 Enables the wakeup function in low power mode.
0 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.

6.4.4 I2C Status register (I2C_S)

Address: FFFF_8040h base + 3h offset = FFFF_8043h

Bit	7	6	5	4	3	2	1	0
Read	TCF	IAAS	BUSY	ARBL	0	SRW	IICIF	RXAK
Write				w1c			w1c	
Reset	1	0	0	0	0	0	0	0

I2C_S field descriptions

Field	Description
7 TCF	Transfer Complete Flag This bit sets on the completion of a byte and acknowledge bit transfer. This bit is valid only during or immediately following a transfer to or from the I2C module. The TCF bit is cleared by reading the I2C data register in receive mode or by writing to the I2C data register in transmit mode. 0 Transfer in progress 1 Transfer complete
6 IAAS	Addressed As A Slave This bit is set by one of the following conditions: • The calling address matches the programmed slave primary address in the A1 register. • GCAEN is set and a general call is received. This bit sets before the ACK bit. The CPU must check the SRW bit and set TX/RX accordingly. Writing the C1 register with any value clears this bit. 0 Not addressed 1 Addressed as a slave
5 BUSY	Bus Busy Indicates the status of the bus regardless of slave or master mode. This 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 one to it. 0 Standard bus operation. 1 Loss of arbitration.
3 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
2 SRW	Slave Read/Write When addressed as a slave, SRW indicates the value of the R/$\overline{W}$ command bit of the calling address sent to the master.

Table continues on the next page...

I2C_S field descriptions (continued)

Field	Description
	0 Slave receive, master writing to slave 1 Slave transmit, master reading from slave
1 IICIF	Interrupt Flag This bit sets when an interrupt is pending. This bit must be cleared by software by writing a 1 to it, such as in the interrupt routine. One of the following events can set this bit: • One byte transfer, including ACK/NACK bit, completes. An ACK or NACK is sent on the bus by writing 0 or 1 to TXAK after this bit is set in receive mode. • Match of slave address to calling address including primary slave address or general call address. • Arbitration lost 0 No interrupt pending 1 Interrupt pending
0 RXAK	Receive Acknowledge 0 Acknowledge signal was received after the completion of one byte of data transmission on the bus 1 No acknowledge signal detected

6.4.5 I2C Data I/O register (I2C_D)

Address: FFFF_8040h base + 4h offset = FFFF_8044h

Bit	7	6	5	4	3	2	1	0
Read Write	DATA							
Reset	0	0	0	0	0	0	0	0

I2C_D field descriptions

Field	Description
DATA	Data In master transmit mode, when data is written to this register, a data transfer is initiated. The most significant bit is sent first. In master receive mode, reading this register initiates receiving of the next byte of data. **NOTE:** When making the transition out of master receive mode, switch the I2C mode before reading the Data register to prevent an inadvertent initiation of a master receive data transfer. In slave mode, the same functions are available after an address match occurs. The C1[TX] bit must correctly reflect the desired direction of transfer in master and slave modes for the transmission to begin. For example, if the I2C module is configured for master transmit but a master receive is desired, reading the Data register does not initiate the receive. Reading the Data register returns the last byte received while the I2C module is configured in master receive or slave receive mode. The Data register does not reflect every byte that is transmitted on the I2C bus, and neither can software verify that a byte has been written to the Data register correctly by reading it back. In master transmit mode, the first byte of data written to the Data register following assertion of MST (start bit) or assertion of RSTA (repeated start bit) is used for the address transfer and must consist of the calling address (in bits 7-1) concatenated with the required R/$\overline{W}$ bit (in position bit 0).

6.4.6 I2C Control Register 2 (I2C_C2)

Address: FFFF_8040h base + 5h offset = FFFF_8045h

Bit	7	6	5	4	3	2	1	0
Read	GCAEN	ADEXT	0	0	0	AD[10:8]		
Write								
Reset	0	0	0	0	0	0	0	0

I2C_C2 field descriptions

Field	Description
7 GCAEN	General Call Address Enable Enables general call address. 0 Disabled 1 Enabled
6 ADEXT	Address Extension Controls the number of bits used for the slave address. 0 7-bit address scheme 1 10-bit address scheme
5 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
4 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
3 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
AD[10:8]	Slave Address Contains the upper three bits of the slave address in the 10-bit address scheme. This field is valid only while the ADEXT bit is set.

6.4.7 I2C Programmable Input Glitch Filter register (I2C_FLT)

Address: FFFF_8040h base + 6h offset = FFFF_8046h

Bit	7	6	5	4	3	2	1	0
Read	Reserved	0			FLT			
Write								
Reset	0	0	0	0	0	0	0	0

I2C_FLT field descriptions

Field	Description
7 Reserved	This field is reserved. Writing this bit has no effect.

Table continues on the next page...

I2C_FLT field descriptions (continued)

Field	Description
6–5 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
FLT	I2C Programmable Filter Factor Controls the width of the glitch, in terms of bus clock cycles, that the filter must absorb. For any glitch whose size is less than or equal to this width setting, the filter does not allow the glitch to pass. 00h No filter/bypass 01-1Fh Filter glitches up to width of *n* bus clock cycles, where *n*=1-31d

6.5 Functional description

This section provides a comprehensive functional description of the I2C module.

6.5.1 I2C protocol

The I2C bus system uses a serial data line (SDA) and a serial clock line (SCL) for data transfers. 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 depends on the system.

Normally, a standard instance of communication is composed of four parts:

1. START signal
2. Slave address transmission
3. Data transfer
4. STOP signal

The STOP signal should not be confused with the CPU STOP instruction. The following figure illustrates I2C bus system communication.

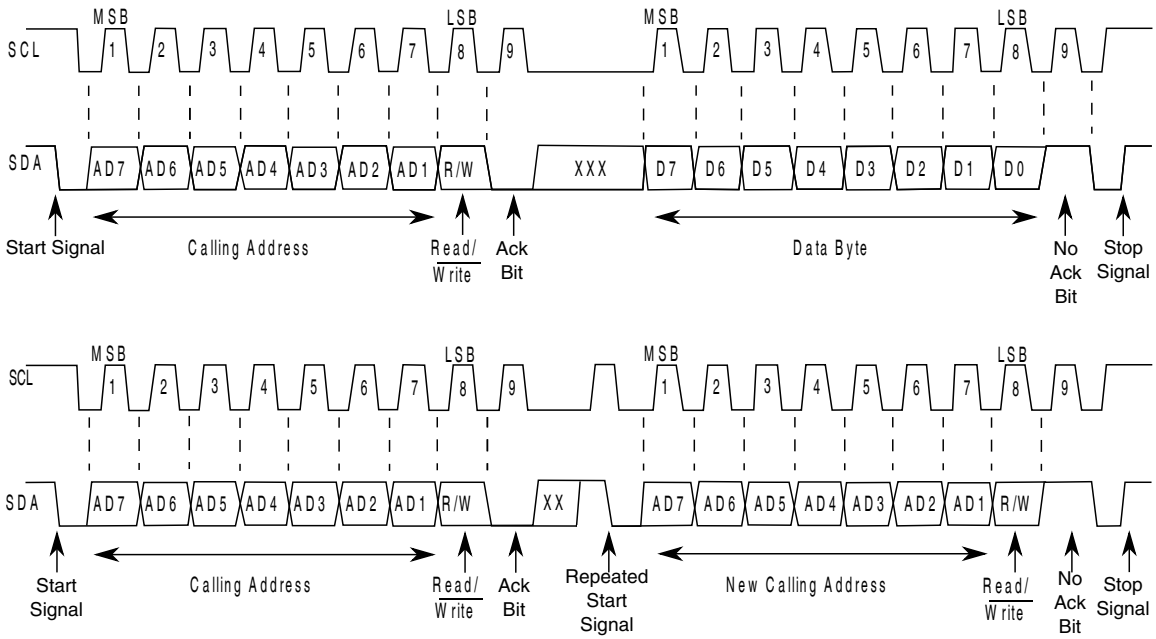


Figure 6-9. I2C bus transmission signals

6.5.1.1 START signal

The bus is free when no master device is engaging the bus (both SCL and SDA are high). When the bus is free, a master may initiate communication by sending a START signal. 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 might contain several bytes of data—and brings all slaves out of their idle states.

6.5.1.2 Slave address transmission

Immediately after the START signal, the first byte of a data transfer is the slave address transmitted by the master. This address is a 7-bit calling address followed by an R/$\overline{W}$ bit. The R/$\overline{W}$ 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 an acknowledge bit. The slave sends the acknowledge bit by pulling SDA low at the ninth clock.

No two slaves in the system can have the same address. If the I2C module is the master, it must not transmit an address that is equal to its own slave address. The I2C module cannot be master and slave at the same time. However, if arbitration is lost during an address cycle, the I2C module reverts to slave mode and operates correctly even if it is being addressed by another master.

6.5.1.3 Data transfers

When successful slave addressing is achieved, data transfer can proceed on a byte-by-byte basis in the direction specified by the R/$\overline{W}$ bit sent by the calling master.

All transfers that follow an address cycle are referred to as data transfers, even if they carry subaddress information for the slave device.

Each data byte is 8 bits long. Data may be changed only while SCL is low. Data must be held stable while SCL is high. There is one clock pulse on SCL for each data bit, and the MSB is transferred first. Each data byte is followed by a ninth (acknowledge) bit, which is signaled from the receiving device by pulling 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, the slave must leave SDA high. The master interprets the failed acknowledgement as an unsuccessful data transfer.

If the master receiver does not acknowledge the slave transmitter after a data byte transmission, the slave interprets it as an end to data transfer and releases the SDA line.

In the case of a failed acknowledgement by either the slave or master, the data transfer is aborted and the master does one of two things:

- Relinquishes the bus by generating a STOP signal.
- Commences a new call by generating a repeated START signal.

6.5.1.4 STOP signal

The master can terminate the communication by generating a STOP signal to free the bus. A STOP signal is defined as a low-to-high transition of SDA while SCL is asserted.

The master can generate a STOP signal even if the slave has generated an acknowledgement, at which point the slave must release the bus.

6.5.1.5 Repeated START signal

The master may generate a START signal followed by a calling command without generating a STOP signal first. This action is called a repeated START. The master uses a repeated START to communicate with another slave or with the same slave in a different mode (transmit/receive mode) without releasing the bus.

6.5.1.6 Arbitration procedure

The I2C bus is a true multimaster 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. The bus clock's low period is equal to the longest clock low period, and the high period 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 level 1 while another master transmits logic level 0. The losing masters immediately switch 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, hardware sets a status bit to indicate the loss of arbitration.

6.5.1.7 Clock synchronization

Because wire AND logic is performed on SCL, a high-to-low transition on SCL affects all devices connected on the bus. The devices start counting their low period and, after a device's clock has gone low, that device holds SCL low until the clock reaches its high state. However, the change of low to high in this device clock might not change the state of SCL if another device clock is still within its low period. Therefore, the 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 the following diagram. When all applicable devices have counted off their low period, the synchronized clock SCL is released and pulled high. Afterward there is no difference between the device clocks and the state of SCL, and all devices start counting their high periods. The first device to complete its high period pulls SCL low again.

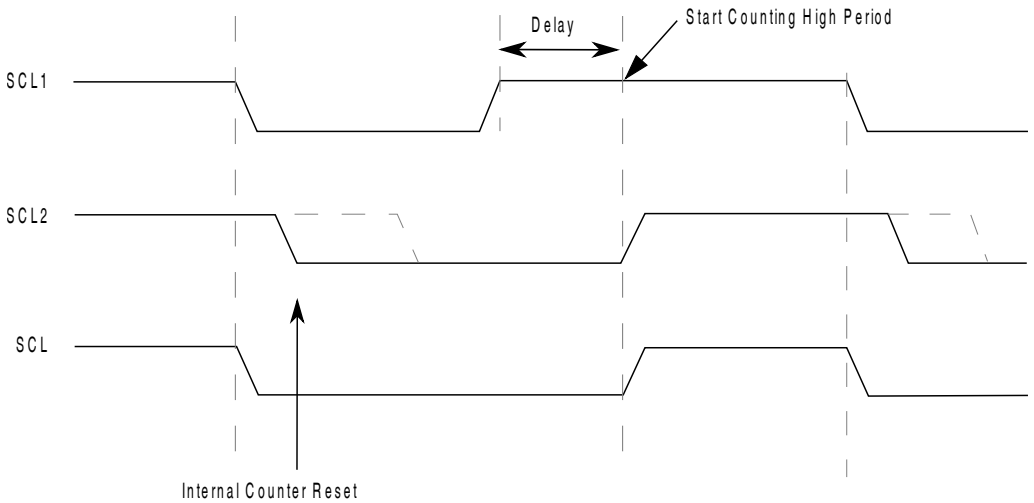


Figure 6-10. I2C clock synchronization

6.5.1.8 Handshaking

The clock synchronization mechanism can be used as a handshake in data transfers. A slave device may hold SCL low after completing a single byte transfer (9 bits). In this case, it halts the bus clock and forces the master clock into wait states until the slave releases SCL.

6.5.1.9 Clock stretching

The clock synchronization mechanism can be used by slaves to slow down the bit rate of a transfer. After the master drives SCL low, a slave can drive SCL low for the required period and then release it. If the slave's SCL low period is greater than the master's SCL low period, the resulting SCL bus signal's low period is stretched. In other words, the SCL bus signal's low period is increased to be the same length as the slave's SCL low period.

6.5.1.10 I2C divider and hold values

NOTE

For some cases on some devices, the SCL divider value may vary by +/-2 or +/-4 when ICR’s value ranges from 00h to 0Fh. These potentially varying SCL divider values are highlighted in the following table. For the actual SCL divider values for your device, see the chip-specific details about the I2C module.

Table 6-11. I2C divider and hold values

ICR (hex)	SCL divider	SDA hold value	SCL hold (start) value	SCL hold (stop) value	ICR (hex)	SCL divider (clocks)	SDA hold (clocks)	SCL hold (start) value	SCL hold (stop) value
00	20	7	6	11	20	160	17	78	81
01	22	7	7	12	21	192	17	94	97
02	24	8	8	13	22	224	33	110	113
03	26	8	9	14	23	256	33	126	129
04	28	9	10	15	24	288	49	142	145
05	30	9	11	16	25	320	49	158	161
06	34	10	13	18	26	384	65	190	193
07	40	10	16	21	27	480	65	238	241
08	28	7	10	15	28	320	33	158	161
09	32	7	12	17	29	384	33	190	193
0A	36	9	14	19	2A	448	65	222	225
0B	40	9	16	21	2B	512	65	254	257
0C	44	11	18	23	2C	576	97	286	289
0D	48	11	20	25	2D	640	97	318	321
0E	56	13	24	29	2E	768	129	382	385
0F	68	13	30	35	2F	960	129	478	481
10	48	9	18	25	30	640	65	318	321
11	56	9	22	29	31	768	65	382	385
12	64	13	26	33	32	896	129	446	449
13	72	13	30	37	33	1024	129	510	513
14	80	17	34	41	34	1152	193	574	577
15	88	17	38	45	35	1280	193	638	641
16	104	21	46	53	36	1536	257	766	769
17	128	21	58	65	37	1920	257	958	961
18	80	9	38	41	38	1280	129	638	641
19	96	9	46	49	39	1536	129	766	769
1A	112	17	54	57	3A	1792	257	894	897
1B	128	17	62	65	3B	2048	257	1022	1025
1C	144	25	70	73	3C	2304	385	1150	1153
1D	160	25	78	81	3D	2560	385	1278	1281
1E	192	33	94	97	3E	3072	513	1534	1537
1F	240	33	118	121	3F	3840	513	1918	1921

6.5.2 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.

6.5.2.1 Master-transmitter addresses a slave-receiver

The transfer direction is not changed. When a 10-bit address follows a START condition, each slave compares the first 7 bits of the first byte of the slave address (11110XX) with its own address and tests whether the eighth bit (R/$\overline{W}$ direction bit) is 0. It is possible that more than one device finds a match and generates an acknowledge (A1). Each slave that finds a match compares the 8 bits of the second byte of the slave address with its own address, but 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.

Table 6-12. Master-transmitter addresses slave-receiver with a 10-bit address

S	Slave address first 7 bits 11110 + AD10 + AD9	R/$\overline{W}$ 0	A1	Slave address second byte AD[8:1]	A2	Data	A	...	Data	A/A	P

After the master-transmitter has sent the first byte of the 10-bit address, the slave-receiver sees an I2C interrupt. User software must ensure that for this interrupt, the contents of the Data register are ignored and not treated as valid data.

6.5.2.2 Master-receiver addresses a slave-transmitter

The transfer direction is changed after the second R/$\overline{W}$ bit. 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 it tests whether the eighth (R/$\overline{W}$) 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/$\overline{W}$) bit. However, none of them are addressed because R/$\overline{W}$ = 1 (for 10-bit devices), or the 11110XX slave address (for 7-bit devices) does not match.

Table 6-13. Master-receiver addresses a slave-transmitter with a 10-bit address

S	Slave address first 7 bits 11110 + AD10 + AD9	R/$\overline{W}$ 0	A1	Slave address second byte AD[8:1]	A2	Sr	Slave address first 7 bits 11110 + AD10 + AD9	R/$\overline{W}$ 1	A3	Data	A	...	Data	A	P

After the master-receiver has sent the first byte of the 10-bit address, the slave-transmitter sees an I2C interrupt. User software must ensure that for this interrupt, the contents of the Data register are ignored and not treated as valid data.

6.5.3 Address matching

All received addresses can be requested in 7-bit or 10-bit address format.

- AD[7:1] in Address Register 1, which contains the I2C primary slave address, always participates in the address matching process. It provides a 7-bit address.
- If the ADEXT bit is set, AD[10:8] in Control Register 2 participates in the address matching process. It extends the I2C primary slave address to a 10-bit address.

Additional conditions that affect address matching include:

- If the GCAEN bit is set, general call participates the address matching process.

When the I2C module responds to one of these addresses, it acts as a slave-receiver and the IAAS bit is set after the address cycle. Software must read the Data register after the first byte transfer to determine that the address is matched.

6.5.4 Resets

The I2C module is disabled after a reset. The I2C module cannot cause a core reset.

6.5.5 Interrupts

The I2C module generates an interrupt when any of the events in the following table occur, provided that the IICIE bit is set. The interrupt is driven by the IICIF bit (of the I2C Status Register) and masked with the IICIE bit (of the I2C Control Register 1). The IICIF bit must be cleared (by software) by writing 1 to it in the interrupt routine. You can determine the interrupt type by reading the Status Register.

Table 6-14. 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
Wakeup from stop or wait mode	IAAS	IICIF	IICIE & WUEN

6.5.5.1 Byte transfer interrupt

The Transfer Complete Flag (TCF) bit is set at the falling edge of the ninth clock to indicate the completion of a byte and acknowledgement transfer.

6.5.5.2 Address detect interrupt

When the calling address matches the programmed slave address (I2C 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 bit is set. The CPU must check the SRW bit and set its Tx mode accordingly.

6.5.5.3 Exit from low-power/stop modes

The slave receive input detect circuit and address matching feature are still active on low power modes (wait and stop). An asynchronous input matching slave address or general call address brings the CPU out of low power/stop mode if the interrupt is not masked. Therefore, TCF and IAAS both can trigger this interrupt.

6.5.5.4 Arbitration lost interrupt

The I2C is a true multimaster 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 I2C module asserts the arbitration-lost 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:

1. SDA is sampled as low when the master drives high during an address or data transmit cycle.
2. SDA is sampled as low when the master drives high during the acknowledge bit of a data receive cycle.
3. A START cycle is attempted when the bus is busy.
4. A repeated START cycle is requested in slave mode.
5. A STOP condition is detected when the master did not request it.

The ARBL bit must be cleared (by software) by writing 1 to it.

6.5.6 Programmable input glitch filter

An I2C glitch filter has been added outside legacy I2C modules but within the I2C package. This filter can absorb glitches on the I2C clock and data lines for the I2C module. The width of the glitch to absorb can be specified in terms of the number of (half) bus clock cycles. A single Programmable Input Glitch Filter control register is provided. Effectively, any down-up-down or up-down-up transition on the data line that occurs within the number of clock cycles programmed in this register is ignored by the I2C module. The programmer must specify the size of the glitch (in terms of bus clock cycles) for the filter to absorb and not pass.

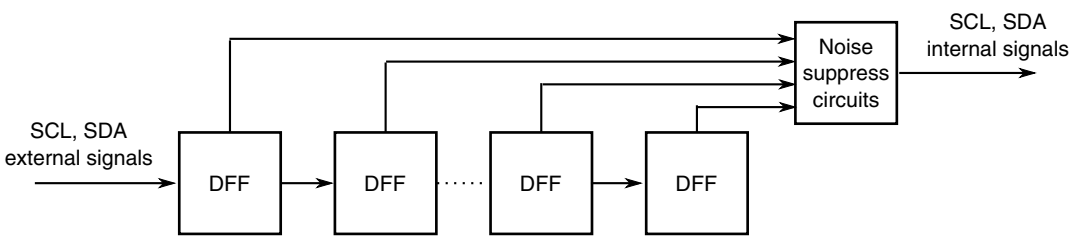


Figure 6-11. Programmable input glitch filter diagram

6.5.7 Address matching wakeup

When a primary or general call address match occurs when the I2C module is in slave receive mode, the MCU wakes from a low power mode where no peripheral bus is running.

After the address matching IAAS bit is set, an interrupt is sent at the end of address matching to wake the core.

NOTE

During the wakeup process, if an external master continues to send data to the slave, the baud rate under Stop mode must be less than 50 kbps. To avoid the slower baud rate under Stop mode, the master can add a short delay in firmware to wait until the wakeup process is complete and then send data.

6.6 Initialization/application information

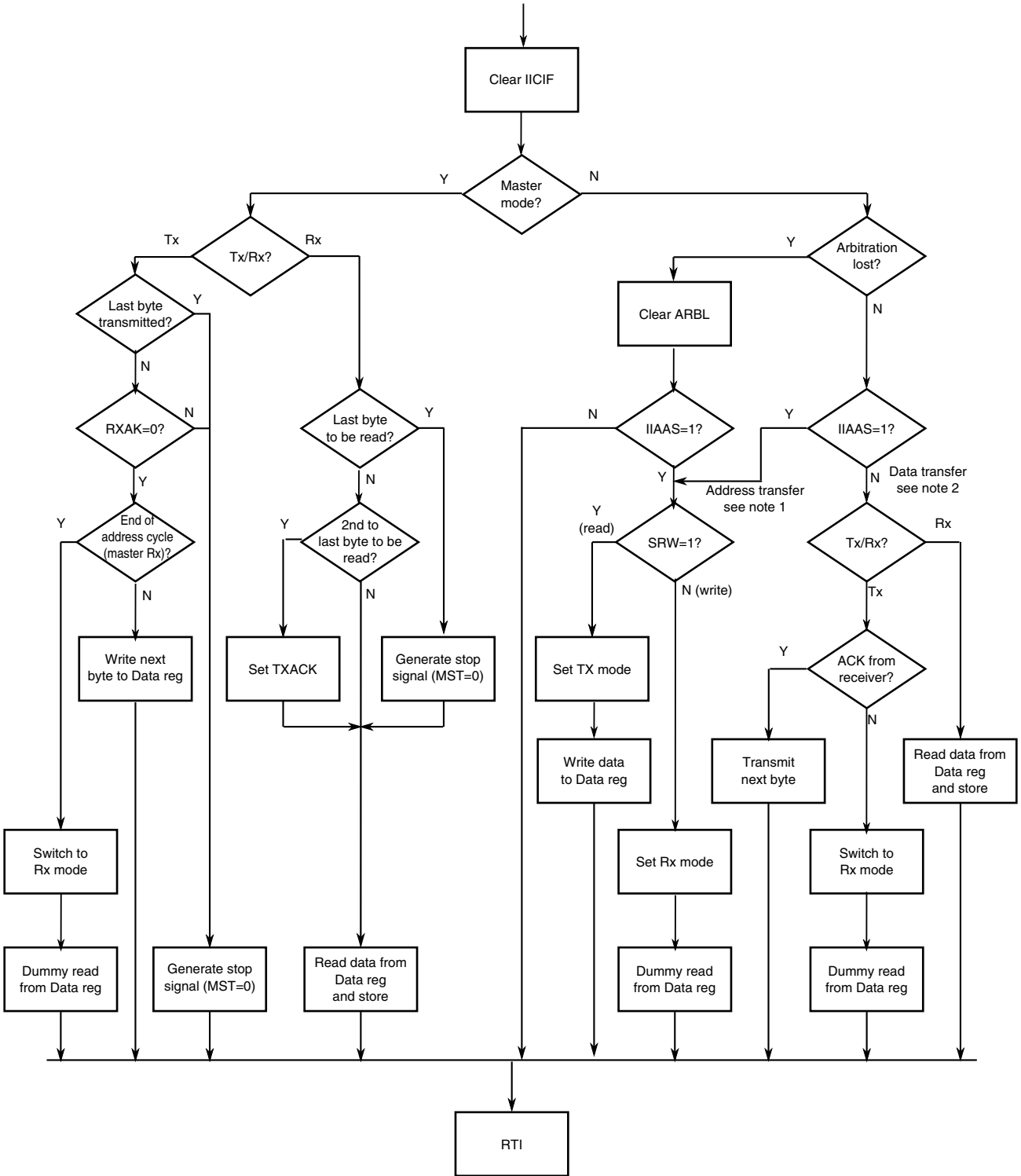
Module Initialization (Slave)

1. Write: Control Register 2
   - to enable or disable general call
   - to select 10-bit or 7-bit addressing mode
2. Write: Address Register 1 to set the slave address
3. Write: Control Register 1 to enable the I2C module 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 the following figure

Module Initialization (Master)

1. Write: Frequency Divider register to set the I2C baud rate (see example in description of ICR)
2. Write: Control Register 1 to enable the I2C module 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 the following figure
5. Write: Control Register 1 to enable TX
6. Write: Control Register 1 to enable MST (master mode)
7. Write: Data register with the address of the target slave (the LSB of this byte determines whether the communication is master receive or transmit)

The routine shown in the following figure encompasses both master and slave I2C operations. For slave operation, an incoming I2C message that contains the proper address begins I2C communication. For master operation, communication must be initiated by writing the Data register.



Notes:

1. If general call is enabled, check to determine if the received address is a general call address (0x00). If the received address is a general call address, the general call must be handled by user software.
2. When 10-bit addressing addresses a slave, the slave sees an interrupt following the first byte of the extended address. Ensure that for this interrupt, the contents of the Data register are ignored and not treated as a valid data transfer.

Figure 6-12. Typical I2C interrupt routine

Chapter 7
Queued Serial Peripheral Interface (QSPI)

7.1 Chip-specific information about QSPI

7.1.1 Maximum master mode frequency

The QSPI module is designed to operate at a maximum master mode frequency of bus frequency ÷ 2. However, on FXLC95000CL, the maximum QSPI master mode frequency is bus frequency ÷ 4, or 4 MHz. See the FXLC95000CL data sheet for more information.

7.1.2 Signal names

The following table correlates the QSPI signal names on FXLC95000CL with the module's generic signal names. See External I/O Signals for detailed information about the signals.

Table 7-1. QSPI signal names on FXLC95000CL

FXLC95000CL signal name	QSPI signal name
MOSI1	MOSI
MISO1	MISO
SCLK1	SCLK
SSB1	$\overline{SS}$

7.2 Introduction

7.2.1 Overview

The serial peripheral interface (SPI) module enables full-duplex, synchronous, serial communication between the chip and peripheral devices, including other chips. Software can poll the SPI status flags or SPI operation can be interrupt driven. The block contains six 16-bit memory mapped registers for control parameters, status, and data transfer.

Features of the SPI module include the following:

- Full-duplex operation
- Master and slave modes
- Double-buffered operation with separate transmit and receive registers
- Programmable Length Transactions (2 to 16 bits)
- Programmable transmit and receive shift order (MSB or LSB first)
- Fourteen master mode frequencies (maximum = bus frequency ÷ 2)
- Maximum slave mode frequency = bus frequency ÷ 4
- Serial clock with programmable polarity and phase
- Two separately enabled interrupts:
  - SPRF (SPI receiver full)
  - SPTE (SPI transmitter empty)
- Mode fault error flag with interrupt capability
- Overflow error flag with interrupt capability
- Wired OR mode functionality enabling connection to multiple SPIs
- Separate RX and TX FIFO capable of handling 4 transactions

Maximum SPI data rates are limited by I/O pad performance as specified in the device's data sheet.

7.2.2 Block Diagram

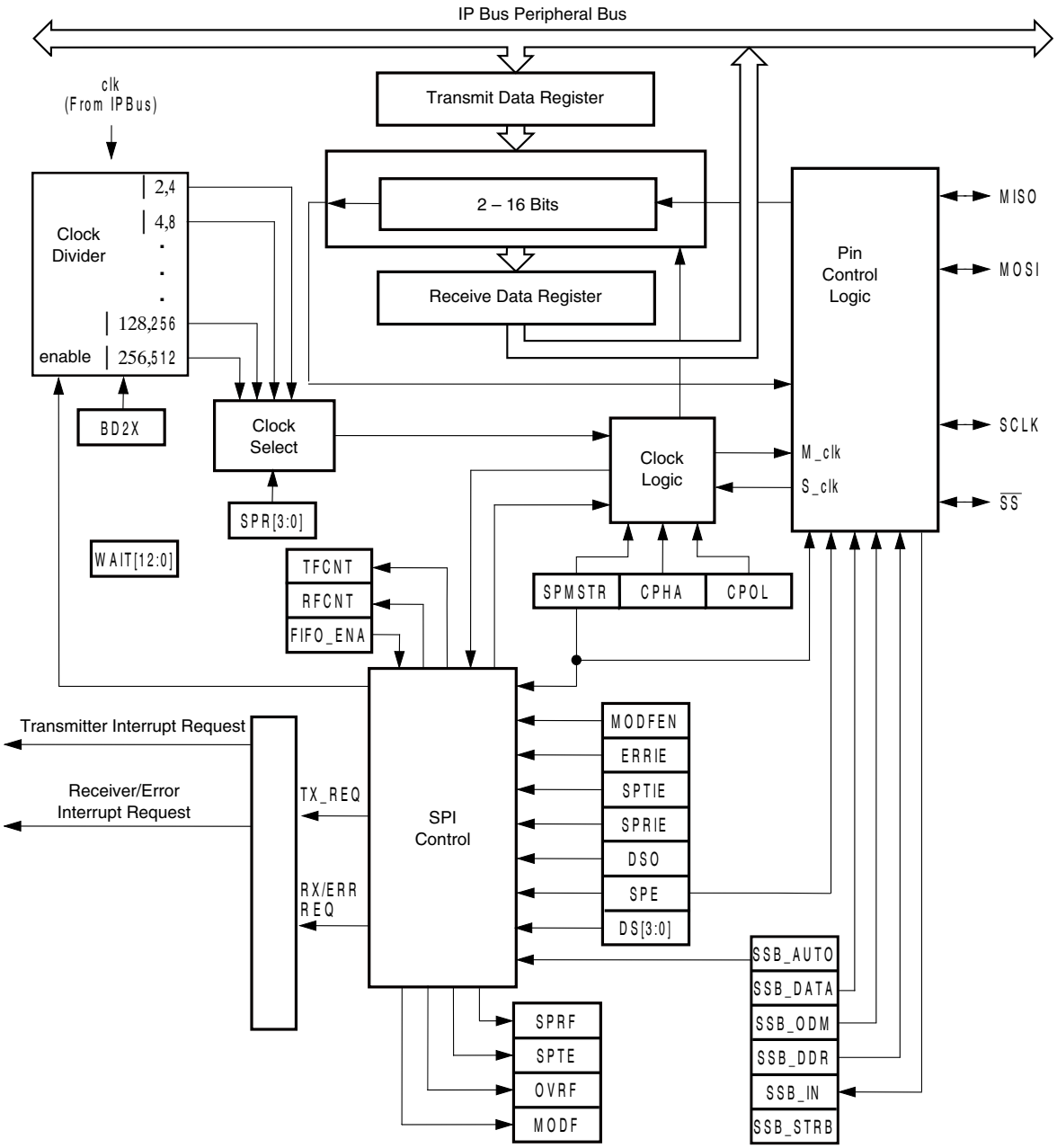


Figure 7-1. SPI Block Diagram

7.3 Signal Descriptions

7.3.1 External I/O Signals

The following are external I/O signals at the device interface. All of these pins are bidirectional.

Table 7-2. External I/O

Signal Name	Description	Direction	
		Master	Slave
MOSI	Master-out Slave-in Pad Pin	Output	Input
MISO	Master-in Slave-out Pad Pin	Input	Output
SCLK	Slave Clock Pad Pin	Output	Input
$\overline{SS}$	Slave Select Pad Pin (Active Low)	See Table 7-3.	

7.3.1.1 MISO (Master In/Slave Out)

MISO is one of the two SPI module pins that transmit serial data. In full duplex operation, the MISO pin of the master SPI module is connected to the MISO pin of the slave SPI module. The master SPI simultaneously receives data on its MISO pin and transmits data from its MOSI pin.

Slave output data on the MISO pin is enabled only when the SPI is configured as a slave. The SPI is configured as a slave when its SPMSTR bit (see the description of the SPI status and control register) is logic zero and its SS pin is at logic zero. To support a multiple-slave system, a logic one on the $\overline{SS}$ pin puts the MISO pin in a high-impedance state.

7.3.1.2 MOSI (Master Out/Slave In)

MOSI is one of the two SPI module pins that transmits serial data. In full-duplex operation, the MOSI pin of the master SPI module is connected to the MOSI pin of the slave SPI module. The master SPI simultaneously transmits data from its MOSI pin and receives data on its MISO pin.

7.3.1.3 SCLK (Serial Clock)

The serial clock synchronizes data transactions between master and slave devices. In a master device, the SCLK pin is the clock output. In a slave device, the SCLK pin is the clock input. In full duplex operation, the master and slave devices exchange data in the same number of clock cycles as the number of bits of transmitted data.

7.3.1.4 $\overline{SS}$ (Slave Select)

The $\overline{SS}$ pin has various functions depending on the current state of the SPI. For an SPI configured as a slave, the $\overline{SS}$ is used to select a slave. For CPHA = 0, the $\overline{SS}$ is used to define the start of a transaction. Because it is used to indicate the start of a transaction, the $\overline{SS}$ must be toggled high and low between each full length set of data transmitted for the CPHA = 0 format. However, it can remain low between transactions for the CPHA = 1 format.

When an SPI is configured as a slave, the $\overline{SS}$ pin is always configured as an input. The MODFEN bit can prevent the state of the $\overline{SS}$ from creating a MODF error.

NOTE

A logic one voltage on the $\overline{SS}$ pin of a slave SPI puts the MISO pin in a high-impedance state. The slave SPI ignores all incoming SCLK clocks, even if it is already in the middle of a transaction. A mode fault occurs if the $\overline{SS}$ pin changes state during a transaction.

When an SPI is configured as a master, the $\overline{SS}$ input can be used in conjunction with the MODF flag to prevent multiple masters from driving MOSI and SCLK. For the state of the $\overline{SS}$ pin to set the MODF flag, the MODFEN bit in the SPI Status and Control register must be set.

Table 7-3. SPI IO Configuration

SPE	SPMSTR	MODFEN	SPI CONFIGURATION	STATE OF SS_B LOGIC
0	X[1]	X	Not Enabled	$\overline{SS}$ ignored by SPI
1	0	X	Slave	Input-only to SPI
1	1	0	Master without MODF	$\overline{SS}$ input ignored by SPI, $\overline{SS}$ output may be activated under software or hardware control to select slave devices.
1	1	1	Master with MODF	Input-only to SPI

1. X = don't care

7.4 Memory Map Registers

Six registers control and monitor QSPI module operation. These registers should be accessed only with word accesses. Accesses with lengths other than word lengths result in undefined results. Before QSPI registers can be changed, the bit corresponding to the QSPI must be set to 1 in the appropriate PCE register of the SIM.

QSPI memory map

Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
FFFF_EC80	SPI Status and Control Register (QSPI_SPSCR)	16	R/W	6141h	7.4.1/130
FFFF_EC82	SPI Data Size and Control Register (QSPI_SPDSR)	16	R/W	See section	7.4.2/133
FFFF_EC84	SPI Data Receive Register (QSPI_SPDRR)	16	R	0000h	7.4.3/136
FFFF_EC86	SPI Data Transmit Register (QSPI_SPDTR)	16	W	0000h	7.4.4/137
FFFF_EC88	SPI FIFO Control Register (QSPI_SPFIFO)	16	R/W	000Ch	7.4.5/139
FFFF_EC8A	SPI Word Delay Register (QSPI_SPWAIT)	16	R/W	0000h	7.4.6/141

7.4.1 SPI Status and Control Register (QSPI_SPSCR)

This register does the following:

- Selects master SPI baud rate
- Determines data shift order
- Enables SPI module interrupt requests
- Configures the SPI module as master or slave
- Selects serial clock polarity and phase

NOTE

Using BFCLR or BFSET instructions on this register can cause unintended side effects on the status bits.

Address: FFFF_EC80h base + 0h offset = FFFF_EC80h

Bit	15	14	13	12	11	10	9	8
Read / Write	SPR[2:0]			DSO	ERRIE	MODFEN	SPRIE	SPMSTR
Reset	0	1	1	0	0	0	0	1

Bit	7	6	5	4	3	2	1	0
Read	CPOL	CPHA	SPE	SPTIE	SPRF	OVRF	MODF	SPTE
Write								
Reset	0	1	0	0	0	0	0	1

QSPI_SPSCR field descriptions

Field	Description

15–13
SPR[2:0]
	SPI Baud Rate Select
	
	In master mode, these read/write bits select one of eight baud rates. SPR2, SPR1, and SPR0 have no effect in slave mode. Reset sets SPR[2:0] to b011.
	
	Use the following formula to calculate the SPI baud rate:
	
	`Baud rate = clk/BD`
	
	where:
	
	`clk = Peripheral Bus Clock`
	`BD = baud rate divisor`
	
	Restriction: The maximum data transmission rate for the SPI is typically limited by the bandwidth of the I/O drivers on the chip, which can be a function of manufacturing technology. The typical limit in Normal mode is 40 MHz and in Wired-OR mode is 10 MHz. These baud rate limitations apply to both master and slave mode. The value of BD must be set to ensure the module remains within these ranges.
	
	NOTE: The value of BD can also depend on the values of the SPR3 and BD2X fields in the SPI Data Size and Control Register.
	
	000 BD = 2 when SPR3 = 0, BD = 512 when SPR3 = 1 (double BD when BD2X = 1)
	001 BD = 4 when SPR3 = 0, BD = 1024 when SPR3 = 1 (double BD when BD2X = 1)
	010 BD = 8 when SPR3 = 0, BD = 2048 when SPR3 = 1 (double BD when BD2X = 1)
	011 BD = 16 when SPR3 = 0, BD = 4096 when SPR3 = 1 (double BD when BD2X = 1)
	100 BD = 32 when SPR3 = 0, BD = 8192 when SPR3 = 1 (double BD when BD2X = 1)
	101 BD = 64 when SPR3 = 0 (double BD when BD2X = 1), BD = 16384 when SPR3 = 1 (regardless of BD2X)
	110 BD = 128 when SPR3 = 0 (double BD when BD2X = 1), BD = 16384 when SPR3 = 1 (regardless of BD2X)
	111 BD = 256 when SPR3 = 0 (double BD when BD2X = 1), BD = 16384 when SPR3 = 1 (regardless of BD2X)

12 DSO	Data Shift Order This read/write bit determines which bit is transmitted or received first, either the MSB or LSB. Both master and slave SPI modules must transmit and receive packets of the same length. Regardless of how this bit is set, when reading from the data receive register or writing to the data transmit register, the LSB is always at bit location 0 and the MSB is at the correct bit position. If the data length is less than 16 bits, the upper bits of data are zero padded. 0 MSB transmitted first (MSB -> LSB) 1 LSB transmitted first (LSB -> MSB)
11 ERRIE	Error Interrupt Enable This read/write bit enables the MODF (if MODFEN is also set) and OVRF bits to generate device interrupt requests. Reset clears the ERRIE bit.

Table continues on the next page...

QSPI_SPSCR field descriptions (continued)

Field	Description
	0 MODF and OVRF cannot generate device interrupt requests 1 MODF and OVRF can generate device interrupt requests
10 MODFEN	Mode Fault Enable This read/write bit, when set to 1, allows the MODF flag to be set. If the MODF flag is set, clearing the MODFEN bit does not clear the MODF flag. If the MODFEN bit is low, the level of the SS_B pin does not affect the operation of an enabled SPI configured as a master. If configured as a master and MODFEN=1, a transaction in progress will stop if SS_B goes low. For an enabled SPI configured as a slave, having MODFEN low only prevents the MODF flag from being set. It does not affect any other part of SPI operation.
9 SPRIE	SPI Receiver Interrupt Enable This read/write bit enables interrupt requests generated by the SPRF bit or the receive FIFO watermark register. 0 SPRF interrupt requests disabled 1 SPRF interrupt requests enabled
8 SPMSTR	SPI Master This read/write bit selects master mode operation or slave mode operation. 0 Slave mode 1 Master mode
7 CPOL	Clock Polarity This read/write bit determines the logic state of the SCLK pin between transactions. To transmit data between SPI modules, the SPI modules must have identical CPOL values. 0 Rising edge of SCLK starts transaction 1 Falling edge of SCLK starts transaction
6 CPHA	Clock Phase This read/write bit controls the timing relationship between the serial clock and SPI data. To transmit data between SPI modules, the SPI modules must have identical CPHA values. When CPHA = 0, the SS_B pin of the slave SPI module must be set to 1 between data words. To set SSB to 1 between data words when SSB_AUTO is 1, set SSB_STRB to 1.
5 SPE	SPI Enable This read/write bit enables the SPI module. Clearing SPE causes a partial reset of the SPI. **Restriction:** When you change the SPE bit, the write statement must change *only* the SPE bit. Change any other bits in a separate write statement. In master mode the SPE bit can be cleared by a mode fault condition. 0 SPI module disabled 1 SPI module enabled
4 SPTIE	Transmit Interrupt Enable This read/write bit enables interrupt requests generated by the SPTE bit or the transmit FIFO watermark register.

Table continues on the next page...

QSPI_SPSCR field descriptions (continued)

Field	Description
	0 SPTE interrupt requests disabled 1 SPTE interrupt requests enabled
3 SPRF	SPI Receiver Full This clearable, read-only flag is set each time data transfers from the shift register to the data receive register and no space is available in the RX queue to receive new data (RX FIFO is full). SPRF generates an interrupt request if the SPRIE bit in the SPI control register is set. This bit automatically clears after the data receive register is read. 0 Receive data register or FIFO is not full. (If using the FIFO, read RFCNT to determine the number of valid words available.) 1 Receive data register or FIFO is full.
2 OVRF	Overflow This clearable, read-only flag is set if software does not read the data in the receive data register before the next full data enters the shift register. In an overflow condition, the data already in the receive data register is unaffected, and the data shifted in last is lost. Clear the OVRF bit by reading the SPI status and control register with OVRF set and then reading the receive data register. 0 No overflow 1 Overflow
1 MODF	Mode Fault This clearable, read-only flag is set in a slave SPI if the SS_B pin goes high during a transaction with the MODFEN bit set. In a master SPI, the MODF flag is set if the SS_B pin goes low at any time with the MODFEN bit set. Clear the MODF bit by writing a one to the MODF bit when it is set. 0 SS_B pin at appropriate logic level 1 SS_B pin at inappropriate logic level
0 SPTE	SPI Transmitter Empty This clearable, read-only flag is set each time the transmit data register transfers data into the shift register and there is no more new data available in the TX queue (TX FIFO is empty). SPTE generates an interrupt request if the SPTIE bit in the SPI control register is set. SPTE is cleared by writing to the data transmit register. **CAUTION:** Do not write to the SPI data register unless the SPTE bit is high. Otherwise, data may be lost. 0 Transmit data register or FIFO is not empty. (If using the FIFO, read TFCNT to determine how many words can be written safely.) 1 Transmit data register or FIFO is empty.

7.4.2 SPI Data Size and Control Register (QSPI_SPDSR)

This read/write register determines the data length for each transaction. The master and slave must transfer the same size data on each transaction. A new value takes effect only at the time the SPI is enabled (the SPE bit in the status and control register is set from 0 to 1). To have a new value take effect, disable and then re-enable the SPI with the new value in the register.

To use the SS_B control functions in master mode, set the appropriate bit in a GPIO peripheral enable register to enable peripheral control of the SS_B pin.

Address: FFFF_EC80h base + 2h offset = FFFF_EC82h

Bit	15	14	13	12	11	10	9	8
Read	WOM	0		BD2X	SSB_IN	SSB_DATA	SSB_ODM	SSB_AUTO
Write								
Reset	0	0	0	0	x*	1	0	0

Bit	7	6	5	4	3	2	1	0
Read	SSB_DDR	SSB_STRB	SSB_OVER	SPR3	DS[3:0]			
Write								
Reset	0	0	0	0	1	1	1	1

* Notes:
- x = Undefined at reset.

QSPI_SPDSR field descriptions

Field	Description
15 WOM	Wired-OR Mode The Wired-OR mode (WOM) control bit is used to select the nature of the SPI pins. When enabled (the WOM bit is set), the SPI pins are configured as open-drain drivers. When disabled (the WOM bit is cleared), the SPI pins are configured as push-pull drivers. 0 The SPI pins are configured as push-pull drivers. 1 The SPI pins are configured as open-drain drivers with the pull-ups disabled.
14–13 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
12 BD2X	Baud Divisor Times Setting this bit causes the Baud Rate Divisor (BD) to be multiplied by two. The SPR bits define BD.
11 SSB_IN	SS_B Input This read only bit shows the current state of the SS_B pin in all modes. The bit's reset state is undefined.
10 SSB_DATA	SS_B Data This read/write bit is the value to drive on the SS_B pin. This bit is disabled when SSB_AUTO=1 or SSB_STRB=1. 0 SS_B pin is driven low if SSB_DDR=1 1 SS_B pin is driven high if SSB_DDR=1
9 SSB_ODM	SS_B Open Drain Mode This read/write bit enables open drain mode on the SS_B pin in master mode. 0 SS_B is configured for high and low drive. This mode is generally used in single master systems. 1 SS_B is configured as an open drain pin (only drives low output level). This mode is useful for multiple master systems.

Table continues on the next page...

QSPI_SPDSR field descriptions (continued)

Field	Description
8 SSB_AUTO	SS_B Automatic Mode This read/write bit enables hardware control of the SS_B pin in master mode. (The legacy design requires software to control the SS_B output pin.) The initial falling edge of SS_B is generated and SS_B is held low until the TX buffer or FIFO is empty. This bit may be used alone or in combination with SS_STRB to generate the required SS_B signal. 0 SS_B output signal is software generated by directly manipulating the various bits in this register or the GPIO registers (compatible with legacy SPI software). 1 SS_B output signal is hardware generated to create the initial falling edge and final rising edge. The idle state of the SS_B is high. **Restriction:** Do not use if MODFEN = 1.
7 SSB_DDR	SS_B Data Direction This read/write bit controls input/output mode on the SS_B pin in master mode. 0 SS_B is configured as an input pin. Use this setting in slave mode or in master mode with MODFEN=1. 1 SS_B is configured as an output pin. Use this setting in master mode with MODFEN=0.
6 SSB_STRB	SS_B Strobe Mode This read/write bit enables hardware pulse of the SS_B pin in master mode between words. This bit may be used alone or in combination with the SS_AUTO to generate the required SS_B signal. Pulses are generated between words irrespective of the setting of CPHA. 0 No SS_B pulse between words. 1 SS_B output signal is pulsed high between words. This adds 1.5 baud clocks to the total word period. The idle state of SS_B is low unless SSB_AUTO is high and then the idle state is high. **Restriction:** Do not use if MODFEN = 1.
5 SSB_OVER	SS_B Override This read/write bit overrides the internal SS_B signal input from the I/O pad and replaces it with a level equal to the setting of the SPMSTR bit. This allows the SPI to function in slave mode, when CPHA=1, without commiting a GPIO pin to be tied low. **Restriction:** This bit should not be used in multi-slave systems or when CPHA=0. **Restriction:** This bit should not be used in a multi-master system because in master mode a mode fault error cannot be generated. 0 SS_B internal module input is selected to be connected to a GPIO pin. 1 SS_B internal module input is selected to be equal to SPMSTR.
4 SPR3	SPI Baud Rate Select Use this bit with SPR[2:0] in the status and control register and BD2X to define the Baud Rate Divisor (BD).
DS[3:0]	Transaction data size 4'h0 Not allowed 4'h1 2 bits transaction data size 4'h2 3 bits transaction data size 4'h3 4 bits transaction data size

Table continues on the next page...

QSPI_SPDSR field descriptions (continued)

Field	Description
	4'h4 5 bits transaction data size 4'h5 6 bits transaction data size 4'h6 7 bits transaction data size 4'h7 8 bits transaction data size 4'h8 9 bits transaction data size 4'h9 10 bits transaction data size 4'hA 11 bits transaction data size 4'hB 12 bits transaction data size 4'hC 13 bits transaction data size 4'hD 14 bits transaction data size 4'hE 15 bits transaction data size 4'hF 16 bits transaction data size

7.4.3 SPI Data Receive Register (QSPI_SPDRR)

The SPI data receive register is read-only. Reading data from the register shows the last data received after a complete transaction. The SPRF bit is set when new data is transferred to this register.

Address: FFFF_EC80h base + 4h offset = FFFF_EC84h

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	R15	R14	R13	R12	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

QSPI_SPDRR field descriptions

Field	Description
15 R15	Receive Data Bit 15
14 R14	Receive Data Bit 14
13 R13	Receive Data Bit 13
12 R12	Receive Data Bit 12
11 R11	Receive Data Bit 11
10 R10	Receive Data Bit 10

Table continues on the next page...

QSPI_SPDRR field descriptions (continued)

Field	Description
9 R9	Receive Data Bit 9
8 R8	Receive Data Bit 8
7 R7	Receive Data Bit 7
6 R6	Receive Data Bit 6
5 R5	Receive Data Bit 5
4 R4	Receive Data Bit 4
3 R3	Receive Data Bit 3
2 R2	Receive Data Bit 2
1 R1	Receive Data Bit 1
0 R0	Receive Data Bit 0

7.4.4 SPI Data Transmit Register (QSPI_SPDTR)

The SPI data transmit register is write-only. Writing data to this register writes the data to the transmit data buffer. When the SPTE bit is set, new data should be written to this register. If new data is not written while in master mode, a new transaction will not begin until this register is written.

When selected in slave mode, the old data will be re-transmitted. When *not* selected and in slave mode, transmit data will remain unchanged. All data should be written with the LSB at bit 0. This register can only be written when the SPI is enabled (SPE = 1).

Address: FFFF_EC80h base + 6h offset = FFFF_EC86h

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read																
Write	T15	T14	T13	T12	T11	T10	T9	T8	T7	T6	T5	T4	T3	T2	T1	T0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

QSPI_SPDTR field descriptions

Field	Description
15 T15	Transmit Data Bit 15
14 T14	Transmit Data Bit 14
13 T13	Transmit Data Bit 13
12 T12	Transmit Data Bit 12
11 T11	Transmit Data Bit 11
10 T10	Transmit Data Bit 10
9 T9	Transmit Data Bit 9
8 T8	Transmit Data Bit 8
7 T7	Transmit Data Bit 7
6 T6	Transmit Data Bit 6
5 T5	Transmit Data Bit 5
4 T4	Transmit Data Bit 4
3 T3	Transmit Data Bit 3
2 T2	Transmit Data Bit 2
1 T1	Transmit Data Bit 1
0 T0	Transmit Data Bit 0

7.4.5 SPI FIFO Control Register (QSPI_SPFIFO)

This register is used for FIFO control and status.

Address: FFFF_EC80h base + 8h offset = FFFF_EC88h

Bit	15	14	13	12	11	10	9	8
Read	0	TFCNT			0	RFCNT		
Write								
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
Read	0	TFWM		0	RFWM		0	FIFO_ENA
Write								
Reset	0	0	0	0	1	1	0	0

QSPI_SPFIFO field descriptions

Field	Description
15 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
14–12 TFCNT	TX FIFO Level These read-only bits show how many words are used in the TX FIFO. Writes to the data transmit register cause TFCNT to increment, and, as words are pulled for transmission, TFCNT is decremented. Attempts to write new data to the data transmit register are ignored when TFCNT indicates the FIFO is full. If master mode is enabled, transmission continues until the FIFO is empty, even if SPE is set to 0. 000 Tx FIFO empty (if enabled Transmit Empty Interrupt asserted) 001 One word used in Tx FIFO 010 Two words used in Tx FIFO 011 Three words used in Tx FIFO 100 Tx FIFO full
11 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
10–8 RFCNT	RX FIFO Level These read-only bits show how many words are used in the RX FIFO. As words are received, the value of RFCNT is incremented; as words are read from the data receive register, the value of RFCNT is decremented. There is one word time to read the data receive register between when the SPRF status bit is set (interrupt asserted) and when an overflow condition is flagged. 000 Rx FIFO empty 001 One word used in Rx FIFO 010 Two words used in Rx FIFO 011 Three words used in Rx FIFO 100 Rx FIFO full (if enabled Receiver Full Interrupt asserted)

Table continues on the next page...

QSPI_SPFIFO field descriptions (continued)

Field	Description
7 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
6–5 TFWM	Tx FIFO Watermark These read/write bits determine how many words must remain in the Tx FIFO before an interrupt is generated. Increasing the value of TFWM increases the allowable latency in servicing the Tx interrupt without underrunning the Tx buffer space. Larger values of TFWM may also increase the number of Tx interrupt service requests because the maximum number of Tx words may not be available when the service routine is activated. If TFWM is set to the minimum value then only one SPI word time in interrupt service latency is allowed before an underrun condition results and continuous transmission is stopped in master mode or the last data word is re-transmitted in slave mode. This field is ignored when FIFO_ENA = 0. To clear an interrupt generated by TFWM, new words must be written to the data transmit register or the value of TFWM must be reduced. 00 Transmit interrupt active when Tx FIFO is empty 01 Transmit interrupt active when Tx FIFO has one or fewer words available 10 Transmit interrupt active when Tx FIFO has two or fewer words available 11 Transmit interrupt active when Tx FIFO has three or fewer words available
4 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
3–2 RFWM	Rx FIFO Watermark These read/write bits determine how many words must be used in the Rx FIFO before an interrupt is generated. Decreasing the value of RFWM increases the allowable latency in servicing the Rx interrupt without overrunning the Rx buffer space. Smaller values of RFWM may also increase the number of Rx interrupt service requests because the maximum number of Rx words may not have been used when the service routine is activated. If RFWM is set to the maximum value then only one SPI word time in interrupt service latency is allowed before an overrun condition results and receive data is lost. This field is ignored when FIFO_ENA = 0. To clear an interrupt generated by RFWM, words must be read from the data receive register or the value of RFWM must be increased. 00 Receive interrupt active when Rx FIFO has at least one word used 01 Receive interrupt active when Rx FIFO has at least two words used 10 Receive interrupt active when Rx FIFO has at least three words used 11 Receive interrupt active when Rx FIFO is full
1 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
0 FIFO_ENA	FIFO Enable This read/write bit enables Tx and Rx FIFOs' mode. 0 FIFOs are disabled and reset. 1 FIFOs are enabled. FIFOs retain their status even if SPE is set to 0.

7.4.6 SPI Word Delay Register (QSPI_SPWAIT)

This register is used to control the delay between words.

Address: FFFF_EC80h base + Ah offset = FFFF_EC8Ah

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0			WAIT												
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

QSPI_SPWAIT field descriptions

Field	Description
15–13 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
WAIT	Wait Delay This 13-bit register controls the time between data transactions in master mode. It sets the delay between words to be a number of Peripheral Bus Clocks equal to (WAIT + 1). This delay is used only when a word is waiting to be transmitted at the completion of the transmission of the current word. If no word is waiting to be transmitted, the SPI goes idle at the completion of the current transmission and subsequently starts transmission immediately when a new word is written to the Data Transmit register.

7.5 Functional Description

7.5.1 Operating Modes

7.5.1.1 Master Mode

The SPI operates in master mode when the SPI master bit, SPMSTR, is set.

Note

Configure the SPI module as master or slave before enabling the SPI. Enable the master SPI before enabling the slave SPI. Disable the slave SPI before disabling the master SPI.

Only a master SPI module can initiate transactions. With the SPI enabled, software begins the transaction from the master SPI module by writing to the transmit data register. If the shift register is empty, the data immediately transfers to the shift register, setting the SPI transmitter empty bit, SPTE. The data begins shifting out on the MOSI pin under the control of the SPI serial clock, SCLK.

The SPR3, SPR2, SPR1, and SPR0 bits in the SPI registers control the baud rate generator and determine the speed of the shift register. Through the SCLK pin, the baud rate generator of the master also controls the shift register of the slave peripheral

As the data shifts out on the MOSI pin of the master, external data shifts in from the slave on the master's MISO pin. The transaction ends when the receiver full bit, SPRF, becomes set. At the same time that SPRF becomes set, the data from the slave transfers to the SPI Data Receive register. In normal operation, SPRF signals the end of a transaction. Software clears SPRF by reading the SPI Data Receive register. Writing to the SPI Data Transmit register clears the SPTE bit.

The following figure is an example configuration for a full-duplex master-slave configuration. Having the $\overline{SS}$ bit of the master device held high is only necessary if MODFEN = 1. Tying the slave $\overline{SS}$ bit to ground should only be done if CPHA = 1.

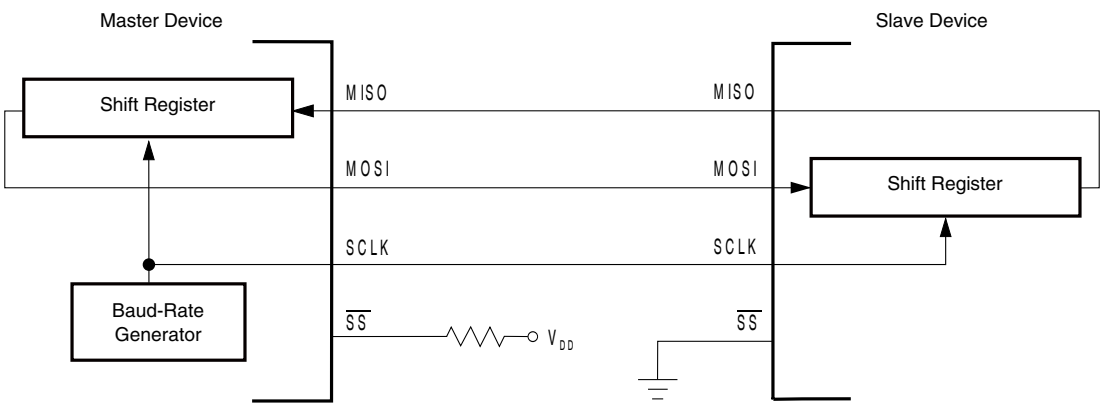


Figure 7-8. Full-Duplex Master-Slave Connections

7.5.1.2 Slave Mode

The SPI operates in slave mode when the SPMSTR bit is 0. In slave mode the SCLK pin is the input for the serial clock from the master device. Before a data transaction occurs, the SS pin of the slave SPI must be at logic zero. $\overline{SS}$ must remain low until the transaction completes or a mode fault error occurs.

Note

The SPI must be enabled (SPE = 1) for slave transactions to be received.

Note

Data in the transmitter shift register is unaffected by SCLK transitions when the SPI operates as a slave but is deselected ($\overline{SS}$ = 1).

In a slave SPI module, data enters the shift register under the control of the serial clock, SCLK, from the master SPI module. After a full data word enters the shift register of a slave SPI, it transfers to the SPI Data Receive register, and the SPRF bit is set. To prevent an overflow condition, slave software then must read the receive data register before another full data word enters the shift register.

The maximum frequency of the SCLK for an SPI configured as a slave is less than 1/2 the bus clock frequency. The frequency of the SCLK for an SPI configured as a slave does not have to correspond to any SPI baud rate as defined by the SPR bits. The SPR bits control only the speed of the SCLK generated by an SPI configured as a master.

When the master SPI starts a transaction, the data in the slave shift register begins shifting out on the MISO pin. The slave can load its shift register with new data for the next transaction by writing to its transmit data register. The slave must write to its transmit data register at least one bus cycle before the master starts the next transaction. Otherwise, the data that was last transmitted is reloaded into the slave shift register and shifts out on the MISO pin again. Data written to the slave shift register during a transaction remains in a buffer until the end of the transaction.

When the clock phase bit (CPHA) is set, the first edge of SCLK starts a transaction. When CPHA is clear, the falling edge of $\overline{SS}$ starts a transaction.

Note

SCLK must be in the proper idle state before the slave is enabled to preserve the proper SCLK, MISO, MOSI timing relationships.

7.5.1.3 Wired-OR Mode

Wired-OR functionality is provided to permit the connection of multiple SPIs. The following figure illustrates the sharing of a single master device between multiple slave SPIs. When the WOM bit is set, the outputs switch from conventional complementary CMOS output to open drain outputs.

This internal pullup resistor brings the line high, and whichever SPI drives the line pulls it low as needed.

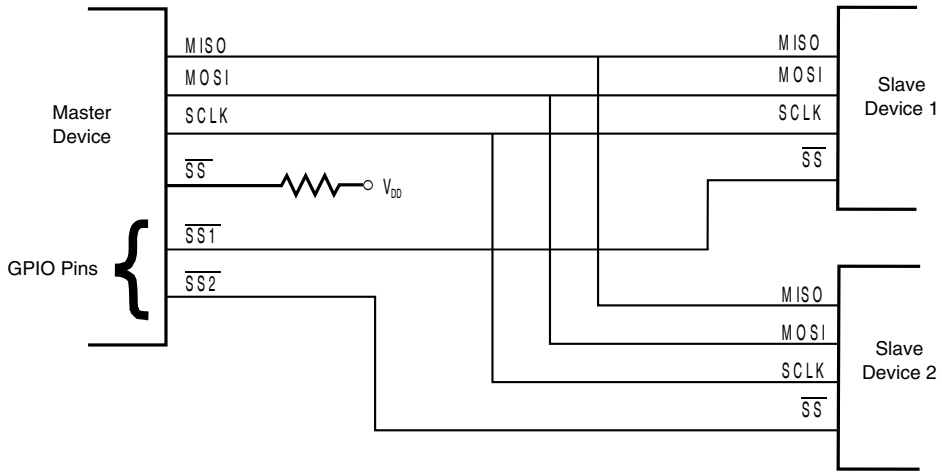


Figure 7-9. Master With Two Slaves

7.5.2 Transaction Formats

During an SPI transaction, data is simultaneously transmitted (shifted out serially) and received (shifted in serially). A serial clock synchronizes shifting and sampling on the two serial data lines. A slave select line enables selection of an individual slave SPI device. Slave devices that are not selected do not interfere with SPI bus activities. On a master SPI device, the slave select line can optionally be used to indicate multiple-master bus contention.

7.5.2.1 Data Transaction Length

The SPI can support data lengths of 2 to 16 bits. The length can be configured in the SPI Data Size and Control register. When the data length is less than 16 bits, the receive data register pads the upper bits with zeros.

Note

Data can be lost if the data length is not the same for both master and slave devices.

7.5.2.2 Data Shift Ordering

The SPI can be configured to transmit or receive the MSB of the desired data first or last, using the DSO bit in the SPI Status and Control register. Regardless of which bit is transmitted or received first, the data is always written to the SPI Data Transmit register and read from the SPI Data Receive register with the LSB in bit 0 and the MSB in correct position depending on the data transaction size.

7.5.2.3 Clock Phase and Polarity Controls

Software can select any of four combinations of serial clock (SCLK) phase and polarity using two bits in the SPI Status and Control register. The clock polarity is specified by the CPOL control bit, which selects an active high or low clock and has no significant effect on the transaction format.

The clock phase (CPHA) control bit selects one of two fundamentally different transaction formats. The clock phase and polarity should be identical for the master SPI device and the communicating slave device. In some cases, the phase and polarity are changed between transactions, to enable a master device to communicate with peripheral slaves having different requirements.

Note

Before writing to the CPOL bit or the CPHA bit, disable the SPI (by clearing the SPI enable bit (SPE)).

7.5.2.4 Transaction Format When CPHA = 0

The following figure shows an SPI transaction in which CPHA is logic zero. The figure should not be used as a replacement for data sheet parametric information. It assumes 16 bit data lengths and the MSB shifted out first.

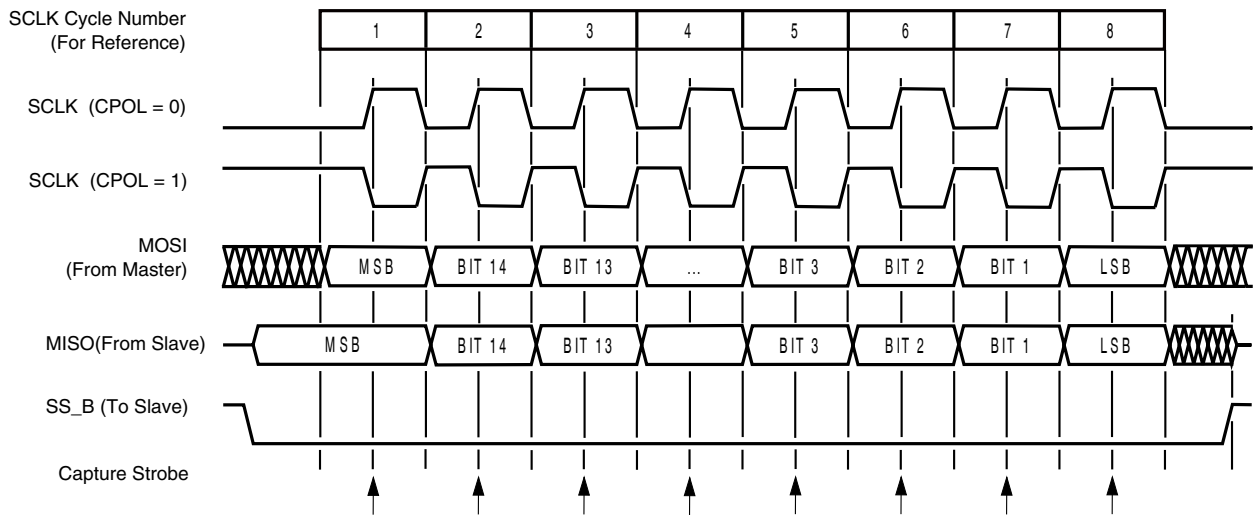


Figure 7-10. Transaction Format (CPHA = 0)

Two waveforms are shown for SCLK: one for CPOL = 0 and another for CPOL = 1. The diagram may be interpreted as a master or slave timing diagram since the serial clock (SCLK), master in/slave out (MISO), and master out/slave in (MOSI) pins are directly connected between the master and the slave. The MISO signal is the output from the slave, and the MOSI signal is the output from the master. When CPHA = 0, the first SCLK edge is the MSB capture strobe. Therefore, the slave must begin driving its data before the first SCLK edge, and a falling edge on the $\overline{SS}$ pin is used to start the slave data transaction. The slave $\overline{SS}$ pin must be toggled back to high and then low again between each data word transmitted, as shown in the following figure.

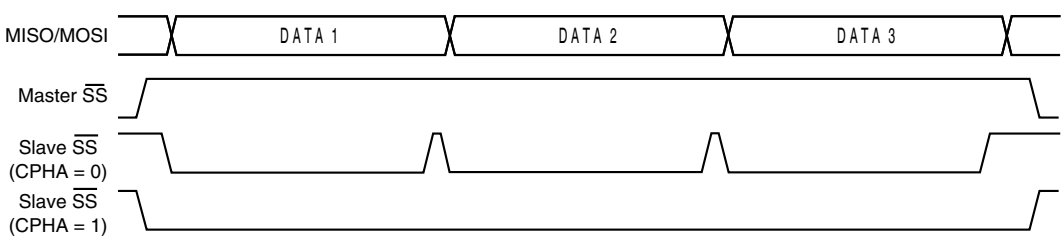


Figure 7-11. CPHA /$\overline{SS}$ Timing

When CPHA = 0 for a slave, the falling edge of $\overline{SS}$ indicates the beginning of the transaction. This causes the SPI to leave its idle state and begin driving the MISO pin with the first bit of its data. After the transaction begins, no new data is allowed into the shift register from the transmit data register. Therefore, the SPI data register of the slave must be loaded with transmit data before the falling edge of $\overline{SS}$. Any data written after the falling edge is stored in the transmit data register and transferred to the shift register after the current transaction. Also, for correct operation of the slave, SPE must be active before the negative edge of $\overline{SS}$ to correctly send/receive the first word. The $\overline{SS}$ line is the slave select input to the slave. The slave SPI drives its MISO output only when its slave select input ($\overline{SS}$) is at logic zero, so that only the selected slave drives to the master.

When CPHA = 0 for a master, normal operation would begin by the master initializing the $\overline{SS}$ pin of the slave high. A transfer would then begin by the master setting the $\overline{SS}$ pin of the slave low and then writing the SPI Data Transmit register. After a data transfer completes, the master device puts the $\overline{SS}$ pin back into the high state. While MODFEN = 1, the $\overline{SS}$ pin of the master must be high or a mode fault error occurs. If MODFEN = 0, the state of the $\overline{SS}$ pin is ignored.

7.5.2.5 Transaction Format When CPHA = 1

The following figure shows an SPI transaction in which CPHA is logic one. The figure should not be used as a replacement for data sheet parametric information. It assumes 16 bit data lengths and the MSB shifted out first.

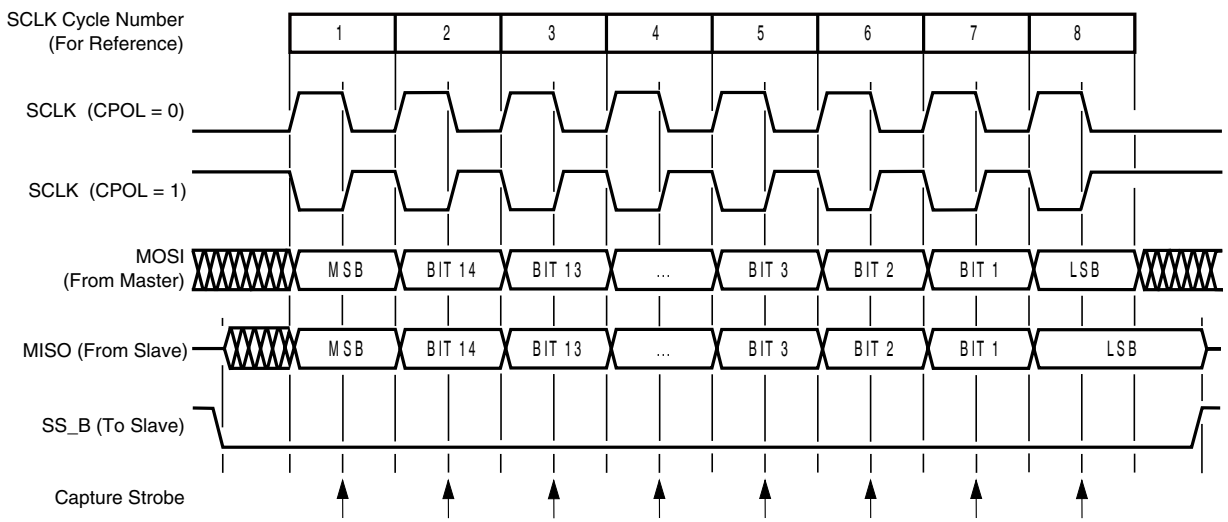


Figure 7-12. Transaction Format (CPHA = 1)

Two waveforms are shown for SCLK: one for CPOL = 0 and another for CPOL = 1. The diagram may be interpreted as a master or slave timing diagram since the serial clock (SCLK), master in/slave out (MISO), and master out/slave in (MOSI) pins are directly connected between the master and the slave. The MISO signal is the output from the slave, and the MOSI signal is the output from the master.

When CPHA = 1 for a slave, the first edge of the SCLK indicates the beginning of the transaction. This causes the SPI to leave its idle state and begin driving the MISO pin with the first bit of its data. After the transaction begins, no new data is allowed into the shift register from the transmit data register. Therefore, the SPI data register of the slave must be loaded with transmit data before the first edge of SCLK. Any data written after the first edge is stored in the transmit data register and transferred to the shift register after the current transaction. The $\overline{SS}$ line is the slave select input to the slave. The slave SPI drives its MISO output only when its slave select input ($\overline{SS}$) is at logic zero, so that only the selected slave drives to the master.

When CPHA = 1 for a master, the MOSI pin begins being driven with new data on the first SCLK edge. If MODFEN = 0 the $\overline{SS}$ pin of the master is ignored. Otherwise, the $\overline{SS}$ pin of the master must be high or a mode fault error occurs. The $\overline{SS}$ pin can remain low between transactions. This format may be preferable in systems with only one master and one slave driving the MISO data line.

7.5.2.6 Transaction Initiation Latency

When the SPI is configured as a master (SPMSTR is 1), writing to the SPI Data Transmit register starts a transaction. CPHA has no effect on the delay to the start of the transaction, but it does affect the initial state of the SCLK signal. When CPHA = 0, the SCLK signal remains inactive for the first half of the first SCLK cycle. When CPHA = 1, the first SCLK cycle begins with an edge on the SCLK line from its inactive to its active level. The SPI clock rate (selected by SPR2, SPR1, and SPR0) affects the delay from the write to the SPI Data Transmit register and the start of the SPI transaction. The internal baud clock in the master is a derivative of the internal device clock. To conserve power, it is enabled only after the SPMSTR bit is set and there is a new word written to the SPI Data Transmit register. If the SPI Data Transmit register has no new word when the current transaction completes, the internal baud clock is stopped. The initiation delay is a single SPI bit time, as the following figure shows. That is, the delay is 4 bus cycles for DIV4, 8 bus cycles for DIV8, 16 bus cycles for DIV16, 32 bus cycles for DIV32, and so on.

Note

The following figure assumes 16-bit data lengths and the MSB shifted out first.

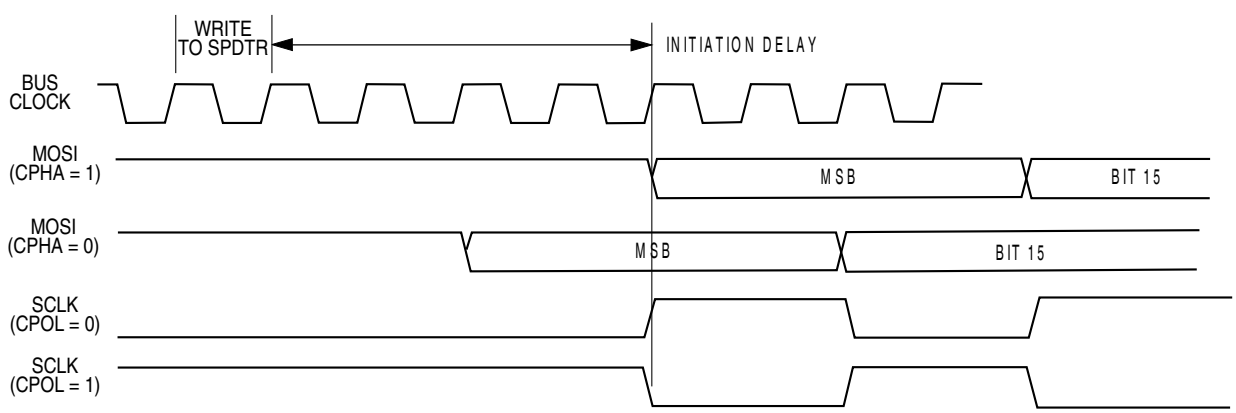


Figure 7-13. Transaction Start Delay (Master)

7.5.2.7 $\overline{SS}$ Hardware-Generated Timing in Master Mode

If the SSB_STRB bit is set in master mode, the SPI generates a word strobe pulse on $\overline{SS}$ for a slave device (see the following figures).

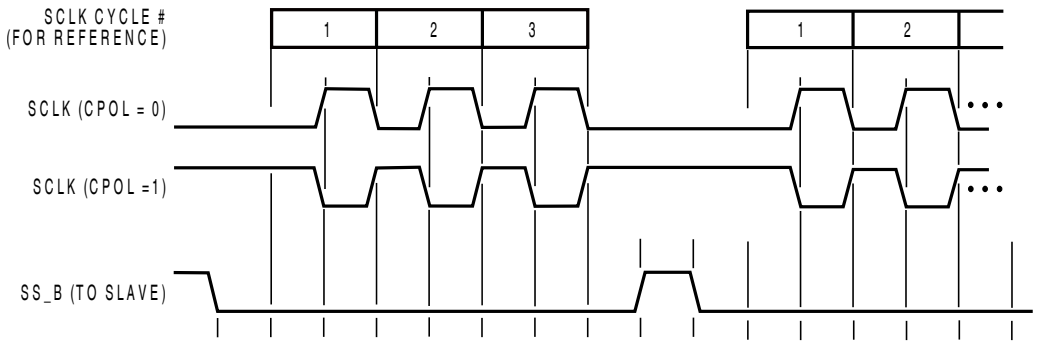


Figure 7-14. $\overline{SS}$ Strobe Timing (CPHA = 0)

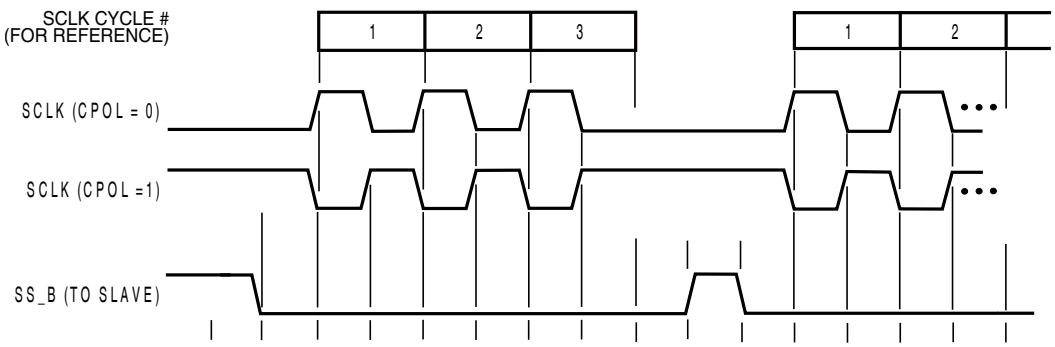


Figure 7-15. $\overline{SS}$ Strobe Timing (CPHA = 1)

If the SSB_AUTO bit is set in master mode, the SPI generates the initial falling edge and the final rising edge of $\overline{SS}$ for a slave device. The $\overline{SS}$ output has a falling edge one bit time before the first edge of SCLK (see the following figure).

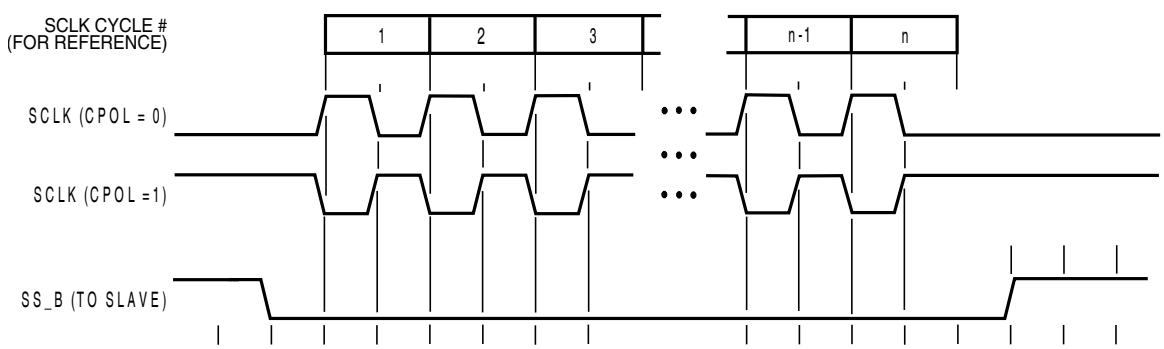


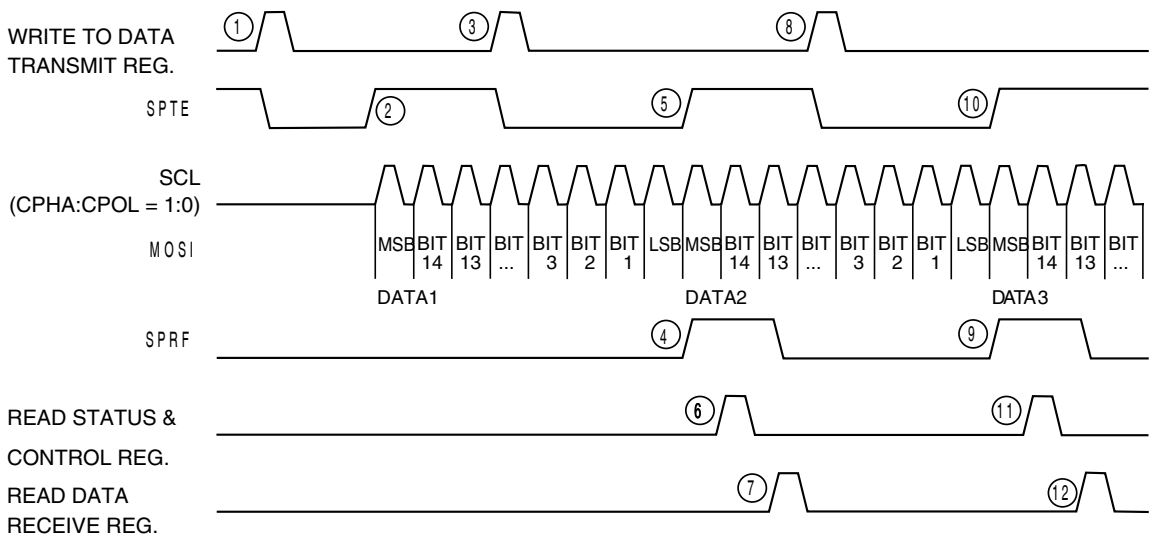
Figure 7-16. $\overline{SS}$ Auto Timing (CPHA = 1)

7.5.3 Transmission Data

The double-buffered data transmit register enables data to be queued and transmitted. For an SPI configured as a master, the queued data is transmitted immediately after the previous transaction has completed. The SPI transmitter empty flag (SPTE) indicates when the transmit data buffer is ready to accept new data. Write to the data transmit register only when the SPTE bit is high. The following figure shows the timing associated with doing back-to-back transactions with the SPI (SCLK has CPHA = 1; CPOL = 0).

Note

The following figure assumes 16-bit data lengths and the MSB shifted out first.



1. Device writes data 1 to Data Transmit register, clearing the SPTE bit.
2. Data 1 transfers from the Data Transmit register to the shift register, setting the SPTE bit.
3. Device writes data 2 to Data Transmit register, queueing data 2 and clearing the SPTE bit.
4. First incoming word transfers from shift register to the Data Receive register, settng the SPRF bit.
5. Data 2 transfers from the Data Transmit register to the shift register, setting the SPTE bit.
6. Device reads Status and Control register with the SPRF bit set.
7. Device reads Data Receive register, clearing the SPRF bit.
8. Device writes data 3 to Data Transmit register, queueing data 3 and clearing the SPTE bit.
9. The second incoming data transfers from the shift register to the Receive Data Register, setting the SPRF bit.
10. Data 3 transfers from the Data Transmit register to the shift register, setting the SPTE bit.
11. Device reads Status and Control register with the SPRF bit set.
12. Device reads Data Receive register, clearing the SPRF bit.

Figure 7-17. SPRF/SPTE Interrupt Timing

The transmit data buffer enables back-to-back transactions without the slave precisely timing its writes between transactions, as occurs in a system with a single data buffer. Also, in slave mode, if no new data is written to the SPI Data Transmit register, the last value contained in the SPI Data Transmit register is retransmitted if the external master starts a new transaction.

For an idle master that has no data loaded into its transmit buffer and no word currently being transmitted, the SPTE is set again no more than two bus cycles after the SPI Data Transmit register is written. This enables the user to queue up at most a 32-bit value to send. For an SPI operating in slave mode, the load of the shift register is controlled by the external master, and back-to-back writes to the transmit data register are not possible. The SPTE bit indicates when the next write can occur.

7.5.4 Error Conditions

The following flags signal SPI error conditions:

- Overflow (OVRF) — Failing to read the SPI Data Receive register before the next data word finishes entering the shift register sets the OVRF bit. The new data word does not transfer to the Receive Data register, and the unread data word can still be read. OVRF is in the SPI Status and Control register.
- Mode fault error (MODF) — The MODF bit indicates that the voltage on the slave select pin ($\overline{SS}$) is inconsistent with the mode of the SPI. MODF is in the SPI Status and Control register.

7.5.4.1 Overflow Error

The overflow flag (OVRF) is set if the SPI Receive Data register still has unread data from a previous transaction when the capture strobe of bit 1 of the next transaction occurs. The bit 1 capture strobe occurs in the middle of SCLK when the data length equals transaction data length minus 1. If an overflow occurs, all data received after the overflow and before the OVRF bit is cleared does not transfer to the SPI Receive Data register and does not set the SPI receiver full bit (SPRF). The unread data that is transferred to the SPI Receive Data register before the overflow occurred can still be read. Therefore, an overflow error always indicates the loss of data. Clear the overflow flag by reading the SPI Status and Control register and then reading the SPI Receive Data register.

OVRF generates a receiver/error interrupt request if the error interrupt enable bit (ERRIE) is also set. It is not possible to enable MODF or OVRF individually to generate a receiver/error interrupt request. However, leaving MODFEN low prevents MODF from being set.

If the SPRF interrupt is enabled and the ERRIE is not enabled, poll the OVRF bit to detect an overflow condition. The following figure shows how it is possible to miss an overflow. The first part of the figure shows how it is possible to read the SPI Status and Control register and the SPI Data Receive register to clear the SPRF without problems. However, as illustrated by the second transaction example, the OVRF bit can be set in between the time that the SPI Status and Control register and the SPI Data Receive register are read.

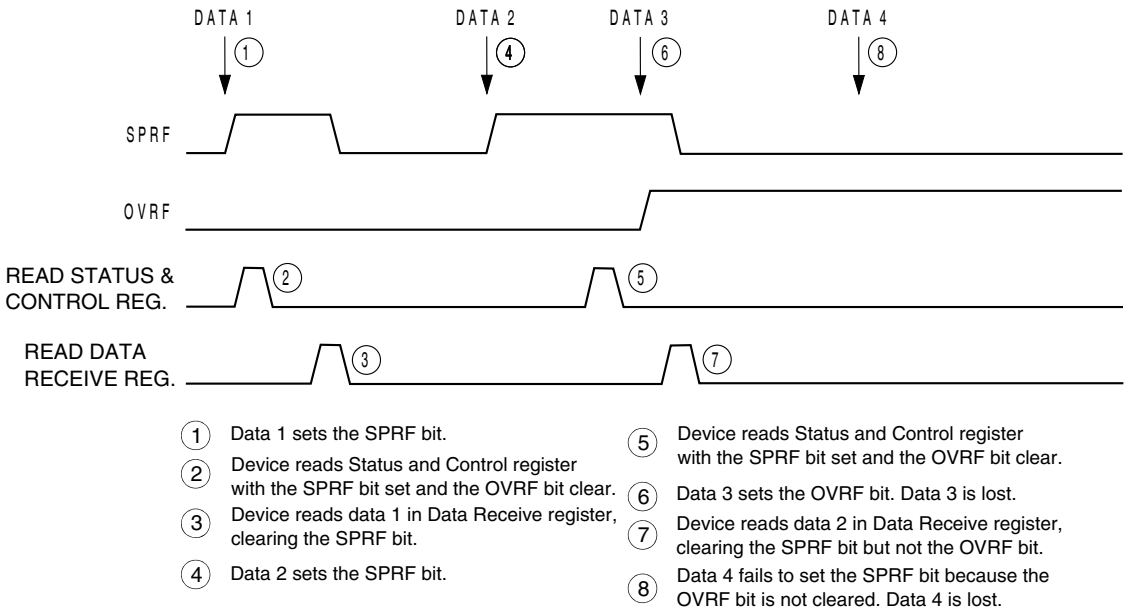


Figure 7-18. Missed Read of Overflow Condition

In this case, an overflow can easily be missed. Since no more SPRF interrupts can be generated until this OVRF is serviced, it is not obvious that data is being lost as more transactions are completed. To prevent this, either enable the OVRF interrupt or do another read of the SPI Status and Control register following the read of the SPI Data Receive register. This ensures that the OVRF was not set before the SPRF was cleared and that future transactions can set the SPRF bit. The following figure illustrates this process. Generally, to avoid this second read of the SPI Status and Control register, enable the OVRF to the device by setting the ERRIE bit.

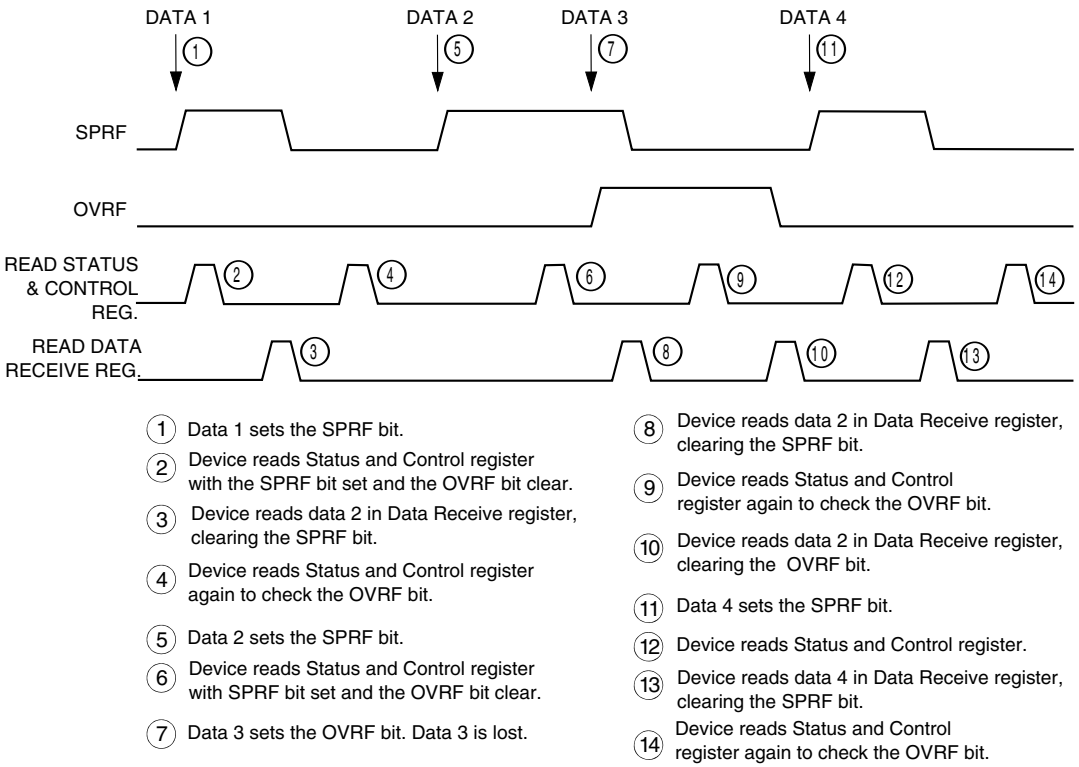


Figure 7-19. Clearing SPRF When OVRF Interrupt Is Not Enabled

7.5.4.2 Mode Fault Error

Setting the SPMSTR bit selects master mode and configures the SCLK and MOSI pins as outputs and the MISO pin as an input. Clearing SPMSTR selects slave mode and configures the SCLK and MOSI pins as inputs and the MISO pin as an output. The mode fault bit, MODF, is set any time the state of the slave select pin, $\overline{SS}$, is inconsistent with the mode selected by SPMSTR. To prevent SPI pin contention and damage to the device, a mode fault error occurs if:

- The $\overline{SS}$ pin of a slave SPI goes high during a transaction.
- The $\overline{SS}$ pin of a master SPI goes low at any time.

For the MODF flag to be set, the mode fault error enable bit (MODFEN) must be set. Clearing the MODFEN bit does not clear the MODF flag but does prevent MODF from being set again after MODF is cleared.

MODF generates a receiver/error interrupt request if the error interrupt enable bit (ERRIE) is also set. It is not possible to enable MODF or OVRF individually to generate a receiver/error interrupt request. However, leaving MODFEN low prevents MODF from being set.

7.5.4.2.1 Master Mode Fault

In a master SPI with the mode fault enable bit (MODFEN) set, the mode fault flag (MODF) is set if $\overline{SS}$ goes to logic zero. A mode fault in a master SPI causes the following events to occur:

- If ERRIE = 1, the SPI generates an SPI receiver/error interrupt request.
- The SPE bit is cleared (SPI disabled).
- The SPTE bit is set.
- The SPI state counter is cleared.

Note

Setting the MODF flag does not clear the SPMSTR bit. The SPMSTR bit has no function when SPE = 0. Reading SPMSTR when MODF = 1 shows the difference between a MODF occurring when the SPI is a master and when it is a slave.

In a master SPI, the MODF flag is not cleared until the $\overline{SS}$ pin is at a logic one or the SPI is configured as a slave.

7.5.4.2.2 Slave Mode Fault

When configured as a slave (SPMSTR = 0), the MODF flag is set if the $\overline{SS}$ pin goes high during a transaction. When CPHA = 0, a transaction begins when $\overline{SS}$ goes low and ends after the incoming SCLK goes back to its idle level following the shift of the last data bit. When CPHA = 1, the transaction begins when the SCLK leaves its idle level and $\overline{SS}$ is already low. The transaction continues until the SCLK returns to its idle level following the shift of the last data bit.

In a slave SPI (SPMSTR = 0), the MODF bit generates an SPI receiver/error interrupt request if the ERRIE bit is set. The MODF bit does not clear the SPE bit or reset the SPI in any way. Software can abort the SPI transaction by clearing the SPE bit of the slave.

Note

A logic one voltage on the $\overline{SS}$ pin of a slave SPI puts the MISO pin in a high impedance state. Also, the slave SPI ignores all incoming SCLK clocks, even if it was already in the middle of a transaction. A mode fault occurs if the $\overline{SS}$ pin changes state during a transaction.

When CPHA = 0, a MODF occurs if a slave is selected ($\overline{SS}$ is at logic 0) and later unselected ($\overline{SS}$ is at logic 1) after the first bit of data has been received (SCLK is toggled at least once). This happens because $\overline{SS}$ at logic 0 indicates the start of the transaction (MISO driven out with the value of MSB) for CPHA = 0. When CPHA = 1, a slave can be selected and then later unselected with no transaction occurring. Therefore, MODF does not occur because a transaction was never begun.

To clear the MODF flag, write a one to the MODF bit in the SPI Status and Control register. If the MODF flag is not cleared by writing a one to the MODF bit, the condition causing the mode fault still exists. In this case, the interrupt caused by the MODF flag can be cleared by disabling the EERIE bit or MODFEN bit (if set) or by disabling the SPI.

7.5.5 Resetting the SPI

Any system reset completely resets the SPI. Partial resets occur whenever the SPI enable bit (SPE) is low. Whenever SPE is low, the following occurs:

1. The SPTE flag is set.
2. Any slave mode transaction currently in progress is aborted.
3. Any master mode transaction currently in progress continues to completion.
4. The SPI state counter is cleared, making it ready for a new complete transaction.
5. All the SPI port logic is disabled.

Items 4 and 5 occur after 2 in slave mode, or after 3 in master mode.

The following items are reset only by a system reset:

- The SPI Data Transmit and SPI Data Receive registers
- All control bits in the SPI Status and Control register and the SPI Data Size and Control register
- The status flags SPRF, OVRF, and MODF

By not resetting the control bits when SPE is low, the user can clear SPE between transactions without having to set all control bits again when SPE is set back high for the next transaction.

By not resetting the SPRF, OVRF, and MODF flags, the user can still service these interrupts after the SPI is disabled. The user can disable the SPI by writing 0 to the SPE bit. The SPI is also disabled when a mode fault occurs in an SPI configured as a master.

7.6 Interrupts

Four SPI status flags can be enabled to generate device interrupt requests.

Table 7-11. SPI Interrupts

Flag	Interrupt Enabled By	Description
SPTE (Transmitter Empty)	SPI Enable / SPI Transmitter Interrupt Enable (SPTIE = 1, SPE = 1)	The SPI transmitter interrupt enable bit (SPTIE) enables the SPI transmitter empty (SPTE) flag or TFWM to generate transmitter interrupt requests, provided that the SPI is enabled (SPE = 1). The SPTE bit becomes set every time data transfers from the SPI Data Transmit register to the shift register and there is no more new data available in the TX queue. The clearing mechanism for the SPTE flag is a write to the SPI Data Transmit register.
SPRF (Receiver Full)	SPI Receiver Interrupt Enable (SPRIE = 1, SPE = 1)	The SPI receiver interrupt enable bit (SPRIE) enables the SPI receiver full (SPRF) bit or RFWM to generate receiver interrupt requests. The SPRF is set every time data transfers from the shift register to the SPI Data Receive register and there is no more room available in the RX queue to receive new data. The clearing mechanism for the SPRF flag is to read the SPI Data Receive register.
OVRF (Overflow)	SPI Receiver/Error Interrupt Enable (ERRIE = 1)	The error interrupt enable bit (ERRIE) enables both the MODF and OVRF bits to generate a receiver/error interrupt request.
MODF (Mode Fault)	SPI Receiver/Error Interrupt Enable (ERRIE = 1)	The mode fault enable bit (MODEFEN) enables the mode fault (MODF) bit to be set. The MODF bit allows the receiver/error interrupt request regardless of the state of the SPE bit as long as the interrupt enable (ERRIE) is set. The mode fault enable bit (MODFEN) can prevent the MODF flag from being set, so that only the OVRF bit is enabled by the ERRIE bit to generate receiver/error device interrupt requests.

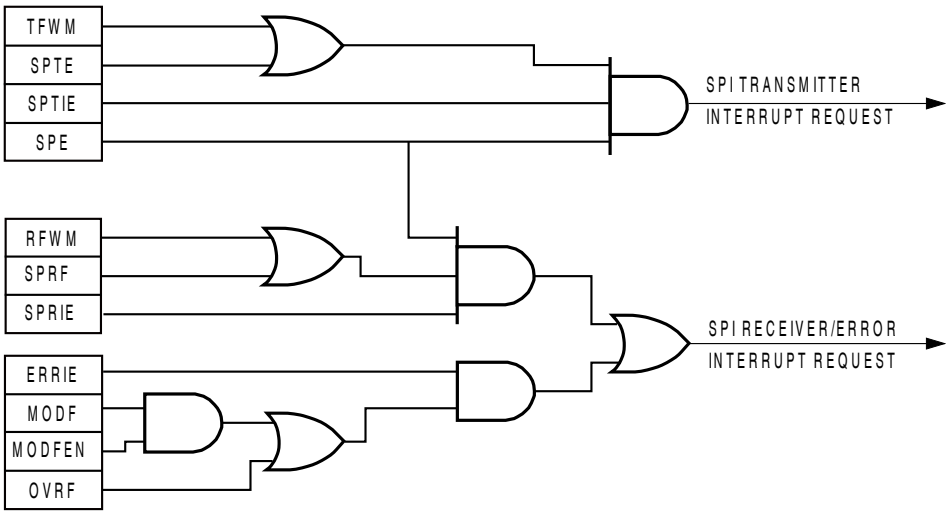


Figure 7-20. SPI Interrupt Request Generation

Chapter 8
Operational Phases and Modes of Operation (OP_PHASES)

8.1 Introduction

The FXLC95000CL device supports various clocking and frame schemes, which in turn facilitates making a "frame-based" software scheduler. The software scheduler provides highly configurable, application-dependent sensor sampling and power management.

8.1.1 Definitions

This section introduces and defines various names and abbreviations that are used in FXLC95000CL documentation.

Frame Rate (FR)	This is the basic unit of time from which all other events are timed.
Output Data Rate (ODR)	The rate at which the FXLC95000CL provides conversion data to the user for a given quantity. This will be SDR/OSR.
Over-Sample Ratio (OSR)	The FXLC95000CL can support on-chip filtering of sensor data. The over-sample ratio specifies how many sample frames are required to support a specified output data rate using a desired filtering algorithm.
Φ_A	Analog phase - All analog (C2V and ADC) processing occurs in this phase. Depending on configuration via registers, the analog subsystem may have processed samples for three dimensions of acceleration and a single auxiliary parameter, during each Φ_A interval. The auxiliary parameters available include temperature and the external ADC inputs. The CPU and associated peripherals are normally "quiet" during this mode.
Φ_D	Digital phase - The CPU and peripherals are active, analog in low-power state. Digital filtering and processing of the converted ADC values occur in this phase. The length of this phase will vary depending upon the CPU load. It cannot exceed (t_F - t_A) for sample frames.
Φ_I	Inactive or Idle phase - Most of the device is powered down for minimal power consumption. This phase is of variable length (t_F - t_A - t_D), where t_F is fixed, t_A is determined by the analog front end (AFE) and t_D varies depending on CPU loading.
Sample Frame	Sample frames correspond, one-to-one, for each "sample" of data.

Table continues on the next page...

Sample Data Rate (SDR)	The rate at which the FXLC95000CL requires raw conversion data from its sensors and converters. If the device is configured for additional over-sampling, this may be some integer times the output data rate or ODR. One sample frame = SDR^{-1} seconds.
t_A	Length of Φ_A
t_D	Length of Φ_D
t_I	Length of Φ_I - The idle phase. This may approach zero, depending on CPU loading.
t_F	Frame interval. This is the reciprocal of the Frame Rate (1/FR)

Additional fixed timing parameters that are used occasionally include:

$F_{osc\text{-}high}$	The high-speed frequency of the on-chip oscillator. This is nominally 16 MHz.
$F_{osc\text{-}low}$	The low-speed frequency of the on-chip oscillator. This is nominally $F_{osc\text{-}high}/256$.
$P_{osc\text{-}high}$	The length of time required for one cycle of the oscillator clock in high-speed mode (= $1/F_{osc\text{-}high}$).
$P_{osc\text{-}low}$	The length of time required for one cycle of the oscillator clock in low-speed mode (= $1/F_{osc\text{-}low}$ = $256/F_{osc\text{-}high}$).

8.1.2 Modes of operation

In addition to RUN, HALT, RESET and STOP modes, FXLC95000CL adds additional controls to STOP mode, effectively creating 3 modes where 1 mode existed previously. Note that the application will mostly use RUN and STOP modes

The FXLC95000CL-supported set of modes then becomes:

RUN	The CPU executes instructions in RUN mode. RUN mode can be further subdivided into User and Supervisor modes.
HALT	Version 1 ColdFire Core HALT/DEBUG mode.
RESET	Reset is asserted; circuits are in a default state. RESET can be divided into several phases of operation. (See System Integration Module.)
$STOP_{FC}$	STOP - Clock in Fast Mode. $STOP_{FC}$ is nominally used for Φ_A. (See Overview.)
$STOP_{SC}$	STOP - Clock in Low-Speed Mode. $STOP_{SC}$ is nominally used for Φ_I. (See Overview.)
$STOP_{NC}$	STOP - No clocks are present; all clocks are disabled. $STOP_{NC}$ mode is nominally used for the SLEEP phase.

8.2 Frame-based sampling

The FXLC95000CL is designed to facilitate a "frame-based" software scheduler. The sensor is either taking samples, converting sample data, or idle; these activities have to be scheduled, to get the best performance using the least amount of power. Output data rates, sample data rates, and frame rates will have to be considered.

Analog sensor conversions are best executed when the CPU is quiet, and there may be times when both AFE and CPU are dormant. The FXLC95000CL includes hardware mechanisms to make it easy to schedule these different functions.

8.2.1 Overview

The FXLC95000CL can be programmed to take a continuous sequence of evenly spaced samples over time. This section specifies the terminology for timing and phases. Figure 8-1, Figure 8-2 and Figure 8-3 illustrate a number of these terms that will be subsequently defined.

Timing is defined using two types of frames: sample and non-sample.

- Sample frames include an analog phase in which sensor outputs are sampled.
- Non-sample frames simply omit the analog phase.

Output Data Rate (ODR), Sample Data Rate (SDR), and Frame Rate (FR) are three important parameters that will be used throughout the following text.

$ODR \leq SDR \leq FR$

The ODR specifies the rate at which an application reads sensor data from the device. The actual SDR is ODR x OSR, where the integer OSR (Over Sample Ratio) is typically in the range of 1 to 4. As a result, several sample frames might be required to support a single sensor reading by the application.

Additionally, non-sample frames, may be intermixed with sample frames.

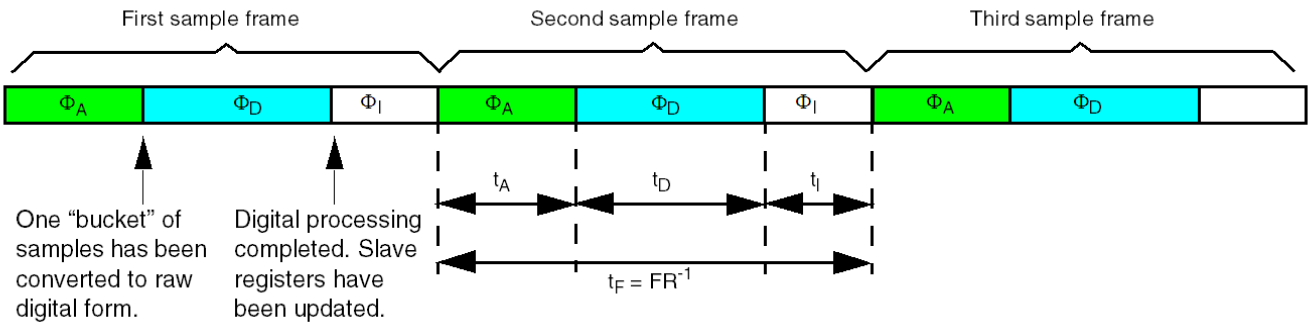


Figure 8-1. Sample frame timing

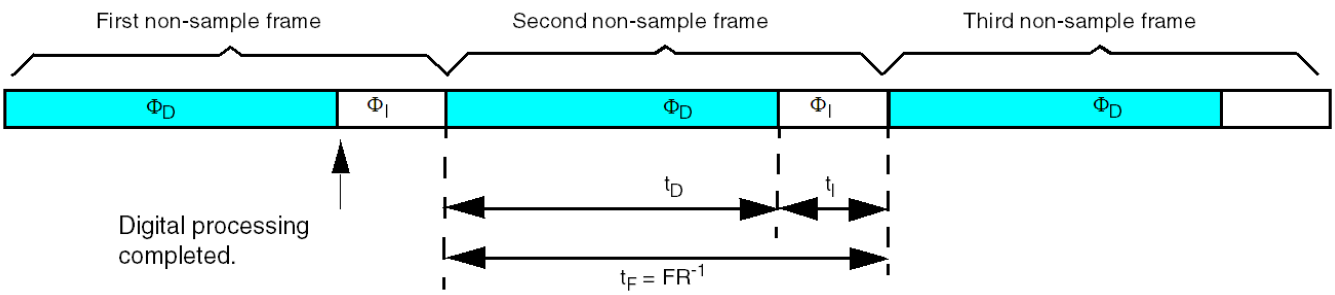


Figure 8-2. Non-sample frame timing

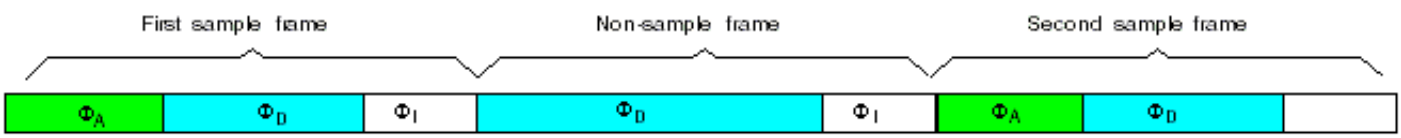


Figure 8-3. Mixed frame timing

8.2.2 Phase triggers

For the FXLC9000CL device, Figure 9-1 shows some of the major interactions between modules in this device:

- The Start-of-frame signal is generated by the frame interval counter within the CLKGEN module.
- SIM hardware is responsible for generating phase triggers for Φ_A and Φ_D.
- The End Φ_A signal is generated by the AFE.
- In sample frames, Start Φ_A results from the Start-of-frame from the CLKGEN module. In this case, Start Φ_D results from the End Φ_A signal.

- For non-sample frames, the Start-of-frame results in Start Φ_D.
- Φ_A Started and Φ_D started signals (not shown) can be slightly delayed from Start Φ_A and Start Φ_D. In the event that the system clock is switching from off (or low speed) to high speed, these signals are not asserted until the oscillator actually switches. The difference in the two sets of triggers is any latency associated with interrupt assertion and/or CLKGEN-mode switching.

Figure 8-4 illustrates sequencing of the Start Φ_A and Start Φ_D hardware triggers. Figure 8-5 shows that a STOP instruction (with STOPCR[SC] = 1) is required to transition into the idle phase. (See the SIM register map.)

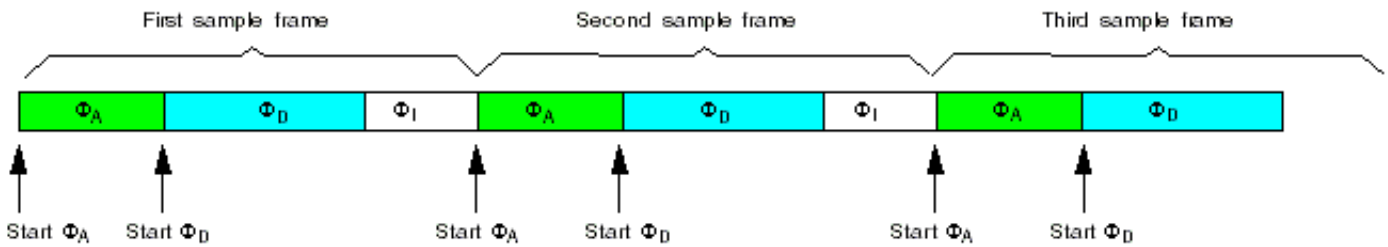


Figure 8-4. Phase triggers required in hardware

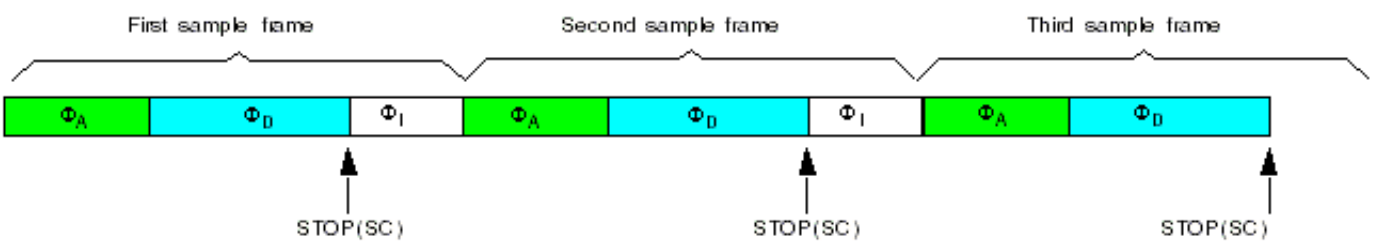


Figure 8-5. Phase triggers required in software

In summary:

Start-of-frame	Initiates Start Φ_A or Start Φ_D, depending on whether the frame is a sample frame or not.
Start Φ_A	Signal to initiate Φ_A.
End Φ_A	Is generated by the AFE and indicates the analog phase has been completed.
Start Φ_D	Signal to initiate Φ_D. This signal results from either Start-of-frame or End Φ_A.
Φ_A started	Φ_A has been initiated and the clock is in high-speed mode.
Φ_D started	Φ_D has been initiated and the clock is in high-speed mode.

8.3 Clock operation as a function of mode/phase

To enable application-specific power savings and to capture data at various rates, FXLC95000CL provides mechanisms to operate the clock differently, depending on the mode or phase.

Figure 8-6 illustrates the nominal phases of clock operation for this device.

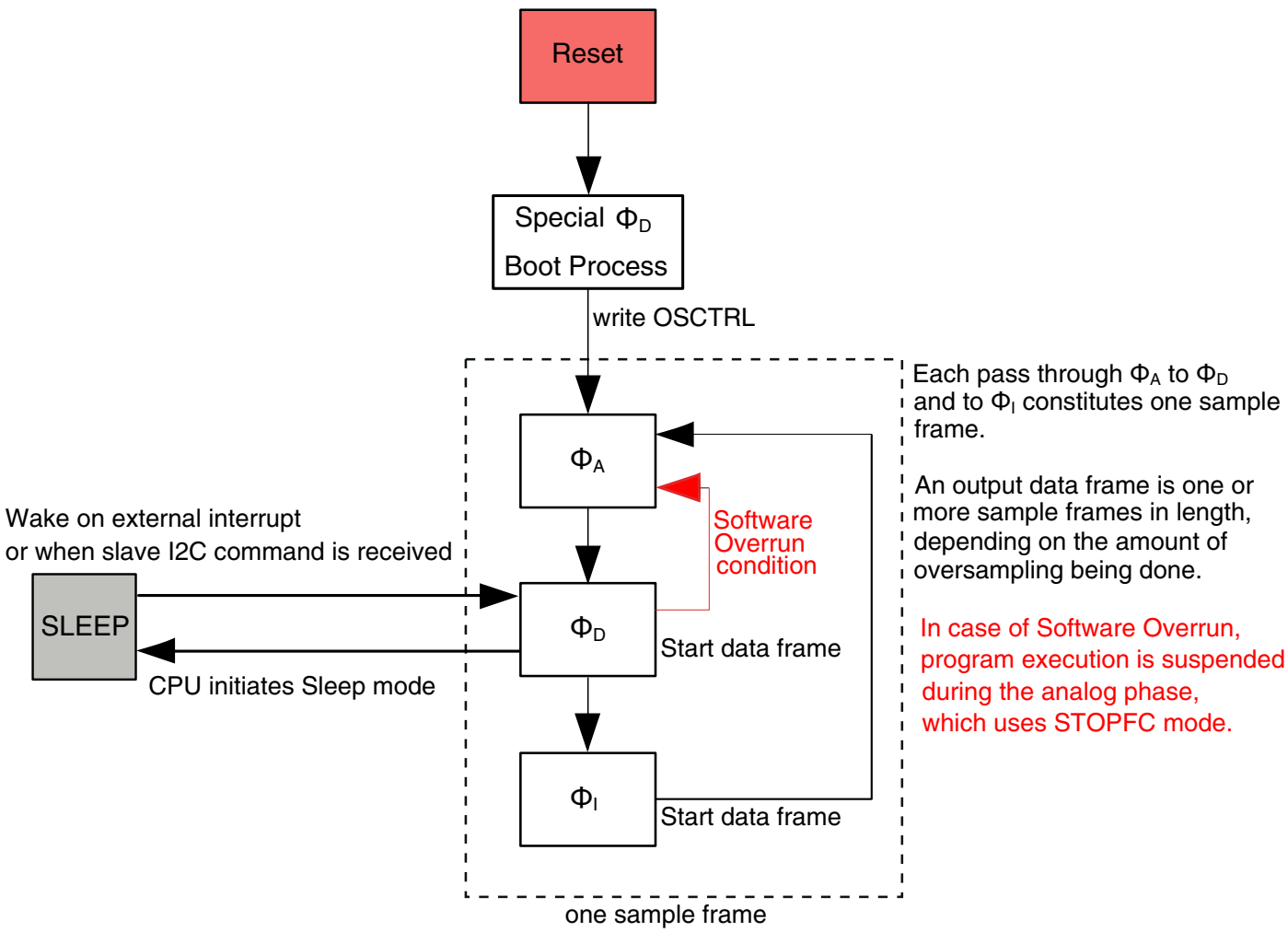


Figure 8-6. Operational phases

The values of Φ_A, Φ_D and Φ_I are discussed in Frame-based sampling.

The reset operation is described in Reset generation.

The sleep phase is defined as the device oscillator being off and all circuitry in its lowest power state.

There is a strong software component to the application phases diagrammed here. They may be rearranged from time to time depending on the tasks assigned to the sensor.

The phases shown earlier have distinct characteristics with regard to clock operation. These are outlined in Table 8-1. The operation of clock-gating registers (PCERUNx, PCESSCx and PCESFCx) in the SIM do not change as a result of debug operation, only the oscillator operation.

Table 8-1. Clock operation per phase

Phase	CPU and Standard Peripherals		Analog Front End		Slave I^2C
	Normal	Debug	Normal	Debug[1]	
Reset	High Speed				Not applicable. The I^2C is externally clocked.
Boot and Φ_D (Run Mode)	High Speed		OFF	High Speed	
Φ_A ($STOP_{FC}$)	OFF	High Speed			
Φ_I ($STOP_{SC}$)	OFF, oscillator in Low-Speed Mode	High Speed	OFF, oscillator in Low-Speed Mode	High Speed	
SLEEP ($STOP_{NC}$)	Oscillator in shutdown	High Speed	Oscillator in shutdown	High Speed	

1. The ENBDM bit in the Version 1 ColdFire Extended Configuration/Status Register (XCSR) is set to "1" to enable BDM communications. The CPU is clocked even during STOP modes. Frequency-hopping is disabled in Debug mode, as BDM communications require a constant clock rate for proper operation.

8.4 Power control modes of operation

The Version 1 ColdFire architecture incorporates several modes of operation. These include Reset, Run, Stop and Halt (debug). For the 6 modes of operation (not including Halt), there are 9 mode transitions to consider.

Φ_A, Φ_I and Sleep phases in Figure 8-6 are all mapped into the ColdFire STOP mode on this device. The CPU has only a single view of STOP operation, but at the device level, additional levels of distinction have been added:

$STOP_{FC}$	STOP - Clock in Fast Mode. Nominally used for Φ_A.
$STOP_{SC}$	STOP - Clock in Low-Speed Mode. Nominally used for Φ_I.
$STOP_{NC}$	STOP - All clocks disabled. Nominally used for the SLEEP phase.

Boot and Φ_D are functionally identical and map into the Run mode. Figure 8-7 adds HALT mode to the set and remaps the collection as a full-state transition diagram, including debug modes. Table 8-2 summarizes the triggers that cause transitions from one mode to the next.

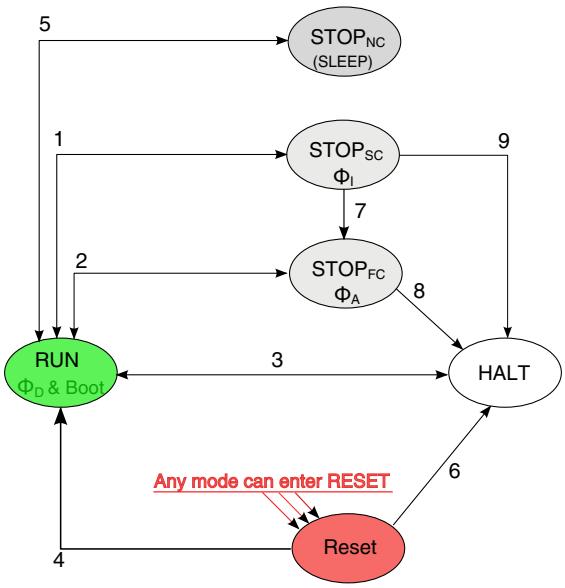


Figure 8-7. Allowable state transitions

Table 8-2. State transitions

Transition #	From	To	Trigger
1	RUN	$STOP_{SC}$	XCSR[ENBDM] = 0, STOPCR[SC] = 1; followed by STOP instruction
	$STOP_{SC}$	RUN	Any interrupt
2	RUN	$STOP_{FC}$	STOPCR[FC] = 1, followed by STOP instruction; OR XCSR[ENBDM] = 1, followed by STOP instruction ($STOP_{SC}$ and $STOP_{NC}$ are emulated by $STOP_{FC}$ in debug mode.)
	$STOP_{FC}$	RUN	Any interrupt
3	RUN	HALT	When a BACKGROUND command is received through the BKGD/MS pin OR when a HALT instruction is executed OR when encountering a BDM breakpoint.
	HALT	RUN	GO instruction issued via BDM
4	RESET	RUN	Deassert all reset sources. Internal deassert is subject to timing sequences outlined in Reset generation.
5	RUN	$STOP_{NC}$	XCSR[ENBDM] = 0, STOPCR[NC] = 1, followed by STOP instruction
	$STOP_{NC}$	RUN	IRQ or Slave Port interrupt
6	RESET	HALT	BDM = 0 during POR (device must be unsecure)
7	$STOP_{SC}$	$STOP_{FC}$	Start of frame signal with STOPCR[A_EN] = 1
8	$STOP_{FC}$	HALT	When a BACKGROUND command is received through the BKGD/MS pin (XCSR[ENBDM] must equal one)
9	$STOP_{SC}$	HALT	In debug mode, $STOP_{SC}$ is emulated by $STOP_{FC}$.

Chapter 9
System Integration Module (SIM)

9.1 System Integration Module (SIM) overview

The System Integration Module (SIM) provides a central mechanism for managing: **Reset generation**, **Mode control**, **Oscillator control**, **Clock gating**

Figure 9-1 illustrates some of the major interactions with other on-chip components. These will be discussed in more detail in the sections that follow.

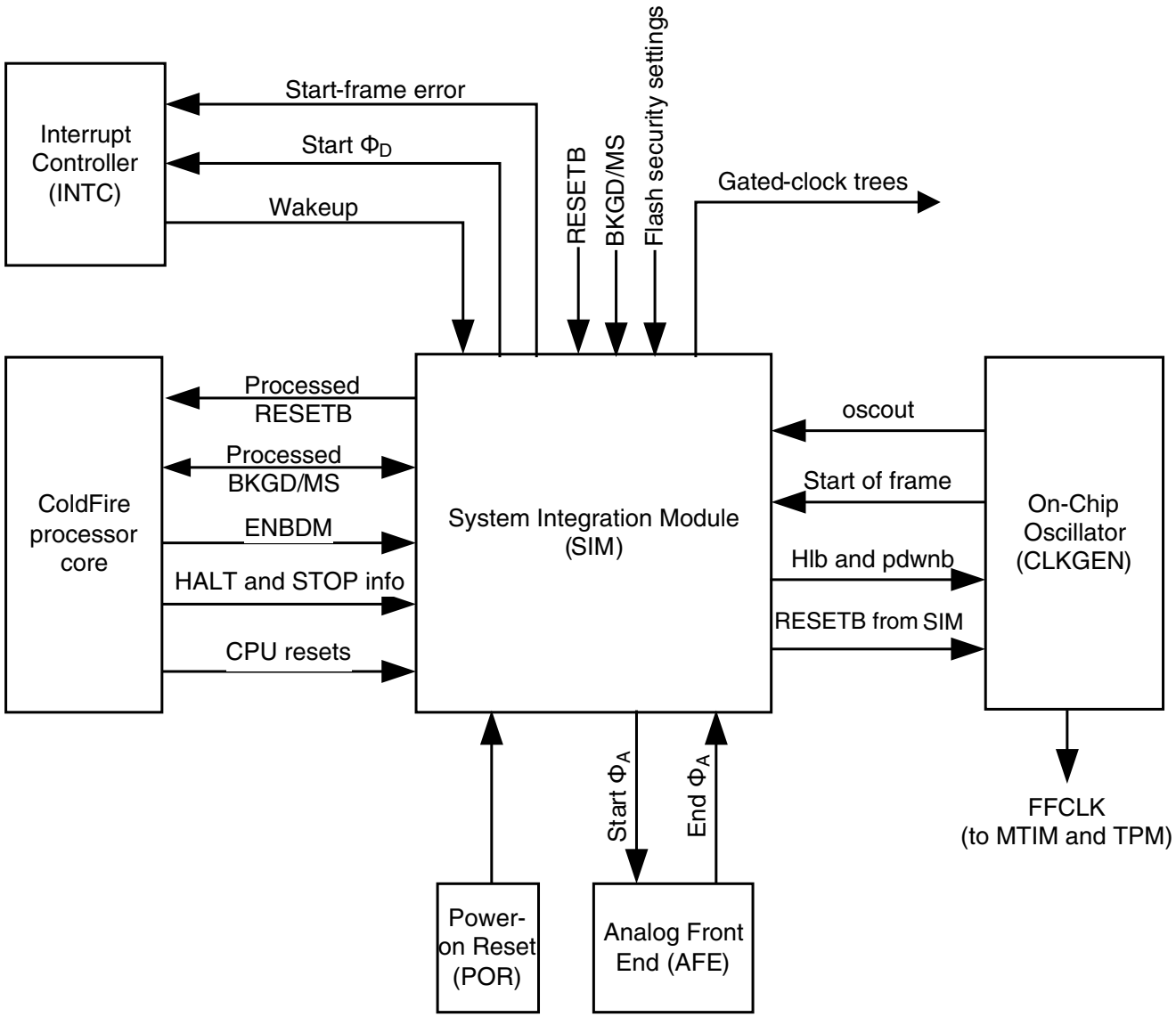


Figure 9-1. Major SIM interactions

9.2 Reset generation

To handle exception conditions or to revert to a known state, resets can be generated while FXLC95000CL is operational.

9.2.1 Reset sources

There are 6 sources that may cause the device to reset:

POR	Active low Power-On-Reset internally generated whenever V_{DDA} < POR brownout threshold.
RESETB	The device can be reset by pulling the external RESETB low.
BDM resetb	The device can be reset via the background debug port.
Software resetb	The CPU can be explicitly shut down by writing a 1b to the software reset bit of the SIM Control Register (RSCR[ASR]).
ILOP resetb	The CPU may request a reset in the event of an illegal operation.
ILAD resetb	The CPU may request a reset in the event of an attempt to access an illegal address.

The ILOP and ILAD reset outputs from the Version 1 ColdFire CPU are collectively shown as "CPU resets" in Figure 9-2. Alternatively, it is also possible to configure the CPU to simply issue an illegal op code or illegal address exception by setting the instruction-related, reset-disable bit in the CPU Configuration Register (CPUCR[IRD]).

9.2.2 Reset outputs

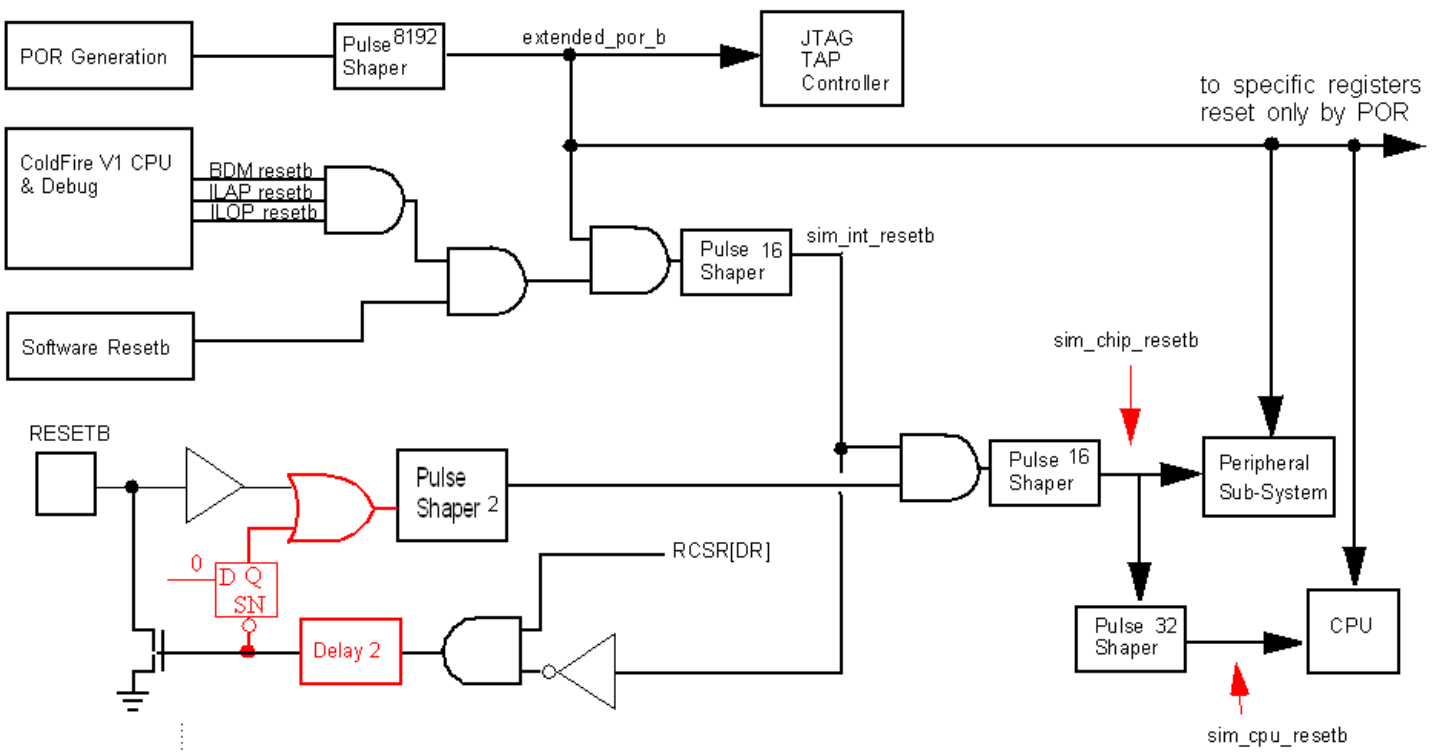


Figure 9-2. Reset generation (functional block diagram)

Figure 9-2 illustrates how the 6 reset sources outlined in the previous section are processed to generate 3 different reset domains. The "Pulse Shaper" blocks have the following characteristics:

- A logic zero on the input is transmitted immediately to the output.
- A logic one on the input is only transmitted to the output on a positive edge of the clock-*N* clock periods after the input deassertion. (*N* is specified as a number in the upper right corner of each block.)

These functions ensure that internal resets assert asynchronously and deassert synchronously. They also guarantee that the internal resets are stable for long enough to propagate properly throughout the system.

The RESETB pin is an open-drain, bidirectional function and must be connected to an external pullup resistor. RESETB is pulled low for a minimum of 16 clock cycles in response to any internally generated reset event when the RCSR[DR] bit is set. The external RESETB input is only sampled when the internal CPU reset has been deasserted for two cycles or longer.

Internal reset domains are the following:

extended_por_b	This signal is a minimum of 8192 cycles long and is initiated by the internal, power-on-reset circuitry. It feeds the specialized registers which are reset only on power-up.
sim_chip_resetb	This is the general chip reset used to place on-chip logic into its default state.
sim_cpu_resetb	The CPU is the last block in the digital domain to be freed from reset. This occurs 32 cycles after deassertion of sim_chip_resetb. The 32 cycles are required to fetch the device security setting from on-chip flash memory before allowing the CPU to boot.

Figure 9-3 shows an idealized start-up sequence in which the raw POR signal starts immediately at time zero. The oscillator starts some time after desertion of the raw POR. The extended_por_b signal is extended a minimum of 8192 cycles after that.

NOTE

For the selected boot from flash to occur, the RESETB pin must be cleared (i.e., reach a high level) before the end of the 8192-cycle extended period (about 500 µs), or the device will boot to ROM. This bounds the time constant of any external filtering circuit that may be added for EMC and noise immunity.

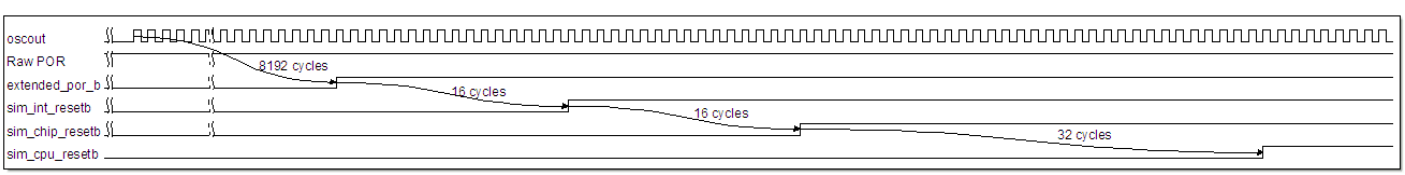


Figure 9-3. Power-on-reset sequence

Figure 9-4 shows a similar sequence, but in this case the reset was initiated by one of the on-chip synchronous sources. Note how even a short source assertion immediately propagated through the tree, but followed the same, well-defined deassertion sequence shown in the previous figure.

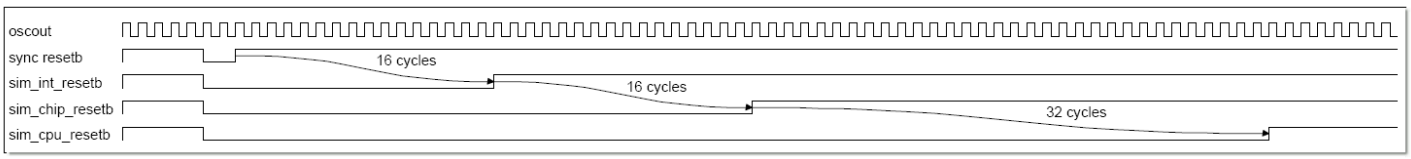


Figure 9-4. Synchronous reset sequence

As depicted in Figure 9-2, the RESETB pin is an open-drain, bidirectional function. At power-up, it is configured strictly as an input pin, but can be programmed to become bidirectional afterwards. Output drive capability is enabled by setting the SIM_RSCR[DR] field to 1b in the SIM Reset Control and Status Register. This will result in RESETB being pulled low for a minimum of 16 clock cycles in response to any internally generated reset event. Using this feature, the FXLC95000CL can reset external devices whenever it is reset for any purpose other than power-on-reset.

9.3 Operating mode control

The STOP Control and Status Register (SIM_STOPCR) controls STOP-mode operation. The CPU controls the various mechanisms to enter BDM HALT mode.

9.3.1 STOP mode control and operation

"Modes of operation" discusses how the various phases of operation are mapped into the Version 1 ColdFire CPU's operating modes. The STOP Control and Status Register (SIM_STOPCR) can be used to control STOP-mode operation. At any point in time, the software must select one of four choices that determine operation for the next STOP command:

$STOP_{FC}$	STOP with oscillator in high-speed (fast) mode
$STOP_{SC}$	STOP with oscillator in slow-speed mode
$STOP_{NC}$	STOP with oscillator completely off
STOP disabled	Stop disabled. In this mode, a STOP instruction will either result in a system reset or an exception.

These choices are mutually exclusive at any one point in time, but the value of the SIM_STOPSCR register can be changed by the CPU as desired. STOP mode is exited via RESET or any active interrupt.

There are separate peripheral clock enable registers for RUN, $STOP_{FC}$ and $STOP_{SC}$. These registers enable the user to specify which peripheral clocks are running (or not) for each of these three modes. CPU clocks are normally disabled when in any STOP mode. The exception to this rule is when debug operation is enabled (XCSR[ENBDM] = 1). For more information on XCSR, see Table 22-2.

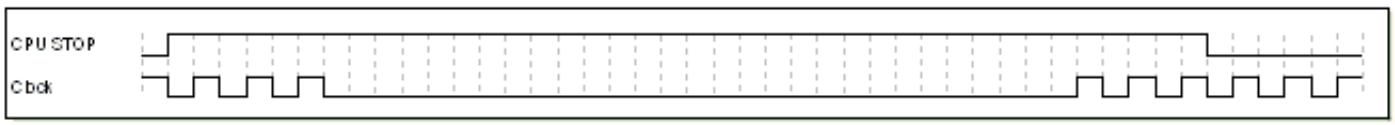


Figure 9-5. High speed clock operation extends into STOP region

Stop modes are initiated by a STOP instruction executed by the CPU, which then signals the SIM to begin gating clocks as appropriate and/or change oscillator frequency. When switching from RUN mode into any STOP mode, high-speed operation is continued for a minimum of three clock cycles into STOP and may resume up to three cycles prior to exiting STOP. The SIM is responsible for "re-shaping" the STOP request signal from the CPU such that it matches operation shown in Figure 9-5.

Note that interrupt operation may be limited on a module-by-module basis, depending on clock availability and the asynchronous features for each module.

9.3.2 DEBUG mode control

The CPU can enter BDM HALT mode through any of the following mechanisms:

1. BKGD = 0b during POR
2. BKGD = 0b during BDM reset
3. CSR2[BFHBR] = 1b during BDM reset
4. Illegal op code reset and CSR2[IOPHR] = 1b
5. Illegal address reset and CSR2[IADHR] = 1b
6. Issue BACKGROUND command via BDM interface
7. HALT instruction
8. BDM breakpoint
9. ColdFire fault-on-fault

Of these, only Method 1 is guaranteed to work under all circumstances (except when the device is secured). Methods 1 through 5 are partially managed via the System Integration Module.

Version 1 ColdFire devices support a bidirectional, one-wire background debug port (BKGD) that is commonly multiplexed with the mode-select function (MS, which is active during the reset sequence). These two signals are demultiplexed shortly after being routed on chip, processed separately, and recombined, prior to being accessed by the CPU debug module.

9.4 Oscillator control

General oscillator control is shared between the CLKGEN module and the SIM. The ColdFire core receives a general CPU clock gated by the SIM in STOP mode, and a dedicated clock for serial communication gated using an enable signal from the CPU.

9.4.1 General

Because the frequency of operation is coupled with the mode of operation, oscillator control is shared between the CLKGEN module (discussed in On-Chip Oscillator (CLKGEN)) and the SIM. The SIM supplies the power-down control and speed control. All other controls are managed locally by the CLKGEN module.

Speed and power-down choices are intrinsic in the choices of operating mode (RUN or STOP). Operation in STOP modes is affected by the settings specified in the STOP Control and Status Register.

9.4.2 CPU

The ColdFire core receives two clocks from the System Integration Module (SIM).

- A general CPU clock. This clock is gated off by the SIM in STOP mode.
- A dedicated clock used for serial communication on the BDM port. It is gated using an enable signal from the CPU.

Both clocks on the FXLC95000CL are active during reset. Both are derived from the CLKGEN module oscout signal.

9.5 Clock gating

Many applications may not need all the capabilities the device offers. Unneeded functions may have their clocks disabled to save power. Subsystems that are unclocked during RUN mode cannot be read/written by the CPU.

Many applications may not need all the capabilities the device offers. Unneeded functions may have their clocks disabled, to save power.

Clock settings can be separately controlled for RUN, $STOP_{FC}$, and $STOP_{SC}$ modes. This means that mode transitions naturally enable/disable various systems automatically. The programmer is not required to explicitly enable/disable them each time modes are switched.

The SIM is always clocked in RESET, RUN, and HALT modes. It is not clocked in any of the STOP modes. This implies that control paths for the start of frame signal are combinational in nature, as the SIM must be able to generate either "start Φ_A" or "start Φ_D"signal when "start frame" is asserted.

ALL clocks are disabled in $STOP_{NC}$.

Subsystems that are unclocked during RUN mode cannot be read/written by the CPU. Writes are ignored and reads return unknown quantities.

Table 9-1 shows that the slave I^2C and external interrupt pin are the only two peripheral functions capable of issuing interrupts for wake-up when not clocked.

Table 9-1. Unclocked interrupt support

Sub-system	Unclocked interrupts?
Slave Port	Yes - This module operates independently of the CLKGEN module.
Master I^2C	No
16-bit Modulo Timer	No
Two-channel Timer/PWM	No
IRQ (external interrupt)	Yes
AFE	No
SIM	Start Φ_D

9.6 Register settings for I²C slave port

Table 9-2. Recommended register field settings based on location of I²C port 1 (slave port)

Register field	SCL1 on RGPIO14 (Pin 1)	SCL1 on RGPIO2 (Pin 11)	SDA1 on RGPIO15 (Pin 2)	SDA1 on RGPIO3 (Pin 12)
PMCR1[A2]	00 or 10	01	Don't care	Don't care
PMCR1[A3]	Don't care	Don't care	00 or 10	01
RGPIO_ENB[14]	0	1	Don't care	Don't care
RGPIO_ENB[15]	Don't care	Don't care	0	1
RGPIO_ENB[2]	Don't care	0	Don't care	Don't care
RGPIO_ENB[3]	Don't care	Don't care	Don't care	0

9.7 SIM memory map/register definitions

SIM memory map

Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
FFFF_8060	STOP Control and Status Register (SIM_STOPCR)	8	R/W	00h	9.7.1/176
FFFF_8061	Frame Control and Status Register (SIM_FCSR)	8	R/W	20h	9.7.2/177
FFFF_8062	Reset Status and Control Register (SIM_RSCR)	8	R/W	01h	9.7.3/178
FFFF_8064	Peripheral Clock Enable Register 0 for STOPFC mode (SIM_PCESFC0)	8	R/W	7Fh	9.7.4/180
FFFF_8065	Peripheral Clock Enable Register 1 for STOPFC mode (SIM_PCESFC1)	8	R/W	07h	9.7.5/181
FFFF_8066	Peripheral Clock Enable Register 0 for STOPSC mode (SIM_PCESSC0)	8	R/W	7Fh	9.7.6/182
FFFF_8067	Peripheral Clock Enable Register 1 for STOPSC mode (SIM_PCESSC1)	8	R/W	07h	9.7.7/183
FFFF_8068	Peripheral Clock Enable Register 0 for RUN mode (SIM_PCERUN0)	8	R/W	7Fh	9.7.8/184
FFFF_8069	Peripheral Clock Enable Register 1 for RUN mode (SIM_PCERUN1)	8	R/W	07h	9.7.9/185
FFFF_806A	Pin Mux Control Register0 (SIM_PMCR0)	8	R/W	00h	9.7.10/186
FFFF_806B	Pin Mux Control Register1 (SIM_PMCR1)	8	R/W	00h	9.7.11/187
FFFF_806C	Pin Mux Control Register2 (SIM_PMCR2)	8	R/W	00h	9.7.12/188

9.7.1 STOP Control and Status Register (SIM_STOPCR)

NOTE

Before enabling automatic STOPSC-to-STOPFC transitions via the SIM STOPCR[SCtoFC] bit, make sure to set the CK_OSCTRL[FLE] field in the CLKGEN module to a valid value. Failure to do so can result in higher than desired IDD stop mode currents.

Address: FFFF_8060h base + 0h offset = FFFF_8060h

Bit	7	6	5	4	3	2	1	0
Read	0			SIM_CLK_EN	FC	SC	NC	SCtoFC
Write								
Reset	0	0	0	0	0	0	0	0

SIM_STOPCR field descriptions

Field	Description
7–5 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
4 SIM_CLK_EN	SIM Clock Enable This bit overrides the clock gating that otherwise occurs within the SIM during STOP mode. This capability is not used during normal operation so this bit should be left at 0 which is the default state out of reset. 0 The SIM will enter a non-clocked, low-power, state during STOP modes 1 Internal SIM clocking remains active during STOP modes
3 FC	STOP Mode Enable for STOP With Fast Clock The FC, SC and NC bits are mutually exclusive. They control the mode of operation to be initiated by the next STOP instruction. A maximum of one of the three can be asserted at any time. If none of them are enabled, STOP is considered an illegal instruction. Instead of entering one of the STOP modes, the MCU will initiate an illegal opcode reset if CPUCR[IRD] is cleared. If CPUCR[IRD] is set, an illegal instruction exception is initiated. (See ColdFire Core CPU Configuration Register for details.) If the CPU attempts to write more than one of FC, SC or NC, all three will be cleared and, again, STOP will be considered an illegal instruction. 0 STOP with Fast Clock is NOT enabled. 1 The next STOP instruction will result in the CPU entering STOP with the oscillator in high-speed mode
2 SC	STOP Mode Enable for STOP With Slow Clock The FC, SC and NC bits are mutually exclusive. They control the mode of operation to be initiated by the next STOP instruction. A maximum of one of the three can be asserted at any time. If none of them are enabled, STOP is considered an illegal instruction. Instead of entering one of the STOP modes, the MCU will initiate an illegal opcode reset if CPUCR[IRD] is cleared. If CPUCR[IRD] is set, an illegal instruction exception is initiated. (See ColdFire Core CPU Configuration Register for details.) If the CPU attempts to write more than one of FC, SC or NC, all three will be cleared and, again, STOP will be considered an illegal instruction.

Table continues on the next page...

SIM_STOPCR field descriptions (continued)

Field	Description
	0 STOP with Slow Clock is NOT enabled. 1 The next STOP instruction will result in the CPU entering STOP, with the oscillator in low-speed mode.
1 NC	STOP Mode Enable for STOP With No Clock The FC, SC and NC bits are mutually exclusive. They control the mode of operation to be initiated by the next STOP instruction. A maximum of one of the three can be asserted at any time. If none of them are enabled, STOP is considered an illegal instruction. Instead of entering one of the STOP modes, the MCU will initiate an illegal opcode reset if CPUCR[IRD] is cleared. If CPUCR[IRD] is set, an illegal instruction exception is initiated.(See ColdFire Core CPU Configuration Register for details.) If the CPU attempts to write more than one of FC, SC or NC, all three will be cleared and, again, STOP will be considered an illegal instruction. 0 STOP with No Clock is NOT enabled. 1 The next STOP instruction will result in the CPU entering STOP, with the oscillator disabled. The device will be in deep-sleep mode. In this mode, the device can be awakened only by asynchronous interrupts (one of which can be initiated via the slave I^2C interface) or a reset assertion.
0 SCtoFC	Slow Clock to Fast Clock STOP Transition Enabled The device can be programmed to transition from $STOP_{SC}$ to $STOP_{FC}$ when a "Start Sample Frame" signal is asserted. Simply program SCtoFC to 1. This bit allows the CPU to initiate IDLE mode with a $STOP_{SC}$ transition. That state will automatically transition to $STOP_{FC}$ when the AFE needs to be started up for the next frame. This operation occurs without CPU intervention if this bit is set. 0 Automatic transition from $STOP_{SC}$ to $STOP_{FC}$ is not enabled. 1 The "Start Sample Frame" signal will cause the device to transition from $STOP_{SC}$ to $STOP_{FC}$ should it occur while the device is parked in $STOP_{SC}$. It has no affect otherwise.

9.7.2 Frame Control and Status Register (SIM_FCSR)

Address: FFFF_8060h base + 1h offset = FFFF_8061h

Bit	7	6	5	4	3	2	1	0
Read	0		A_EN	SFEIE		FE	SFDIE	SFD
Write								
Reset	0	0	1	0	0	0	0	0

SIM_FCSR field descriptions

Field	Description
7–6 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
5 A_EN	Φ_A Enable Individual frames can be programmed to start with Φ_A or Φ_D based on the state of this bit. Simply program the desired value for the next frame before entering Φ. If all frames include an analog sample phase, this bit can be left at 1. 0 The next frame will skip Φ_A and proceed directly to Φ_D. 1 The next frame start with Φ_A, followed by Φ_D.

Table continues on the next page...

SIM_FCSR field descriptions (continued)

Field	Description
4–3 SFEIE	Start Frame Error Interrupt Enable Frames can be sub-divided into Φ_A, Φ_D and Φ_I. Phases sequence from Φ_A (optional) to Φ_D to Φ_I and repeat. The transition from Φ_D to Φ_I is initiated by the CPU via a STOP instruction with STOPCR[SC] or STOPCR[FC] set to one. The transition from Φ_I to Φ_A is normally initiated by the "start frame" signal from the CLKGEN module to the SIM. The SIM then issues the "Start Φ_A" or "Start Φ_D" signal. However, it is possible that errors in the CPU software could result in an overrun of Φ_D into the period normally budgeted for the next frame. This software design error can be trapped by programming FCSR[SFEIE]. When set, a "start frame" signal occurring during RUN mode will set FCSR[FE] and issue a Level-7, non-maskable interrupt. 00 No error checking is performed. 01 Error checking is performed when background debug mode is not enabled (XCSR[ENBDM] = 0), but not during debug mode (XCSR[ENBDM] = 1). 10 RESERVED 11 Error-checking is performed in both normal and debug mode.
2 FE	Frame Error This bit is set when a frame error has been detected. FCSR[SFEIE] must have been set to 01 or 11 for this to occur. 0 No error detected. 1 Frame error detected. Clear this flag by writing a "1" to this location.
1 SFDIE	Start Φ_D Interrupt Enable When this bit is set, an interrupt will be asserted to wake the CPU at the start of Φ_D (SFD will be asserted). Asserted interrupts are cleared by clearing SF. The SIM also outputs a Start Φ_D signal to the PDB regardless of whether or not SFDIE is asserted. 0 Start of Φ_D interrupt is not enabled. 1 Start of Φ_D interrupt is enabled.
0 SFD	Start Frame This bit is set when that start FD signal is asserted. It is cleared by writing a 1 to this location. 0 Start of Φ_D has not been asserted. 1 Start of Φ_D has been asserted. Clear this flag by writing a "1" to this location.

9.7.3 Reset Status and Control Register (SIM_RSCR)

This register includes read-only status flags to indicate the source of the most-recent reset. When a debug host forces reset by setting CSR2[BDFR], none of the status bits in RCSR will be set (RCSR[4:0] = $00). The reset state of these bits depends on what caused the microcontroller to reset. Also included are bits for asserting software reset and controlling operation of the reset pin.

Bits [4:1] of this register are mutually exclusive. This register is asynchronously reset during Power-On-Reset and subsequently is synchronously updated based on the precedence level of reset inputs. Only the most-recent reset source will be indicated if multiple resets occur. If multiple reset sources assert simultaneously, the highest-precedence source will be indicated. The precedence from highest to lowest is:

1. POR
2. PIN
3. BDM (no bits asserted)
4. ILAD
5. ILOP
6. SW

Note that the POR bit must be explicitly cleared by software running on the device. This is required to ensure that power-up routines always execute as desired.

Address: FFFF_8060h base + 2h offset = FFFF_8062h

Bit	7	6	5	4	3	2	1	0
Read	0	DR	0	SW	ILOP	ILAD	PIN	POR
Write			ASR					
Reset	0	0	0	0	0	0	0	1

SIM_RSCR field descriptions

Field	Description
7 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
6 DR	Drive Reset Pin The DR bit is only reset via POR. 0 Do not drive the external reset pin. 1 Internally generated resets (excluding POR) will result in the RESETB pin being pulled low.
5 ASR	Assert Software Reset 0 Do nothing 1 A software initiated reset can be generated by writing "1" to this bit location. The SW bit will be set when exiting this reset sequence. ASR is self-clearing, it should always read as 0.
4 SW	Software Reset 0 Reset not caused by a software reset. 1 Reset initiated by writing ASR to 1
3 ILOP	Illegal Opcode Reset 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. The HALT instruction is considered illegal if the BDM interface is disabled by XCSR[ENBDM] = 0.

Table continues on the next page...

SIM_RSCR field descriptions (continued)

Field	Description
	0 Reset not caused by an illegal opcode. 1 Reset caused by an illegal opcode
2 ILAD	Illegal Address Reset 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, the appropriate processor exception is generated instead of the reset. 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
1 PIN	External Pin Reset The PIN bit is only set when RESETB is externally driven. A low on RESETB during an internal reset while DR = 1 does not impact this bit. A low on RESETB after an internal reset deasserts will impact this bit. 0 Reset was not the result of an external pin reset. 1 Reset was the result of an external pin reset (RESETB=0).
0 POR	Power-On Reset Read 0 Reset was not a result of a power on sequence. Read 1 The last reset was the result of a power on sequence. Write 0 No effect. Write 1 Clear POR bit.

1. This is subject to certain restrictions associated with the DR bit. See details in the description of the PIN bit field.

9.7.4 Peripheral Clock Enable Register 0 for STOPFC mode (SIM_PCESFC0)

The FXLC95000CL contains a number of resources that may not be needed for all applications. The clock control registers can be used to individually program clocks on or off for each of the following modes: $STOP_{SC}$ (PCESSC*x*), $STOP_{FC}$ (PCESFC*x*) and RUN (PCERUN*x*). In $STOP_{NC}$, the oscillator is disabled, so a separate control is not needed for that mode.

Operation of these registers is unaffected by debug mode.

Address: FFFF_8060h base + 4h offset = FFFF_8064h

Bit	7	6	5	4	3	2	1	0
Read	0	T2	T1	T0	IRQ	AFE	PCTRL	FLSH
Write								
Reset	0	1	1	1	1	1	1	1

SIM_PCESFC0 field descriptions

Field	Description
7 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
6 T2	Timer 2 Clock Enable 0 Programmable Delay Block (PDB) is not enabled. 1 The clock to the PDB is enabled for this mode of operation.
5 T1	Timer 1 Clock Enable 0 Timer PWM (TPM) clock is not enabled. 1 The clock to the timer PWM (TPM) is enabled for this mode of operation.
4 T0	Timer 0 Clock Enable 0 Modulo timer clock is not enabled. 1 The clock to the modulo timer is enabled for this mode of operation.
3 IRQ	IRQ Clock Enable 0 IRQ clock is not enabled. The module can issue only asynchronous interrupts (if so programmed) 1 The clock to the IRQ module is enabled. Rising and falling interrupts may be used. The IRQ clock must be enabled to program the interrupt, although it can be disabled afterwards if only level-sensitive interrupts are enabled.
2 AFE	Analog Front End Clock Enable 0 The AFE clock is not enabled. 1 The clock to the AFE is enabled for this mode of operation.
1 PCTRL	Port Control Clock Enable The PCTRL bit affects both PC0 and PC1. 0 The port control clock is not enabled. 1 The clock to the port control module is enabled for this mode of operation.
0 FLSH	Flash Controller Clock Enable 0 The flash controller clock is not enabled. The flash can still be read, but program/erase operations are not possible, nor can the flash controller registers be accessed. 1 The clock to the flash controller is enabled for this mode of operation.

9.7.5 Peripheral Clock Enable Register 1 for STOPFC mode (SIM_PCESFC1)

Address: FFFF_8060h base + 5h offset = FFFF_8065h

Bit	7	6	5	4	3	2	1	0
Read	0					MSPI	MI2C	SLAVE
Write								
Reset	0	0	0	0	0	1	1	1

SIM_PCESFC1 field descriptions

Field	Description
7–3 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
2 MSPI	Master SPI Clock Enable 0 The clock to the master SPI interface is not enabled. The module cannot be used in this mode of operation. 1 The clock to the master SPI interface is enabled for this mode of operation
1 MI2C	Master I^2C Clock Enable 0 The clock to the master I^2C interface is not enabled. The module cannot be used in this mode of operation. 1 The clock to the master I^2C interface is enabled for this mode of operation.
0 SLAVE	Slave Port Clock Enable This bit only controls the clock to the IP-bus interface for the slave port. The slave SPI/I^2C clock is supplied from off-device and the serial port can be used at all times. 0 The clock to the slave port, IP-bus interface is not enabled. Registers in the slave mailbox cannot be accessed by the CPU. 1 The clock to the slave port, IP-bus interface is enabled for this mode of operation.

9.7.6 Peripheral Clock Enable Register 0 for STOPSC mode (SIM_PCESSC0)

The FXLC95000CL contains a number of resources that may not be needed for all applications. The clock control registers can be used to individually program clocks on or off for each of the following modes: RUN (PCERUN*x*), $STOP_{SC}$ (PCESSC*x*) and $STOP_{FC}$ (PCESFC*x*). In $STOP_{NC}$, the oscillator is disabled, so a separate control is not needed for that mode.

Operation of these registers is unaffected by debug mode.

The bit fields for the three sets of registers are identical and described below.

Address: FFFF_8060h base + 6h offset = FFFF_8066h

Bit	7	6	5	4	3	2	1	0
Read	0	T2	T1	T0	IRQ	AFE	PCTRL	FLSH
Write								
Reset	0	1	1	1	1	1	1	1

SIM_PCESSC0 field descriptions

Field	Description
7 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.

Table continues on the next page...

SIM_PCESSC0 field descriptions (continued)

Field	Description
6 T2	Timer 2 Clock Enable 0 Programmable Delay Block (PDB) is not enabled. 1 The clock to the PDB is enabled for this mode of operation.
5 T1	Timer 1 Clock Enable 0 Timer PWM (TPM) clock is not enabled. 1 The clock to the timer PWM (TPM) is enabled for this mode of operation.
4 T0	Timer 0 Clock Enable 0 Modulo timer clock is not enabled. 1 The clock to the modulo timer is enabled for this mode of operation.
3 IRQ	IRQ Clock Enable 0 IRQ clock is not enabled. The module can issue only asynchronous interrupts (if so programmed) 1 The clock to the IRQ module is enabled. Rising and falling interrupts may be used. The IRQ clock must be enabled to program the interrupt, although it can be disabled afterwards if only level-sensitive interrupts are enabled.
2 AFE	Analog Front End Clock Enable 0 The AFE clock is not enabled. 1 The clock to the AFE is enabled for this mode of operation.
1 PCTRL	Port Control Clock Enable The PCTRL bit affects both PC0 and PC1. 0 The port control clock is not enabled. 1 The clock to the port control module is enabled for this mode of operation.
0 FLSH	Flash Controller Clock Enable 0 The flash controller clock is not enabled. The flash can still be read, but program/erase operations are not possible, nor can the flash controller registers be accessed. 1 The clock to the flash controller is enabled for this mode of operation.

9.7.7 Peripheral Clock Enable Register 1 for STOPSC mode (SIM_PCESSC1)

Address: FFFF_8060h base + 7h offset = FFFF_8067h

Bit	7	6	5	4	3	2	1	0
Read	0					MSPI	MI2C	SLAVE
Write								
Reset	0	0	0	0	0	1	1	1

SIM_PCESSC1 field descriptions

Field	Description
7–3 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
2 MSPI	Master SPI Clock Enable 0 The clock to the master SPI interface is not enabled. The module cannot be used in this mode of operation. 1 The clock to the master SPI interface is enabled for this mode of operation
1 MI2C	Master I^2C Clock Enable 0 The clock to the master I^2C interface is not enabled. The module cannot be used in this mode of operation. 1 The clock to the master I^2C interface is enabled for this mode of operation.
0 SLAVE	Slave Port Clock Enable This bit only controls the clock to the IP-bus interface for the slave port. The slave SPI/I^2C clock is supplied from off-device and the serial port can be used at all times. 0 The clock to the slave port, IP-bus interface is not enabled. Registers in the slave mailbox cannot be accessed by the CPU. 1 The clock to the slave port, IP-bus interface is enabled for this mode of operation.

9.7.8 Peripheral Clock Enable Register 0 for RUN mode (SIM_PCERUN0)

The FXLC95000CL contains a number of resources that may not be needed for all applications. The clock control registers can be used to individually program clocks on or off for each of the following modes: RUN (PCERUN*x*), $STOP_{SC}$ (PCESSC*x*) and $STOP_{FC}$ (PCESFC*x*). In $STOP_{NC}$, the oscillator is disabled, so a separate control is not needed for that mode.

Peripheral registers cannot be read by the CPU if their PCERUN*x* bit has not been set to 1.

Operation of these registers is unaffected by debug mode.

The bit fields for the three sets of registers are identical and described below.

Address: FFFF_8060h base + 8h offset = FFFF_8068h

Bit	7	6	5	4	3	2	1	0
Read	0	T2	T1	T0	IRQ	AFE	PCTRL	FLSH
Write								
Reset	0	1	1	1	1	1	1	1

SIM_PCERUN0 field descriptions

Field	Description
7 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
6 T2	Timer 2 Clock Enable 0 Programmable Delay Block (PDB) is not enabled. 1 The clock to the PDB is enabled for this mode of operation.
5 T1	Timer 1 Clock Enable 0 Timer PWM (TPM) clock is not enabled. 1 The clock to the timer PWM (TPM) is enabled for this mode of operation.
4 T0	Timer 0 Clock Enable 0 Modulo timer clock is not enabled. 1 The clock to the modulo timer is enabled for this mode of operation.
3 IRQ	IRQ Clock Enable 0 IRQ clock is not enabled. The module can issue only asynchronous interrupts (if so programmed) 1 The clock to the IRQ module is enabled. Rising and falling interrupts may be used. The IRQ clock must be enabled to program the interrupt, although it can be disabled afterwards if only level-sensitive interrupts are enabled.
2 AFE	Analog Front End Clock Enable 0 The AFE clock is not enabled. 1 The clock to the AFE is enabled for this mode of operation.
1 PCTRL	Port Control Clock Enable The PCTRL bit affects both PC0 and PC1. 0 The port control clock is not enabled. 1 The clock to the port control module is enabled for this mode of operation.
0 FLSH	Flash Controller Clock Enable 0 The flash controller clock is not enabled. The flash can still be read, but program/erase operations are not possible, nor can the flash controller registers be accessed. 1 The clock to the flash controller is enabled for this mode of operation.

9.7.9 Peripheral Clock Enable Register 1 for RUN mode (SIM_PCERUN1)

Address: FFFF_8060h base + 9h offset = FFFF_8069h

Bit	7	6	5	4	3	2	1	0
Read	0					MSPI	MI2C	SLAVE
Write								
Reset	0	0	0	0	0	1	1	1

SIM_PCERUN1 field descriptions

Field	Description
7–3 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
2 MSPI	Master SPI Clock Enable 0 The clock to the master SPI interface is not enabled. The module cannot be used in this mode of operation. 1 The clock to the master SPI interface is enabled for this mode of operation
1 MI2C	Master I^2C Clock Enable 0 The clock to the master I^2C interface is not enabled. The module cannot be used in this mode of operation. 1 The clock to the master I^2C interface is enabled for this mode of operation.
0 SLAVE	Slave Port Clock Enable This bit only controls the clock to the IP-bus interface for the slave port. The slave SPI/I^2C clock is supplied from off-device and the serial port can be used at all times. 0 The clock to the slave port, IP-bus interface is not enabled. Registers in the slave mailbox cannot be accessed by the CPU. 1 The clock to the slave port, IP-bus interface is enabled for this mode of operation.

9.7.10 Pin Mux Control Register0 (SIM_PMCR0)

The Pin Mux Control Registers are used to determine whether a device pin is programmed for Function 1, 2 or 3 as defined in Table 3-1.

A9 through A0 correspond to pins RGPIO Bit 9 through RGPIO Bit 0.

Address: FFFF_8060h base + Ah offset = FFFF_806Ah

Bit	7	6	5	4	3	2	1	0
Read Write	A9	A8	A7		A6		0	A4
Reset	0	0	0	0	0	0	0	0

SIM_PMCR0 field descriptions

Field	Description
7 A9	RGPIO Bit 9 Pin-Function Select 0 Pin function is BKGD/MS. 1 Pin function is RGPIO Bit 9
6 A8	RGPIO Bit 8 Pin-Function Select 0 Pin function is RGPIO Bit 8 1 Pin function is PDB output B
5–4 A7	RGPIO Bit 7 Pin-Function Select

Table continues on the next page...

SIM_PMCR0 field descriptions (continued)

Field	Description
	00 Pin function is RGPIO Bit 7 01 Pin function is AN1 10 Pin Function is TPMCH1. 11 RESERVED
3–2 A6	RGPIO Bit 6 Pin-Function Select 00 Pin function is RGPIO Bit 6 01 Pin function is AN0 10 Pin Function is TPMCH0 11 RESERVED
1 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
0 A4	RGPIO Bit 4 Pin-Function Select When A4 is programmed as an interrupt, the IRQ function must also be enabled by setting IRQSC[IRQPE]. When operated as an interrupt, the pull-up/down enable is controlled by IRQSC[IRQPDD] instead of PT1PE[PE4]. At the same time, if IRQSC[IRQPDD] is enabled and IRQSC[IRQEDG] is 0, a pull-up is used. If IRQSC[IRQEDG] is 1, a pull-down is used. 0 Pin function is RGPIO Bit 4 1 Pin function is INT

9.7.11 Pin Mux Control Register1 (SIM_PMCR1)

-

The next table shows the recommended register field settings, based on the location of the I^2C port 1 (a slave port):

Register Field	SCL1 on RGPIO14 (pin 1)	SCL1 on RGPIO2 (pin 11)	SDA1 on RGPIO15 (pin 2)	SDA1 on RGPIO3 (pin 12)
PMCR1[A2]	00 or 10	01	Don't Care	Don't Care
PMCR1[A3]	Don't Care	Don't Care	00 or 10	01
RGPIO_ENB[14]	0	1	Don't Care	Don't Care
RGPIO_ENB[15]	Don't Care	Don't Care	0	1
RGPIO_ENB[2]	Don't Care	0	Don't Care	Don't Care
RGPIO_ENB[3]	Don't Care	Don't Care	Don't Care	0

Address: FFFF_8060h base + Bh offset = FFFF_806Bh

Bit	7	6	5	4	3	2	1	0
Read Write	A3		A2		A1		A0	
Reset	0	0	0	0	0	0	0	0

SIM_PMCR1 field descriptions

Field	Description
7–6 A3	RGPIO Bit 3 Pin-Function Select 00 Pin function is RGPIO Bit 3 01 Pin function is SDA1 (Master I^2C data) 10 Pin function is SSB (SPI slave select) 11 RESERVED
5–4 A2	RGPIO Bit 2 Pin-Function Select 00 Pin function is RGPIO Bit 2 01 Pin function is SCL1 (Master I^2C clock) 10 Pin function is MISO (SPI data out) 11 RESERVED
3–2 A1	RGPIO Bit 1 Pin-Function Select 00 Pin function is SDA0 (slave I^2C data) 01 Pin function is RGPIO Bit 1 10 Pin function is MOSI (SPI data in) 11 RESERVED
A0	RGPIO Bit 0 Pin-Function Select 00 Pin function is SCL0 (slave I^2C clock) 01 Pin function is RGPIO Bit 0 10 Pin function is SCLK (SPI clock) 11 RESERVED

1. SDA1 cannot be configured to RGPIO15 when this bit pattern is asserted
2. SCL1 cannot be configured to RGPIO14 when this bit pattern is asserted.

9.7.12 Pin Mux Control Register2 (SIM_PMCR2)

Address: FFFF_8060h base + Ch offset = FFFF_806Ch

Bit	7	6	5	4	3	2	1	0
Read Write	Reserved						A5	
Reset	0	0	0	0	0	0	0	0

SIM_PMCR2 field descriptions

Field	Description
7–2 RESERVED	This field is reserved. RESERVED
A5	RGPIO Bit 5 Pin-Function Select 00 Pin function is RGPIO Bit 5 01 Pin function is PDB output A 10 Pin function is INT_O (output interrupt from slave port) 11 RESERVED

Chapter 10
Analog Front End (AFE)

10.1 Analog Front End Module Overview

The Analog Front End (AFE) includes the accelerometer, signal conditioning blocks, off-chip analog inputs, analog multiplexor and an analog-to-digital converter (ADC) and control logic. In short, the AFE is responsible for converting analog sensor measurements into digital form suitable for subsequent filtering and digital-signal processing.

10.2 AFE features

The Analog Front End (AFE) feature list.

- Up to 3906 conversion frames per second support for a 3906 Hz output data rate (note that this is at a reduced resolution)
- 120 μg/sqrt (Hz) noise at nominal ODR = 488 Hz
- Programmable configuration for ADC conversions on a per-frame basis
  - No conversions (non-sample frame)
  - X, Y, and Z acceleration
  - X, Y, and Z acceleration plus temperature sensor reading
  - X, Y, and Z acceleration plus external ADC input
- Multiple modes of operation enable trade-off of power/latency versus ADC resolution and accuracy
  - Target raw resolution selectable to 10, 12, 14, and 16 bits
- Programmable acceleration ranges: 2g, 4g, and 8g full scale
- Programmable accelerometer anti-aliasing filters
- Integrated temperature sensor: -50°C to 150°C full scale
- External differential ADC input range: 0.65V ± 0.45V

10.3 AFE architecture and theory of operation

Figure 10-1 provides an overview of the AFE architecture. This includes the following major functions:

- Analog-to-digital converter common to all sensor functions
- Accelerometer functions
  - Transducer drive circuitry
  - MEMS transducer
  - Capacitance-to-voltage conversion and amplification
  - Anti-aliasing filters for X, Y, and Z dimensions
- Temperature sensor (9 mV/°C typical from -50°C to 150°C)

10.3.1 ADC operation

A conversion sequence is started when the AFE receives the "Start Φ_A" signal from the system integration module.

The ADC covers a 1.8V span, from -0.9V to +0.9V differential input. Individual sensor inputs are prescaled to match this range.

The XYZ axis accelerations are converted during each analog Phase Φ_A. Optionally, either the internal temperature sensor or external analog signal can be converted as a fourth measurement.

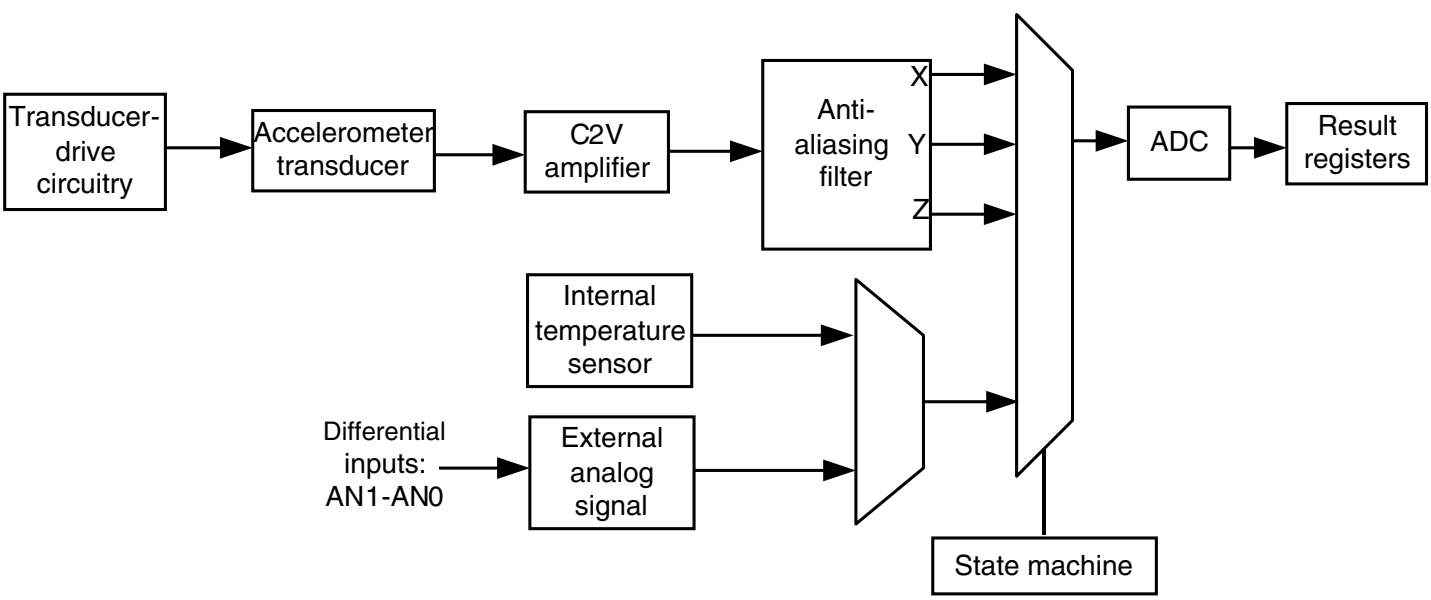


Figure 10-1. AFE data flow

The ADC contains a number of options to trade power consumption and conversion time for resolution and accuracy.

Table 10-1 shows analog-phase duration versus selected resolution. The lower the resolution, the lower the consumption and the longer time left for the digital phase.

- "CM" bits are the Conversion-Mode bits from the AFE_CSR register. These control the target raw resolution of the ADC conversion (10, 12, 14, or 16 bits).
- "N" represents the number of bits required to hold a full-scale ADC result. The actual number of effective bits may be less than N due to noise.

The ADC conversion results are delivered in a left-justified format. They are subsequently treated as 16-bit numbers regardless of the selected Conversion Mode (CM), so there may be missing codes in the low-order bits. This also keeps the scaling (Unit/LSB) independent of CM choice.

Table 10-1. Resolution and timing versus conversion mode

CM	ADC Number of Bits N	Analog Phase Time in µs 3 Samples	Analog Phase Time in µs[1] 4 Samples
11	10	135	154
10	12	159	186
01	14	207	250
00	16	303	378

Besides the ADC resolution, the Full-Scale range of the measured accelerometer data can be selected through the FS field of the AFE_CSR register.

Table 10-2 shows the various ranges versus FS setting, this applies only to the acceleration measurement. The third column reflects the sensitivity in mg/LSB associated with a given range. The appropriate scaling is realized by the AFE software routines that are further described in the Freescale *FXLC95000CL Software Reference Manual.*

Table 10-2. AFE scaling selection

FS	Full-Scale Data Range	XYZ Acceleration Scaling (mg/LSB)
00	+/-8 g	0.244
01	+/-2 g	0.061
10	+/-4 g	0.122
11	+/-8 g	0.244

10.3.2 Accelerometer principle of operation

The Freescale accelerometer is a surface micromachined, integrated-circuit accelerometer.

The device consists of three surface micromachined capacitive sensing cells (g-cells) and a CMOS signal conditioning ASIC contained in a single, integrated-circuit package. The sensing elements are sealed hermetically at the wafer level using a bulk micromachined "cap" wafer.

The g-cells are fabricated as a mechanical structure formed from semiconductor materials (polysilicon) using semiconductor processes (masking and etching). They can be modeled as a set of beams attached to a movable central mass that moves between fixed beams. The movable beams can be deflected from their rest position by subjecting the system to an acceleration. This is shown for a single dimension in Figure 10-2 (A).

When the beams attached to the center masses move, the distance from them to the fixed beams on one side will increase by the same amount that the distance to the fixed beams on the other side decreases. The change in distance is a measure of acceleration. The g-cell beams form two back-to-back capacitors Figure 10-2 (B). As the center plate moves with acceleration, the distance between the beams change and each capacitor's value will change.

The CMOS ASIC uses switched-capacitor techniques to measure the g-cell capacitors and extract the acceleration data from the difference between each set of two capacitors. The ASIC also conditions and filters the signal, providing a high-level output voltage that is ratiometric and proportional to acceleration.

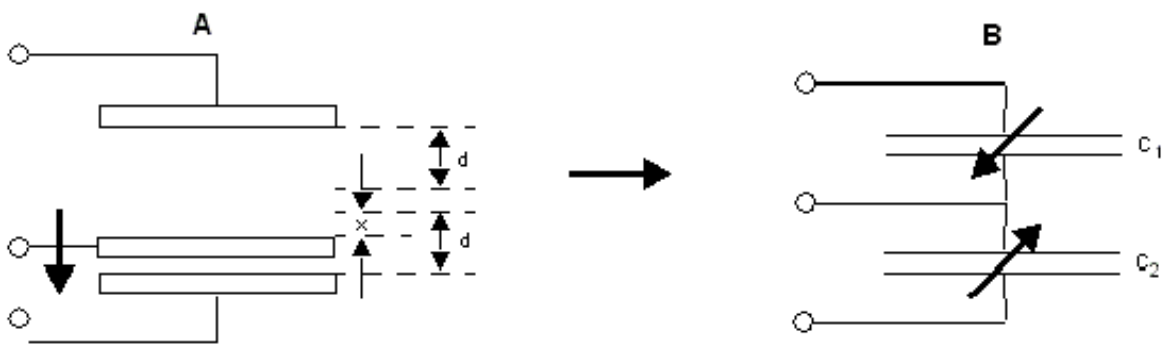


Figure 10-2. Transducer physical model (A) and equivalent circuit model (B)

Normally, the center mass in Figure 10-2 (A) is equidistant from both upper and lower plates such that C_1 and C_2 are equal. However, if the mass is displaced, the two capacitance values move in opposite directions. C_1 and C_2 form a capacitive divider. Figure 10-3 shows the outer legs of the divider (N_1 and N_2) driven with square waves that are 180 degrees out of phase from one another.

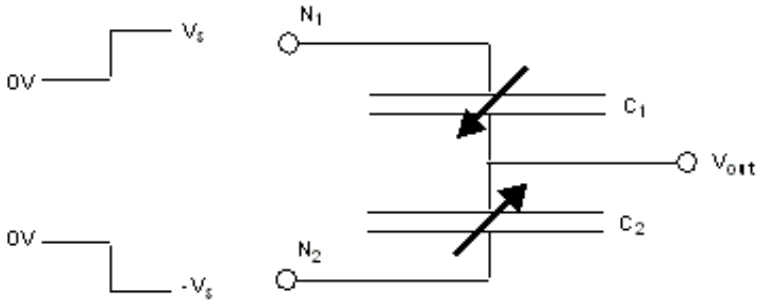


Figure 10-3. Capacitive divider showing outer legs

- When N_1 and N_2 switch from zero volts to V_s and $-V_s$, respectively, V_{out} is determined by the capacitor ratios as follows:

Vout = -Vs + [2 Vs C1 / (C1 + C2)]

Vout = [- (C1 + C2) Vs + 2 Vs C1] / (C1 + C2)

Vout = [- Vs C2 + Vs C1] / (C1 + C2)

Vout = Vs (C1 - C2) / (C1 + C2)

Capacitance Theory

This can be converted to an expression of V_{out} as a function of displacement by considering capacitance theory. If $C_1 + C_2$ is modelled as standard, parallel-plate capacitors and ignore fringing capacitance, the reader can note:

K = N A ε

C1 = K / (d + x)

C2 = K / (d - x)

C0 = K / d

Where:

- N = The number of beams in the g-cell (unitless)
- A = The area of the facing side of the beam in meter^2
- ε = The dielectric constant in farads per meter (1 farad = 1 coulomb per volt)
- K = A constant used to simplify the equations
- d = The nominal distance between the beams in meters
- x = The beam displacement in meters
- C_1 and C_2 = Measured capacitances
- C_0 = Nominal value of C_1 and C_2

Vout = Vs (C1 - C2) / (C1 + C2)

$Vout = Vs (K / (d + x) - K / (d - x)) / (K / (d + x) + K / (d - x))$

$Vout = Vs (1 / (d + x) - 1 / (d - x)) / (1 / (d + x) + 1 / (d - x))$

$Vout = Vs ((d - x - (d + x)) / (d^2 - x^2)) / ((d + x + (d - x)) / (d^2 - x^2))$

$Vout = Vs (d - x - (d + x)) / (d + x + (d - x))$

$Vout = -Vs (x / d)$

Newton's Second Law

F = mass * Acceleration

Where:

- F = Force applied to an object in Newtons ($1N = 1 kg * m/s^2$)
- The mass of the object is measured in kilograms
- The acceleration of the object is measured in (m/s^2)

Hooke's Law

F = -k * x

Where:

- F = The displacement force in N
- k = Spring constant in N / m
- x = The beam displacement in meters

Combining F = mass * Acceleration and F = -k * x :

x = mass * Acceleration / (-k)

Finally

Replacing x in Vout = -Vs (x / d) using the expression above gives:

$V_{out} = V_s$ (mass * Acceleration) / (k * d)

Where:

- k = Spring constant in N / m
- d = The nominal distance between the beams in meters
- The mass of the object is measured in kilograms
- The acceleration of the object is measured in (m / s^2)

10.4 AFE memory map and register descriptions

Note

The AFE register set may be accessed (both READ and WRITE) from Supervisor Mode only. After Freescale trim algorithms have been run on the data, ADC output values will be made available via reserved RAM locations.

Because complete trim routines are provide by Freescale, you do not need to fully comprehend the AFE and trim registers. Regardless, a detailed description of AFE_CSR register is given for the sake of completeness, because it contains the CM and FS fields mentioned previously.

The AFE register details are provided in the following section.

AFE memory map

Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
FFFF_EC40	AFE Control and Status Register (AFE_CSR)	16	R/W	0080h	10.4.1/195

10.4.1 AFE Control and Status Register (AFE_CSR)

The AFE_CSR is used to configure the AFE for the next conversion sequence and to monitor status of that conversion.

Address: FFFF_EC40h base + 0h offset = FFFF_EC40h

Bit	15	14	13	12	11	10	9	8
Read	FS		C4S		CM		RES	LP
Write								
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
Read	ST	RES			CCIEN	COCO	0	TT
Write				1			SWTRIG	
Reset	1	0	0	0	0	0	0	0

AFE_CSR field descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>15–14
FS</td><td>Full Scale Selection

AFE_CSR[FS] and AFE_BIAS[SC_AAF_EN] are combined to control AFE gain and trim mode as per the following table. Notice that AFE_BIAS[SC_AAF_EN] acts as an enable for the gain scaling controls of AFE_CSR[FS].

Trim parameters must be initialized once during the boot process. All trim values in this table are in 2's complement format. Those trim registers are reset only during power-on-reset.

Table 10-5. Accelerometer gain and trim settings
<table><tr><th>FS[1]</th><th>FS[0]</th><th>SC_AAF_EN</th><th>Gain</th><th>Trim Mode</th></tr><tr><td>0</td><td>0</td><td>0</td><td>1</td><td>8 g</td></tr><tr><td>0</td><td>1</td><td>0</td><td>1</td><td>8 g</td></tr><tr><td>1</td><td>0</td><td>0</td><td>1</td><td>8 g</td></tr><tr><td>1</td><td>1</td><td>0</td><td>1</td><td>8 g</td></tr><tr><td>0</td><td>0</td><td>1</td><td>1</td><td>8 g</td></tr><tr><td>0</td><td>1</td><td>1</td><td>4</td><td>2 g</td></tr><tr><td>1</td><td>0</td><td>1</td><td>2</td><td>4 g</td></tr><tr><td>1</td><td>1</td><td>1</td><td>1</td><td>8 g</td></tr></table></td></tr>
<tr><td>13–12
C4S</td><td>Conversion 4 Selection

00 Only acceleration measurements will be made in the next analog phase.
01 The temperature sensor output will be measured during the next analog phase. Results will be stored in AFE_TEMP.
10 The external analog input will be measured during the next analog phase. Results will be stored in AFE_EIC.
11 RESERVED.</td></tr>
<tr><td>11–10
CM</td><td>Conversion Mode

These bits control ADC resolution/accuracy versus power and conversion time tradeoffs. Values recommended for end user applications are listed in Table 10-2. From a physical perspective, CM[1:0] define N_{dec}, the number of cycles used for the Sigma-Delta portion of the conversion.

00 32 cycles
01 16 cycles
10 8 cycles
11 4 cycles</td></tr>
<tr><td>9
RES</td><td>Reserved</td></tr>
<tr><td>8
LP</td><td>Low Power Mode Enable

Low power mode or normal power mode. This bit determines if the device is placed in low power mode or normal power mode.

0 Normal power mode; AAF Resistor = 1M
1 Low power mode; AAF Resistor = 500K</td></tr>
<tr><td>7
ST</td><td>Self Test Enable</td></tr>
</table>

Table continues on the next page...

AFE_CSR field descriptions (continued)

Field	Description
	The ST bit activates the self-test function. When ST is set to one, an output change will occur to the acceleromter outputs, thus allowing a check of the functionality of the whole measurement chain. 0 Self test disabled 1 Self test enabled
6–5 RES	Reserved
4 RES	Reserved
3 CCIEN	Conversion Complete Interrupt Enable The conversion complete interrupt is intended for use in non-sample frames in which AFE conversions are explicitly started via the software trigger mechanism. In this case, the CPU can trigger a software conversion, and then execute a STOP command to drop into STOPFC mode. CPU and peripherals can be shut down temporarily so that digital noise is minimized during data acquisition. CPU and peripherals can then be re-awakened via the conversion complete interrupt. 0 (default value) Interrupt is not enabled. This setting should be left as-is for normal operation for the device using standard frame structures. In that case, wakeup mechanisms are already in place in the SIM Frame Control Register. It is an error to use CCIEN in a normal sample frame, as the conversion complete would generate duplicate interrupts. 1 Generate interrupt when COCO is asserted. Interrupt is cleared by clearing COCO.
2 COCO	Conversion Complete This bit indicates that an AFE cycle has completed and all ADC conversions are complete and ready for use. In the normal case in which the AFE is run only during CPU stop modes, this bit can be safely ignored. It can be useful when AFE conversions are explicitly run during CPU RUN mode with software triggering (see SWTRIG and TT bits below). 0 Read—Conversions not complete 1 Read—Conversions complete 0 Write—No effect 1 Write—Clear COCO bit
1 SWTRIG	Software Trigger When TT=0, writing a one to this field will initiate ADC conversions. 0 No effect 1 Initiate conversions when TT = 1. If TT = 0, this bit has no effect.
0 TT	Trigger Type Specifies whether ADC conversions are initiated via hardware trigger (start F_A) or software trigger (AFE_CSR[SWTRIG]) 0 Conversions are initiated via the "start F_A" signal from the System Integration Module 1 Conversions are initiated by writing "1" to AFE_CSR[SWTRIG]

10.5 Interrupts

The Φ_A signal generated by the AFE module is used by the SIM to generate the start of Φ_D interrupt.

The AFE module generates the End Φ_A signal which can be used by the system integration module, or SIM, to generate the Start Φ_D interrupt. End Φ_A is NOT itself an interrupt. FCSR[SFDIE] must be set to “1” to enable the Start Φ_D interrupt. The interrupt is cleared by writing a “1” to FCSR[SF].

The Conversion Complete interrupt can be enabled by setting AFE_CSR[CCIEN]. This must only be used in non-sample frames. Its use is intended to be mutually exclusive with the mechanism in the previous paragraph.

10.6 AFE Reset

Non-power-on-resets do not change the contents of AFE Trim Registers.

Chapter 11
On-Chip Oscillator (CLKGEN)

11.1 Introduction

The on-chip oscillator provides configurable registers to enable the user to operate the device in three modes: high speed mode of 16 MHz, low speed mode of 62.5kHz, or power down mode. The on-chip oscillator also provides a frame interval counter which enables the user to modulate the sampling and power consumption depending on the application.

The on-chip ocillator's modes of operation are summarized in Table 11-1.

Table 11-1. Oscillator modes

Mode	HLB	PDWNB
High-speed mode ($F_{osc\text{-}high}$ = 16 MHz)	1	1
Low-speed mode ($F_{osc\text{-}low}$ = $F_{osc\text{-}high}$/256 = 62.5 kHz)	0	1
Power down	X	0

Oscillator controls for speed control and module enable are generated by the System Integration Module (SIM).

These include:

- HLB = H/$\overline{L}$ speed control to the oscillator
  - 0 = Low speed. Normally used for Φ_I.
  - 1 = High speed. Normally used for Φ_A and Φ_D.
- PDWNB = Active-low, power-down control to the oscillator
  - 0 = Oscillator is powered down on the next negative edge of oscout, leaving oscout = 0.
  - 1 = Oscillator is powered.

In addition, the CLKGEN module contains controls for setting frame length, oscillator trim, and frame interval timer reset. It can also generate a fixed-frequency clock which is 1/8 x $F_{osc-low}$. This fixed-frequency clock (FFCLK) feeds the modulo timer and PWM module XCLK input.

11.2 High-level overview

The CLKGEN module is a two-speed oscillator, with high speed at 16 MHz nominal and low speed at high speed / 256 (62.5 kHz nominal). This section lists and describes the CLKGEN module mechanisms.

In the discussions that follow, one oscillator cycle is defined from falling edge to the next falling edge. The high and low periods of each oscillator cycle are assumed to be equal. The entire cycle is at low speed or high speed. The two modes are not mixed within a cycle. Transitions from one mode to the next take place on the falling edge of the oscillator.

The oscillator is in high-speed mode when the device is actively taking measurements or the CPU is running. Low-speed mode is used to conserve power during the idle phase. Figure 11-1 shows 256 cycles in high-speed mode, and one in low-speed mode. The second signal in the figure is running at a rate of high-speedclock/128, and is for reference purposes only.

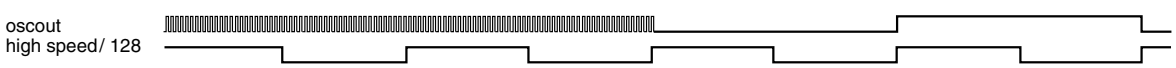


Figure 11-1. Oscillator output frequencies

Because the oscillator frequency varies over time, it is necessary to track how much time is spent in the high-speed mode and how much time in the low-speed mode. By tracking the two separately and scaling and adding the two numbers, we can derive a time base that is relatively invariant over time. The relative accuracy of any one point in time is limited by the time for one oscout cycle in low-speed mode. For 256/16 MHz, that works out to be 16 µs.

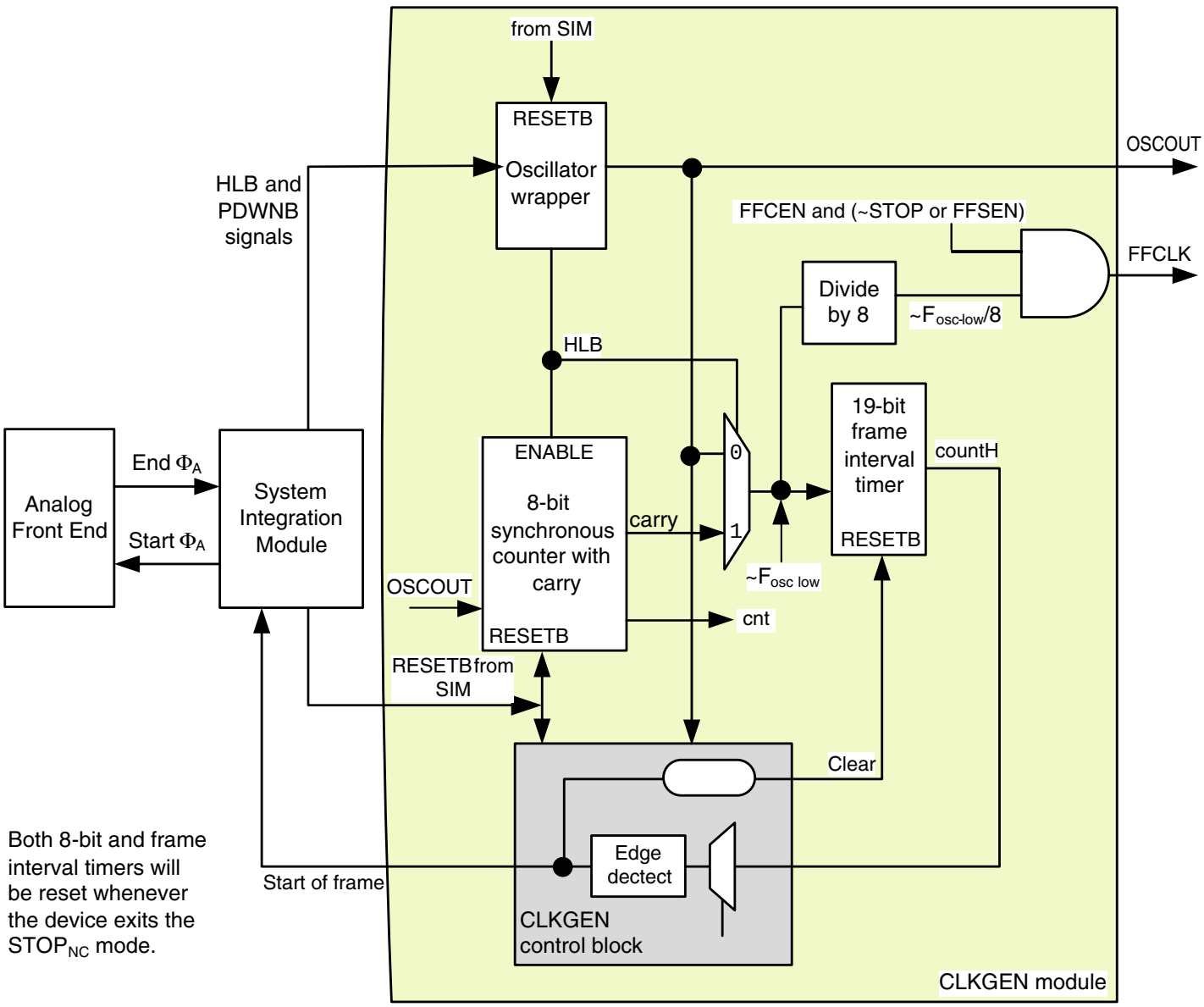


Figure 11-2. CLKGEN block diagram

Figure 11-2 shows the CLKGEN major components, and connections to the SIM and AFE modules. While the diagram provides insight to the CLKGEN module's internal circuits and its interaction with the System Integration module, it is primarily a simplified view of the oscillator to illustrate its control and operation. The image is *not* intended to fully describe the actual hardware implementation.

- Two-speed oscillator. This is contained within the *oscillator wrapper* function in the figure. The faster speed is nominally 16 MHz. The lower speed is this divided by 256 (62.5 kHz).
- Oscillator speed control HLB. This signal normally asserts low in reaction to a request by the CPU to terminate Φ_D. The timing of this request may vary from frame

to frame, depending on CPU loading. HLb asserts high at the beginning of each frame.

- Eight-bit synchronous counter with carry. This counter only increments when the oscillator is in high-speed mode. When the counter rolls over from value $FF to $00, a one-cycle-long carry bit is output. A synchronous counter is required in this location to ensure that the carry bit can be properly fed forward to the frame interval timer.
- This counter is *not* cleared between frames. Any residual count is used as a starting point in subsequent high-speed phases. This keeps the time-base error from accumulating over the course of many frames, though there will be jitter from the start of one phase to the next. The maximum amount of that jitter is equal to the 16 µs number quoted earlier.
- Frame interval timer. This counter controls the times between adjacent frames. The input to this counter is either oscout in low-frequency mode or the carry bit from the high-speed oscillator. The nominal rate of both is one pulse every 16 µs.
- The frame interval timer is automatically cleared at the end of each frame. Each new frame starts from a count of zero. The frame interval counter is also cleared on any exit from $STOP_{NC}$. The counter should restart from zero when the oscillator power down negates.
- The control block enables software control of power-up/down state and frame interval. This block is configured as a peripheral on the eight-bit IP bus. Additional mode control is supported by the SIM. This will be discussed in more detail in SIM.
- A pin of the device can be programmed to wake the device from deep-sleep mode ($STOP_{NC}$). In that mode, the oscillator is completely shut down. An asynchronous event on the pin is sufficient to restart the oscillator in high-speed mode and route the device directly into Φ_D (presumably for device configuration or other wake-up related task). Again, the frame interval counter is cleared as a result of wake-up from $STOP_{NC}$.
- The fixed-frequency clock runs at 1/8 x $F_{osc\text{-}low}$. This clock, which is available to the TPM and MTIM modules, will normally be disabled for applications which require highest sensor accuracy.

11.3 CLKGEN register offsets

The CLKGEN module is organized as a memory-mapped peripheral on the 8-bit IP bus. The following table summarizes the register offsets and functions. The CK_TRIM registers will be directly programmed by the ROM boot function after Power-On Reset (POR), therefore the user is not required to program the CK_TRIM registers.

Table 11-2. CLKGEN register offsets

Register	Offset	Function
CK_OSCTRL	$0	Oscillator Control Register
Reserved	$1	Reserved location
Reserved	$2	Reserved location
Reserved	$3	Reserved location
CK_TRIMLM	$4	Most significant byte of the low frequency trim
CK_TRIMLL	$5	Least significant byte of the low frequency trim
CK_TRIMHM	$6	Most significant byte of the high frequency trim
CK_TRIMHL	$7	Least significant byte of the high frequency trim

11.4 CLKGEN memory map and register descriptions

A detailed description of the CLKGEN_CK_OSCTRL register is provided in this section. The CK_TRIM registers will be directly programmed by the ROM boot function after Power-On Reset (POR), therefore the user is not required to program the CK_TRIM registers.

The CLKGEN module is organized as a memory-mapped peripheral on the eight-bit IP Bus. Register details are provided in the following section.

CLKGEN memory map

Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
FFFF_8080	Oscillator Control Register (CLKGEN_CK_OSCTRL)	8	R/W	80h	11.4.1/203

11.4.1 Oscillator Control Register (CLKGEN_CK_OSCTRL)

Table 11-5. Frame interval and clocks as a function of FLE (Assumed 200 µs Φ_A duration)

FLE	2^{FLE}	t_F (secs)	Max frames per second	Max fast-clock cycles per frame	Φ_D Duration (µs)	Max fast-clock cycles per Φ_D	% CPU available to Φ_D
4	16	2.56E-04	3906.25	2048	56.00	448	21.875%
5	32	5.12E-04	1953.13	4096	312.00	2496	60.938%

Table continues on the next page...

Table 11-5. Frame interval and clocks as a function of FLE (Assumed 200 µs Φ_A duration) (continued)

FLE	2^{FLE}	t_F (secs)	Max frames per second	Max fast-clock cycles per frame	Φ_D Duration (µs)	Max fast-clock cycles per Φ_D	% CPU available to Φ_D
6	64	1.02E-03	976.56	8192	824.00	6592	80,469%
7	128	2.05E-03	488.28	16,384	1848.00	14,784	90.234%
8	256	4.10E-03	244.14	32,768	3896.00	31,168	95.117%
9	512	8.19E-03	122.07	65,536	7992.00	63,936	97.559%
10	1024	1.64E-02	61.04	131,072	16,184.00	129,472	98.779%
11	2048	3.28E-02	30.52	262,144	32,568.00	260,544	99.390%
12	4096	6.55E-02	15.26	524,288	65,336.00	522,688	99.695%
13	8192	1.31E-01	7.63	1,048,576	130,872.00	1,046,976	99.847%
14	16384	2.62E-01	3.81	2,097,152	261,944.00	2,095,552	99.924%
15	32,768	5.24E-01	1.91	4,194,304	524,088.00	4,192,704	99.962%
16	65,536	1.05E-00	0.95	8,388,608	1,048,376.00	8,387,008	99.981%
17	131,072	2.10E+00	0.48	16,777,216	2,096,952.00	16,775,616	99.990%
18	262,144	4.19E+00	0.24	33,554,432	4,194,104.00	33,552,832	99.995%

Address: FFFF_8080h base + 0h offset = FFFF_8080h

Bit	7	6	5	4	3	2	1	0
Read Write	FCEN	FFCEN	FFSEN	FLE				
Reset	1	0	0	0	0	0	0	0

CLKGEN_CK_OSCTRL field descriptions

Field	Description
7 FCEN	Frame Counter Enable This bit disables both the Frame Interval Counter and the seven-bit synchronous counter that tracks fast clocks. Both counters restart from zero when re-enabled. 0 Frame Interval Counter not enabled. 1 Frame Interval Counter enabled (default).
6 FFCEN	Fixed Frequency Clock Enable The CLKGEN module can generate a fixed frequency clock of frequency 1/8 x $F_{osc\text{-}low}$. This clock is inactive during reset. This clock is subject to a significant amount of jitter (approximately $\pm F_{osc\text{-}low}^{-1}$) as it is reconstructed from two mutually exclusive (in time) frequencies. The divider for FFCLK is set to zero during system reset and $STOP_{NC}$. **NOTE:** Using the FFCLK during STOP modes may increase noise in converted ADC results. For best results, disable this feature in STOP (FFSEN=0). 0 FFCLK is not enabled. 1 FFCLK enabled during RUN. Operation in $STOP_{SC}$ and $STOP_{FC}$ is dependent upon the FFSEN bit.
5 FFSEN	Fixed Frequency Clock STOP Enable This bit only applies when FFCEN = 1.

Table continues on the next page...

CLKGEN_CK_OSCTRL field descriptions (continued)

Field	Description
	0 FFCLK is disabled in all STOP modes 1 FFCLK is enabled during STOPFC and STOPSC. It is disabled during STOPNC.
FLE	Frame Length Exponent The Frame Interval Timer is reset to zero whenever the FLE field is written (even if the value does not change). Additionally, the frame interval counter is held inactive and set to zero whenever FLE is outside of the range $04 to $12. Ensure the CK_OSCTRL[FLE] field is set to a valid value prior to enabling automatic STOPSC to STOPFC transitions via the SIM STOPCR[SCtoFC] bit. Failure to do so can result in higher than desired IDD stop mode currents.

11.5 Interrupts

The CLKGEN module generates the start Φ_A signal, which acts as a wake-up to the analog front end or be programmed to bypass the analog phase by generating a Φ_D interrupt. This would bypass the analog phase and go straight to the digital phase.

Under certain circumstances, the Φ_A signal could cause the System Integration Module (SIM) to generate an interrupt, to signal a start-frame error. See Frame Control and Status Register.

Chapter 12
Flash Memory Controller (FLASH)

12.1 Flash memory overview

The flash memory controller is partitioned into two spaces in memory. The flash array is the primary space for program storage. The flash controller is the secondary space and enables supervisor access to module registers.

Figure 12-1 illustrates the interaction of the two memory spaces, the flash array and the flash controller. In-circuit programming enables the operating program to be loaded into the flash memory after final assembly of the application product. It is possible to program the entire flash array through the single-wire, background-debug interface.

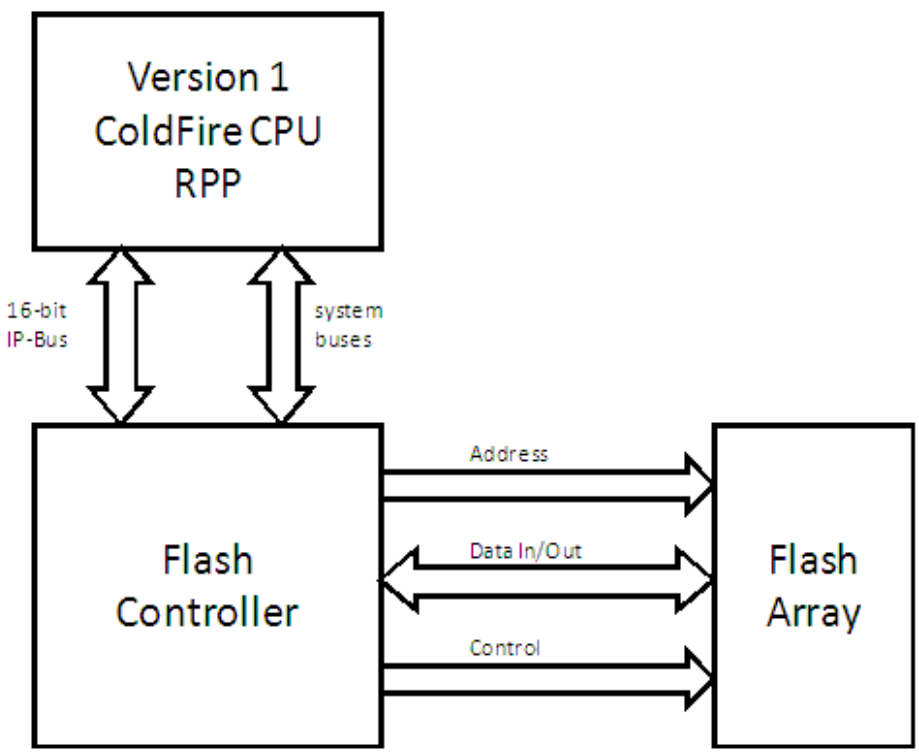


Figure 12-1. Flash block diagram

Flash address and data values are communicated over system busses. Flash controls are managed via registers mapped onto the IP-bus space. User access to program/erase functions is via dedicated ROM function calls. Direct access to flash controller registers is disallowed.

NOTE

Flash controller registers are available only from Supervisor mode. User access to flash functions is encapsulated via a set of ROM routines. The flash array can only be written in Supervisor mode. Violations to this, as well as the restrictions above, will result in an access-error exception. **Program and erase operations should only be done using ROM routines.**

12.2 Features

Features of the on-chip flash memory include:

- 32K-deep by 32-bit main array (128 KB total)
- Page erase size = 1024 bytes
- Security lockout
- Protection against accidental programming/erase operations
- Program, erase and mass-erase procedures can be performed using pre-programmed ROM routines.

12.3 Theory of operation

Flash memory is nonvolatile and is ideal for single-supply applications, enabling field reprogramming *without using external, high-voltage sources* (for programming or erase operations). Contents are retained for an extended period of time over 100 years under nominal conditions.

Contents of flash memory can be read randomly, just like RAM. Array read-access time is one bus cycle for bytes, aligned words and aligned double-words. Unlike random access memory, flash memory cannot simply be written with a desired value. It must first be "erased" and "programmed." For flash memory, an erased bit reads 1 and a programmed bit reads 0. Once programmed to 0, a bit cell remains in that state until erased again. A bit cell cannot be "programmed" to change from 0 to 1.

It is not possible to read from flash memory while it is being erased or programmed.

Bit cells can be erased/programmed a finite number of times before data integrity issues begin to occur. However, the minimum number of erase/program cycles can exceed 20,000 under nominal conditions.

NOTE

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.

The flash hard block has a number of control signals associated with programming and erase operations. These must be sequenced over time and in a specified manner, to erase and subsequently program flash memory. Internally generated, high voltages are applied for specific periods of time which must not be exceeded.

The hardware wrapper for flash memory provides rudimentary interlocks and safeguards, as well as some strobe generation. Higher-level intelligence is provided via canned ROM routines for basic flash operations.

All program erase operations must be performed using ROM routines executed while the CPU is in Supervisor mode. A special trap function (call_trap) is supplied which places the CPU in the Supervisor state, calls the appropriate ROM routine and then returns to User mode. For additional details, see User-Callable ROM functions .

Any attempt to directly write any flash controller registers in normal mode of operation will result in generation of an access-error exception.

12.4 Flash controller modes of operation

There are four modes of operation for the flash controller: IDLE, READ, PROGRAM and ERASE. **PROGRAM and ERASE modes are only accessible when the CPU is in the Supervisor state.**

Table 12-1. Flash controller modes

Flash mode	Description
IDLE	Whenever the flash is not accessed by the CPU, including during WAIT and STOP modes, the flash will be in IDLE mode. The flash module will be in Standby and consume minimal power.
READ	The flash will be in READ mode when it is read by the CPU. However, when the flash is in either PROGRAM or ERASE mode, the flash module cannot be read; any attempt to read data from flash will return undefined data.
PROGRAM	In Program mode, the flash array can be programmed 32 bits at a time. Individual data bits can be programmed from 1 to 0, but not from 0 to 1.
ERASE	Flash memory can be erased one page (1024 bytes) at a time or the entire main array can be erased in one mass-erase action. The erase state of all data bits in the array is 1.

12.5 Memory maps

The flash module is partitioned into two areas in memory:

- The first area is the array memory that contains the main flash array.
- The second area enables supervisor access to module registers, and is mapped into the 16-bit, IP-bus space. User access to the flash controller is through dedicated ROM functions. Direct user access to the controller register set is prohibited.

12.5.1 Flash controller array memory map

The main flash array is designed to support 128 KB of general program storage. Four bytes of this are reserved for use in storing nonvolatile parameters.

Table 12-2. Flash controller array memory map

Address range	Function
(00) 00_0000 - (00) 01_FFFB 128 K - 4 bytes	General storage
(00) 01_FFFC - (00) 01_FFFF 4 bytes	Reserved for nonvolatile options (4 bytes)

FOPT[7:0] is loaded from address 0x01_FFFF during each reset sequence.

The boot-to-flash flag (FOPT[FB]) is set to the inverse of Bit 5 of address 0x01_FFFE during the ROM boot process on power-on-reset. Thus, if Bit 5 of 0x01_FFFE is set to "1," the device will *not* boot to flash. As a consequence, a virgin device with erased flash will boot directly into the ROM command interpreter on power-up.

Similarly, the FOPT[9:8] bits are loaded from bits [1:0] of 0x01_FFFE during the POR boot sequence by the ROM bootloader.

12.6 FLASH registers and control bits

The last four bytes of the flash array (at 0x01_FFFC) are reserved and should not be used by the application program. The least-significant byte of this location, 0x01_FFFF, is referred to as NVOPT. It contains bits that define flash security and write-protection levels.

Table 12-3. Reserved locations in the main array

	0x01_FFFC	0x01_FFFD	0x01_FFFE	0x01_FFFF
Identifier	CRC[15:8]	CRC[7:0]	NVBOPT	NVOPT
Used for	Expected CRC to be computed over 0x0000 to 0x01_FFFB		FOPT[15:8] The boot-to-flash flag (FOPT[FB]) is set to the inverse of Bit 5 of address 0x01_FFFE during the ROM boot process on power-on-reset. FOPT[10:8] are loaded from bits 2:0 of 0x01_FFFE during the ROM sequence.	FOPT[7:0] FOPT[7:0] is loaded with NVOPT byte at reset.

The second least-significant byte of this location (0x00_FFFE) is referred to as NVBOPT. It contains control bits that define whether or not a CRC check is run at boot time and whether the device boots to flash or not. Finally, the 16-bit CRC value is stored at 0x00_FFFC.

12.7 FLASH memory map and register descriptions

Flash control registers are not available directly from User mode. The FOPT register will be altered during POR and reset with the content of the upper two bytes of the main flash array. Flash functions can only be accessed via the ROM routines described in ROM.

FLASH memory map

Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
FFFF_EC20	Flash Options Register (FLASH_FOPT)	16	R/W	00FEh	12.7.1/211

12.7.1 Flash Options Register (FLASH_FOPT)

FOPT[7:0] is loaded from the last byte of the main array (NVOPT) during the reset sequence. Therefore all modifications to FOPT[7:0] are lost at the next reset. Permanent changes to FOPT[7:0] can only be done by modifying the flash data stored at NVOPT.

To change the value in this register, erase and reprogram the NVOPT location in flash memory as usual and then issue a new MCU reset.

FOPT can only be read or modified when the CPU is in Supervisor mode.

The upper byte of FOPT is cleared only on power-on-reset.

Address: FFFF_EC20h base + 0h offset = FFFF_EC20h

Bit	15	14	13	12	11	10	9	8
Read	CSR1		BF		0	MECFB	CHECKB	
Write		0		0				
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
Read	1	1	PW	PROTB	1	SSW	SSC	
Write								
Reset	1	1	1	1	1	1	1	0

FLASH_FOPT field descriptions

Field	Description
15 CSR1	Clear Security Request — This bit is inspected by the boot function in ROM. The CSR1 bit is set via the BDM by setting XCSR[ERASE]. If CSR1 is set, then the boot routine will erase all user-assigned pages of flash memory. • Read 0, no security request is present. • Read 1, BDM has requested that the flash be erased and unsecured • Write 0, no effect. • Write 1, clear CSR1. To learn more about the BDM interface, see “Extended Configuration/Status Register (XCSR)”. Erase functions for CSR1 are implemented in firmware via the ROM boot function.
14 Reserved	This field is reserved. Reserved Always write as 0.
13 BF	Boot from flash This is a simple R/W bit in the flash controller. This bit is initialized to the inverse of Bit 5 of flash location 0x01_FFFE by the boot ROM on power-up. It is read by the ROM code in a later step of the reset process. The code value affects where control is ultimately transferred. 0 Do not boot from flash. 1 Boot from flash on next reset.
12 Reserved	This field is reserved. Reserved Always write as 0.
11 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
10 MECFB	Mass Erase on CRC Failure This bit field is used as a control bit for the ROM boot function. It is only applicable if CHECKB = 01 or 10. In those cases, if the CRC check fails and MECFB=0, then the user portion of the flash memory will be erased. This bit provides additional protection of customer code from hacker attempts to bypass security via "interrupted" erase operations. This bit is initialized to Bit 2 of flash location 0x01_FFFE by the boot ROM on power-up. It is read by the ROM code in a later step of the reset process.

Table continues on the next page...

FLASH_FOPT field descriptions (continued)

Field	Description
	1 Do nothing. 0 Erase the user portion of the main flash array.
9–8 CHECKB	Perform Flash Checksum This bit field is used as a control bit for the ROM boot function. It controls whether or not a flash checksum is computed and checked against expected results before transferring control to code executing in flash. This field is loaded from location 0x01_FFFE by the ROM bootloader on POR only. It can be modified via software and will affect operation during subsequent non-power-on reset sequences. 00 Do not perform checksum. 01 Perform checksum on POR only. 10 Perform checksum on any reset. 11 Do not perform checksum.
7 Reserved	This field is reserved. This read-only field is reserved and always has the value 1.
6 Reserved	This field is reserved. This read-only field is reserved and always has the value 1.
5 PW	PROTB Writeable The PROTB bit can be written from software only when PW = 1. If PW = 0, it must first be reset to 1 before PROTB can be modified.
4 PROTB	Active Low Write Protect Used to inhibit programming and erase operations. This bit can only be written when PW = 1. 0 Array is protected from unintentional program/erase operations. 1 Array is not protected from unintentional program/erase operations.
3 Reserved	This field is reserved. This read-only field is reserved and always has the value 1.
2 SSW	Security State Writeable The SSC bit field can be written from software only when SSW = 1. If SSW = 0, it must first be reset to one before SSC can be modified.
SSC	Security State Code These bits determine the security state of the MCU. When the MCU is secure, the contents of flash memory cannot be accessed by instructions from any unsecure source including the background debug interface. These bits can only be written when SSW = 1. Security can be temporarily cleared by setting these bits to 11, however, they will be re-initialized from NVOPT on every reset. These bits are initialized from bits 1:0 of flash location 0x01_FFFF during each reset sequence. These bits are initialized to 10 (Secure) by when the peripheral reset is asserted. The flash wrapper will almost immediate overwrite them as the module exits reset. 00 Unsecured 01 Unsecured 10 SECURE 11 Unsecured

12.8 Initialization information

12.8.1 Factory

Devices are usually shipped with the flash memory in an *erased* condition. However, trim algorithms and basic sensor functions will be included as header and source files in a CodeWarrior C project framework.

12.8.2 End user

The flash module can be read after the device has completed the reset operation. No special initialization procedure is required to initialize the module.

FOPT[7:0] is automatically loaded from NVOPT (0x01_FFFF) during any reset sequence.

A user program may need to be programmed to the flash module before the device can be used in the targeted application. The following sections describe the programming and erase operation of the flash module.

To facilitate user, flash-area erase and program operations, Freescale will provide appropriate abstraction tools (in the FXLC95000CL evaluation kit), which will isolate the end-user from the ROM routines.

12.9 Programming model

All user access to the flash controller is via Freescale supplied ROM routines which are described in Callable utilities. Please note that interrupts are disabled when these functions execute and STOP mode operation is temporarily disabled. System clocks will remain in their high-speed states (16 MHz) during these operations.

For details of the ROM function for flash programming and flash erase, see User-Callable ROM functions.

The user can control the state of the FOPT bit (PROTB) via RMF_FLASH_PROTECT and RMF_FLASH_UNPROTECT.

Security can be temporarily suspended via RMF_FLASH_UNSECURE.

12.10 Security

This family of devices includes circuitry to prevent unauthorized access to the contents of flash memory. When security is engaged, BDM control/communication with the CPU is extremely limited. Read/Write access via BDM is then limited to XCSR[31-24], CSR2[31-24].

It is possible to check STOP/HALT status of the CPU, enable BDM clocks, configure reset behavior and assert reset.

Table 12-6. CPU resources available via BDM in secure mode

Register field	Field name	R/W	Function
XCSR[31]	CPU_HALT	R	1, if CPU is halted
XCSR[30]	CPU_STOP	R	1, if CPU is in STOP mode
XCSR[29:27]	CSTAT	R	BDM command status
XCSR[26]	CLKSW	R/W	BDM clock select (no function on FXLC95000CL)
XCSR[25]	SEC	R/W	Security status (1 = Secured)
XCSR[24]	ENBDM	R/W	Enable BDM (1 = BDM is enabled)
CSR2[31]	PSTBP	R	PST buffer stop
CSR2[30]	RESERVED	N/A	
CSR2[29]	COPHR	R/W	COP halt after reset (no function on FXLC95000CL)
CSR2[28]	IOPHR	R/W	Illegal operation halt after reset
CSR2[27]	IADHR	R/W	Illegal address halt after reset
CSR2[26]	RESERVED	N/A	
CSR2[25]	BFHBR	R/W	BDM force halt on BDM reset
CSR2[24]	BDFR	W	Background debug force reset

Security is engaged or disengaged based on the state of nonvolatile register bits shown in FOPT[SSC]. During the reset sequence, the contents in bits 7:0 of the nonvolatile location NVOPT (0x01_FFFF) are copied from flash into bits 7:0 of FOPT register. A user engages security by programming the NVOPT location which can be done at the same time that the flash memory is programmed.

Notice the erased state (SSC = 11) makes the MCU unsecure. When SSC bits of NVOPT are programmed to SECURE (10), the next reset will engage security. In order to permanently disengage security, the NVOPT bits must be erased. Security can be disengaged by a software interrupt (SWI) that will switch the FXLC95000CL to Supervisor mode.

The SWI should perform the following functions:

1. If necessary, set PROTB = 1.

2. Mass-erase the flash and verify that the contents have been erased.
3. Set SSC = 11, assuming verify passed.
4. Return.

NOTE

When the device boots up to normal operating mode-where MS pin is high during RESET and SSC programmed to SECURE (10)-flash security is engaged. In this state, all BDM communication is blocked and background debugging is not allowed.

Chapter 13
Port Controls (PC)

13.1 FXLC95000CL port control customizations

Port control register bits are mapped to the Rapid GPIO (RGPIO) pins, and enable their customization according to general rules. Some exceptions to those rules exist when specific pins function are different from the standard RGPIO.

There are 2 instances of the port-control module on the FXLC95000CL. Each port control module can control up to 8 pins.

PC0 controls drive strength, slew rate and pull-up controls for RGPIO[15:8]. PC1 controls these parameters for RGPIO[7:0].

Table 13-1. RGPIO Port Controls Mapping

RGPIO Bit	Port Control
15	PC0[7]
14	PC0[6]
13	PC0[5]
12	PC0[4]
11	PC0[3]
10	PC0[2]
9	PC0[1]
8	PC0[0]
7	PC1[7]
6	PC1[6]
5	PC1[5]
4	PC1[4]
3	PC1[3]
2	PC1[2]
1	PC1[1]
0	PC1[0]

13.1.1 General rules

- PCxSE = 0 (output slew rate control is disabled)
- PCxDS = 0 (low output drive strength)
- PCxIFE = 1 (input filters are enabled)
- PCxPE = 0 (pull-ups are not enabled)
- I^2C pins are open-drain when the pins are configured for use as I^2C in the SIM Pin Mux Control Registers. (See SIM memory map/register definitions.)

13.1.2 Exceptions to the general rules

- When no slew rate control is enabled by pull-up resistors, the BKGD/MS pin (RGPIO9) defaults to high drive strength.
- When A4 is programmed as an interrupt, the pull-up/down enable is controlled by the IRQSC[IRQPDD] signal (instead of being controlled by PC1PE[PE4]). At the same time, if the IRQSC[IRQPDD] signal is enabled and IRQSC[IRQEDG] is 0, then a pull-up resistor is used; however, if IRQSC[IRQEDG] is 1, then a pull-down resistor is used.

13.1.3 Pins not covered by the port control modules

The RESETB pin is not multiplexed with GPIO, so the RESETB pad cell is not configurable. The RESETB pin's fixed configuration is:

- Low output drive strength
- Input filter is enabled
- Pull-up resistor is enabled
- The output buffer of the RESETB pin is open drain. (See RESETB pin.)

13.2 Standard pin controls

The Rapid GPIO (RGPIO) ports use four registers to configure the pull-ups, slew rate, drive strength, and input filters.

13.2.1 Pin controls

A set of registers (shown in Figure 13-1) control pull-ups, slew-rate, drive-strength, and input-filter enables for the pins. This set of registers also may be used in conjunction with the peripheral functions on these pins.

These registers are associated with the Rapid General Purpose IO (RGPIO) ports, but operate independently of those ports.

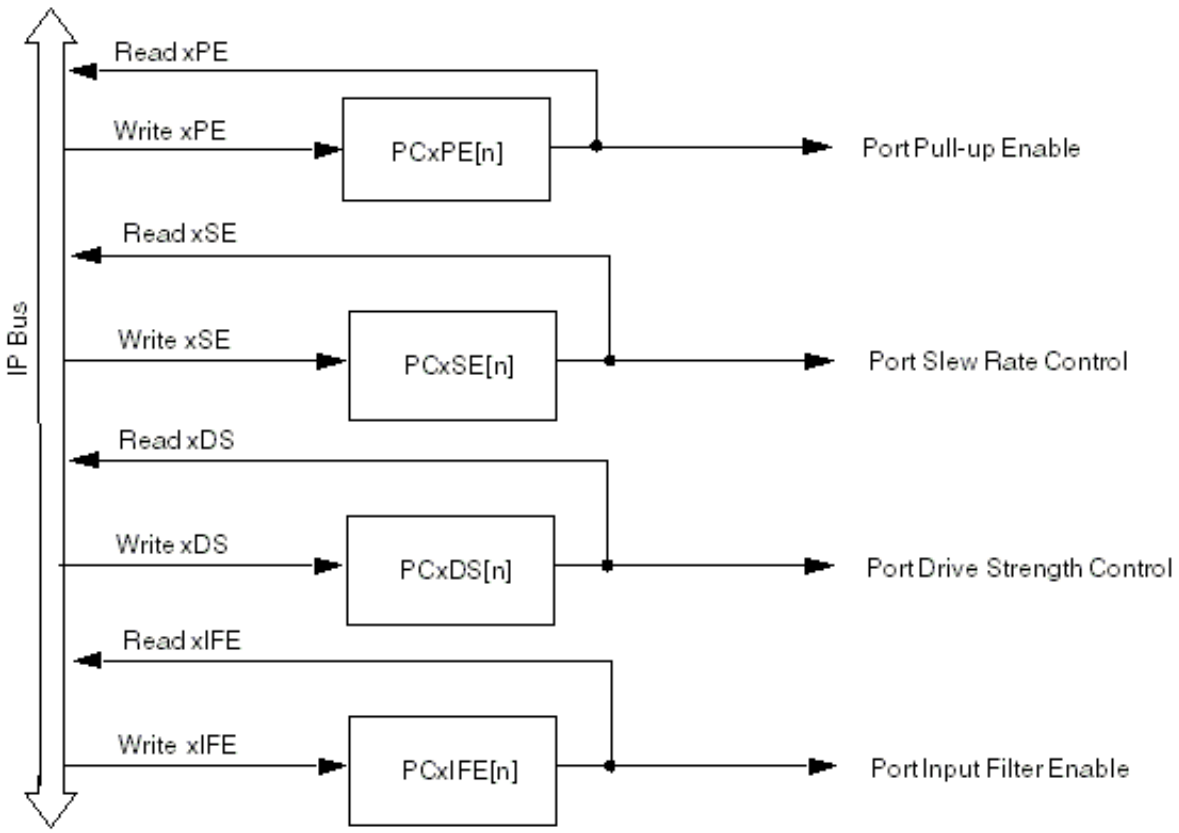


Figure 13-1. Pin control logic

13.2.2 Pin controls programming model

These registers control the pull-ups, slew rate, drive strength, and input filter for all the pins and may be used for the peripheral functions on these pins.

Table 13-2. Pin control registers

Register	Description	Access
PCxPE	Port x Pull-up Enable register	read/write
PCxSE	Port x Slew Rate Enable register	read/write

Table continues on the next page...

Table 13-2. Pin control registers (continued)

PCxDS	Port x Drive Strength Selection register	read/write
PCxIFE	Port x Input Filter Enable register	read/write

For the absolute address assignments for all registers, see the tables in High-level memory map. That section refers to registers and control bits only by their names.

NOTE

A Freescale-provided equate or header file normally is used to translate these names into the appropriate absolute addresses.

13.3 PC memory map/register definition

Port Control registers control and select the characteristics for each pin used in the Rapid GPIO (RGPIO) ports.

The port number applicable to the register is shown in each register's register label, after the "PC" part of the label. For example, the PC0_PE register applies to Port 0. In the following register description sections, "Port x" in the section heading implies that the section information applies to any port, regardless of port number.

PC memory map

Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
FFFF_80E0	Port x Pull-Up Enable Register (PC0_PE)	8	R/W	00h	13.3.1/220
FFFF_80E1	Port x Slew Rate Enable Register (PC0_SE)	8	R/W	00h	13.3.2/222
FFFF_80E2	Port x Drive Strength Selection Register (PC0_DS)	8	R/W	00h	13.3.3/223
FFFF_80E3	Port x Input Filter Enable Register (PC0_IFE)	8	R/W	FFh	13.3.4/224
FFFF_8100	Port x Pull-Up Enable Register (PC1_PE)	8	R/W	00h	13.3.1/220
FFFF_8101	Port x Slew Rate Enable Register (PC1_SE)	8	R/W	00h	13.3.2/222
FFFF_8102	Port x Drive Strength Selection Register (PC1_DS)	8	R/W	00h	13.3.3/223
FFFF_8103	Port x Input Filter Enable Register (PC1_IFE)	8	R/W	FFh	13.3.4/224

13.3.2 Port x Pull-Up Enable Register (PCx_PE)

An internal pull-up device can be enabled for each port pin by setting the corresponding bit in the pull-up-enable register (PE[*n*]).

The pull-up device is disabled if any of the following occur:

- The pin is configured as an output by the parallel I/O control logic
- The pin is configured as disabled by a shared (and controlling) peripheral function
- The pin is controlled by an analog function
- There is a power reset, except for the RESETB and BKGD/MS pins

Each of these control bits determines if the internal pull-up device is enabled for the associated pin. For Port 0 pins that are configured as outputs, these bits have no effect and the internal pull-up devices are disabled.

Address: Base address + 0h offset

Bit	7	6	5	4	3	2	1	0
Read Write	PE7	PE6	PE5	PE4	PE3	PE2	PE1	PE0
Reset	0	0	0	0	0	0	0	0

PCx_PE field descriptions

Field	Description
7 PE7	Detailed description above. 0 Internal pull-up device disabled for Port x bit 7. 1 Internal pull-up device enabled for Port x bit 7.
6 PE6	Detailed description above. 0 Internal pull-up device disabled for Port x bit 6. 1 Internal pull-up device enabled for Port x bit 6.
5 PE5	Detailed description above. 0 Internal pull-up device disabled for Port x bit 5. 1 Internal pull-up device enabled for Port x bit 5.
4 PE4	Detailed description above. 0 Internal pull-up device disabled for Port x bit 4. 1 Internal pull-up device enabled for Port x bit 4.
3 PE3	Detailed description above. 0 Internal pull-up device disabled for Port x bit 3. 1 Internal pull-up device enabled for Port x bit 3.
2 PE2	Detailed description above. 0 Internal pull-up device disabled for Port x bit 2. 1 Internal pull-up device enabled for Port x bit 2.
1 PE1	Detailed description above. 0 Internal pull-up device disabled for Port x bit 1. 1 Internal pull-up device enabled for Port x bit 1.
0 PE0	Detailed description above. 0 Internal pull-up device disabled for Port x bit 0. 1 Internal pull-up device enabled for Port x bit 0.

13.3.3 Port x Slew Rate Enable Register (PCx_SE)

Slew rate control can be enabled for each port pin by setting the corresponding bit in the slew rate control register (SE[*n*]). When enabled, the 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.

Each of these control bits determines if the output slew rate control is enabled for the associated pin. For Port 0 pins configured as inputs, these bits have no effect.

Address: Base address + 1h offset

Bit	7	6	5	4	3	2	1	0
Read Write	SE7	SE6	SE5	SE4	SE3	SE2	SE1	SE0
Reset	0	0	0	0	0	0	0	0

PCx_SE field descriptions

Field	Description
7 SE7	Detailed description above. 0 Output slew rate control disabled for Port x bit 7. 1 Output slew rate control enabled for Port x bit 7.
6 SE6	Detailed description above. 0 Output slew rate control disabled for Port x bit 6. 1 Output slew rate control enabled for Port x bit 6.
5 SE5	Detailed description above. 0 Output slew rate control disabled for Port x bit 5. 1 Output slew rate control enabled for Port x bit 5.
4 SE4	Detailed description above. 0 Output slew rate control disabled for Port x bit 4. 1 Output slew rate control enabled for Port x bit 4.
3 SE3	Detailed description above. 0 Output slew rate control disabled for Port x bit 3. 1 Output slew rate control enabled for Port x bit 3.
2 SE2	Detailed description above. 0 Output slew rate control disabled for Port x bit 2. 1 Output slew rate control enabled for Port x bit 2.
1 SE1	Detailed description above. 0 Output slew rate control disabled for Port x bit 1. 1 Output slew rate control enabled for Port x bit 1.
0 SE0	Detailed description above.

Table continues on the next page...

PCx_SE field descriptions (continued)

Field	Description
	0 Output slew rate control disabled for Port x bit 0. 1 Output slew rate control enabled for Port x bit 0.

13.3.4 Port x Drive Strength Selection Register (PCx_DS)

An output pin can be selected to have high output drive strength by setting the corresponding bit in the drive-strength select register (DS[*n*]). When high drive is selected, a pin is capable of sourcing and sinking greater current. Even though every I/O pin can be selected as high drive, users 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 enables 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.

Each of these control bits selects between low- and high-output drive for the associated pin. For Port 0 pins configured as inputs, these bits have no effect.

Address: Base address + 2h offset

Bit	7	6	5	4	3	2	1	0
Read Write	DS7	DS6	DS5	DS4	DS3	DS2	DS1	DS0
Reset	0	0	0	0	0	0	0	0

PCx_DS field descriptions

Field	Description
7 DS7	Detailed description above. 0 Low output drive strength selected for Port x bit 7. 1 High output drive strength selected for Port x bit 7.
6 DS6	Detailed description above. 0 Low output drive strength selected for Port x bit 6. 1 High output drive strength selected for Port x bit 6.
5 DS5	Detailed description above. 0 Low output drive strength selected for Port x bit 5. 1 High output drive strength selected for Port x bit 5.
4 DS4	Detailed description above. 0 Low output drive strength selected for Port x bit 4. 1 High output drive strength selected for Port x bit 4.

Table continues on the next page...

PCx_DS field descriptions (continued)

Field	Description
3 DS3	Detailed description above. 0 Low output drive strength selected for Port x bit 3. 1 High output drive strength selected for Port x bit 3.
2 DS2	Detailed description above. 0 Low output drive strength selected for Port x bit 2. 1 High output drive strength selected for Port x bit 2.
1 DS1	Detailed description above. 0 Low output drive strength selected for Port x bit 1. 1 High output drive strength selected for Port x bit 1.
0 DS0	Detailed description above. 0 Low output drive strength selected for Port x bit 0. 1 High output drive strength selected for Port x bit 0.

13.3.5 Port x Input Filter Enable Register (PCx_IFE)

The pad cells on this device incorporate optional, low-pass filters on the digital input functions. These are enabled by setting the appropriate bit in the input-filter-enable register (IFE[*n*]). When set, a low-pass filter (with a bandwidth of 10 MHz to 30 MHz) is enabled in the logic input path. When cleared, the filter is bypassed.

Address: Base address + 3h offset

Bit	7	6	5	4	3	2	1	0
Read Write	IFE7	IFE6	IFE5	IFE4	IFE3	IFE2	IFE1	IFE0
Reset	1	1	1	1	1	1	1	1

PCx_IFE field descriptions

Field	Description
7 IFE7	Detailed description above. 0 Input filter disabled for Port x bit 7 1 Input filter enabled for Port x bit 7
6 IFE6	Detailed description above. 0 Input filter disabled for Port x bit 6 1 Input filter enabled for Port x bit 6
5 IFE5	Detailed description above.

Table continues on the next page...

PCx_IFE field descriptions (continued)

Field	Description
	0 Input filter disabled for Port x bit 5 1 Input filter enabled for Port x bit 5
4 IFE4	Detailed description above. 0 Input filter disabled for Port x bit 4 1 Input filter enabled for Port x bit 4
3 IFE3	Detailed description above. 0 Input filter disabled for Port x bit 3 1 Input filter enabled for Port x bit 3
2 IFE2	Detailed description above. 0 Input filter disabled for Port x bit 2 1 Input filter enabled for Port x bit 2
1 IFE1	Detailed description above. 0 Input filter disabled for Port x bit 1 1 Input filter enabled for Port x bit 1
0 IFE0	Detailed description above. 0 Input filter disabled for Port x bit 0 1 Input filter enabled for Port x bit 0

Chapter 14
Rapid General Purpose Input/Output Module (RGPIO)

14.1 Introduction

The Rapid GPIO (RGPIO) module provides a 16-bit general-purpose I/O module directly connected to the processor's high-speed 32-bit local bus. This connection plus support for single-cycle, zero wait-state data transfers allows the RGPIO module to provide improved pin performance when compared to more traditional GPIO modules located on the internal slave peripheral bus.

Many of the pins associated with a device may be used for several different functions. Their primary functions are to provide external interfaces to access off-chip resources. When not used for their primary function, many of the pins may be used as general-purpose digital I/O (GPIO) pins. The definition of the exact pin functions and the affected signals is specific to each device. Every GPIO port, including the RGPIO module, has registers that configure, monitor, and control the port pins.

Note

Most pin functions default to GPIO and must be software configured before using RGPIO.

14.1.1 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 the following figure. The details of the pin muxing and pad logic are device -specific.

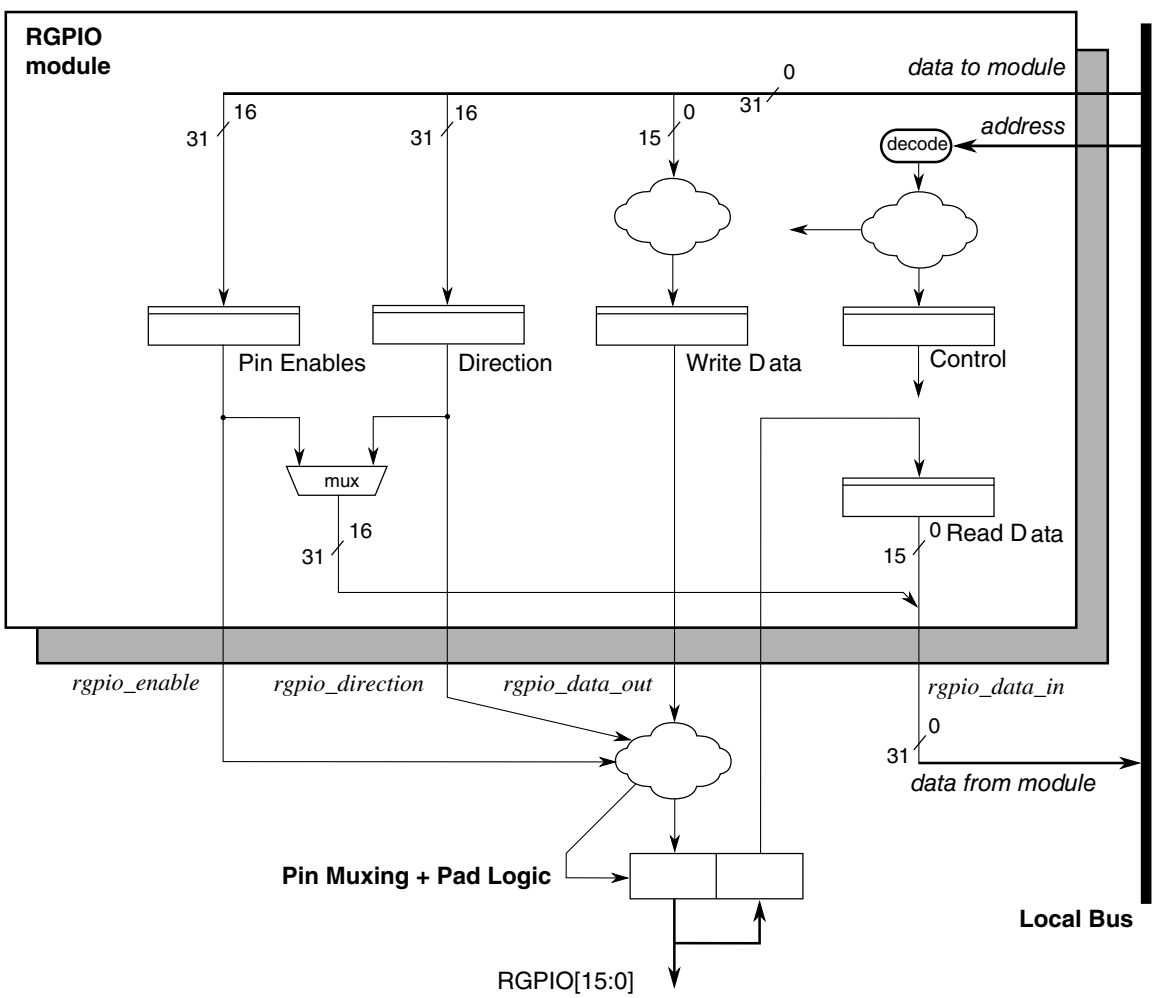


Figure 14-1. RGPIO Block Diagram

14.1.2 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)

14.1.3 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 CPU operating mode (supervisor, user) of its references.

14.2 External Signal Description

14.2.1 Overview

As shown in Figure 14-1, the RGPIO module's interface to external logic is indirect via the device pin-muxing and pad logic. The following table shows a list of the associated RGPIO input/output signals.

Table 14-1. RGPIO Module External I/O Signals

Signal Name	Type	Description
RGPIO[15:0]	I/O	RGPIO Data Input/Output

14.2.2 Detailed Signal Descriptions

The following table provides descriptions of the RGPIO module's input and output signals.

Table 14-2. 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.

14.3 Memory Map and Registers

The RGPIO module provides a compact 16-byte programming model based at a system memory address of 0x(00)C0_0000 (noted as RGPIO_BASE throughout the chapter). The programming model views are different between reads and writes to enable simplified software for manipulating the RGPIO pins.

Additionally, the RGPIO programming model is defined with a 32-bit organization. The basic size of each program-visible register is 16 bits, but the programming model may be referenced using byte (8-bit), word (16-bit) or longword (32-bit) accesses. Performance is typically maximized using 32-bit accesses.

NOTE

Writes to the two-byte fields at RGPIO_BASE + 0x8 and RGPIO_BASE + 0xC are allowed, but do not affect any program-visible register within the RGPIO module.

RGPIO memory map

Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
C0_0000	RGPIO Data Direction Register (RGPIO_DIR)	16	R/W	0000h	14.3.1/231
C0_0002	RGPIO Data Register (RGPIO_DATA)	16	R/W	0000h	14.3.2/232
C0_0004	RGPIO Pin Enable Register (RGPIO_ENB)	16	R/W	0000h	14.3.3/233
C0_0006	RGPIO Clear Data Register (RGPIO_CLR)	16	W	Undefined	14.3.4/234
C0_0008	RGPIO Data Direction Register (RGPIO_DIR)	16	R	0000h	14.3.5/234
C0_000A	RGPIO Set Data Register (RGPIO_SET)	16	W	Undefined	14.3.6/235
C0_000C	RGPIO Data Direction Register (RGPIO_DIR)	16	R	0000h	14.3.7/235
C0_000E	RGPIO Toggle Data Register (RGPIO_TOG)	16	W	Undefined	14.3.8/236

14.3.1 RGPIO Data Direction Register (RGPIO_DIR)

The read/write RGPIO_DIR register defines whether a properly-enabled RGPIO pin is configured as an input or output:

- Setting any bit in RGPIO_DIR configures a properly-enabled RGPIO port pin as an output
- Clearing any bit in RGPIO_DIR configures a properly-enabled RGPIO port pin as an input

At reset, all bits in the RGPIO_DIR are cleared.

Address: C0_0000h base + 0h offset = C0_0000h

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read Write	DIR															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

RGPIO_DIR field descriptions

Field	Description
DIR	Data direction 0 A properly-enabled RGPIO pin is configured as an input. 1 A properly-enabled RGPIO pin is configured as an output.

14.3.2 RGPIO Data Register (RGPIO_DATA)

The read/write RGPIO_DATA register specifies the write data for a properly-enabled RGPIO output pin or the sampled read data value for a properly-enabled pin. An attempted read of the RGPIO_DATA register returns undefined data for disabled pins because the data value depends on the chip-level pin muxing and pad implementation. At reset, all bits in the RGPIO_DATA registers are cleared.

To set bits in the RGPIO_DATA register, directly set the RGPIO_DATA bits or set the corresponding bits in the RGPIO_SET register. To clear bits in the RGPIO_DATA register, directly clear the RGPIO_DATA bits or clear the corresponding bits in the RGPIO_CLR register. Setting a bit in the RGPIO_TOG register inverts (toggles) the state of the corresponding bit in the RGPIO_DATA register.

As shown in Figure 14-1, the *rgpio_data_in* value is registered using the contents of the *rgpio_data* input bus. For situations where the data direction specifies driving the pins from the *rpgio_data_out* register, the *rgpio_data_in* register value is delayed by one cycle following an update of the write data register. This consecutive sequence is an inherent read-after-write data hazard that can occur with consecutive BCHG instructions and any other operand read following an operand write. Application code must be aware that operand reads of the RGPIO_DATA register immediately (on the next cycle) after an operand write to the RGPIO_DATA register may return a stale value. To prevent this consecutive cycle read-after-write sequence and the corresponding stale data value in RGPIO_DATA, insert a "TPF" instruction (opcode = 0x51FC) between the write and read operations to guarantee the updated value is read.

Address: C0_0000h base + 2h offset = C0_0002h

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read Write	DATA															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

RGPIO_DATA field descriptions

Field	Description
DATA	RGPIO data 0 A properly-enabled RGPIO output pin is driven with a logic 0, or a properly-enabled RGPIO pin was read as a logic 0. 1 A properly-enabled RGPIO output pin is driven with a logic 1, or a properly-enabled RGPIO pin was read as a logic 1.

14.3.3 RGPIO Pin Enable Register (RGPIO_ENB)

The read/write RGPIO_ENB register configures the corresponding package pin as an RGPIO pin instead of the normal GPIO pin mapped onto the peripheral bus.

At reset, all bits in the RGPIO_ENB register are cleared, disabling the RGPIO functionality.

Address: C0_0000h base + 4h offset = C0_0004h

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read Write	ENB															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

RGPIO_ENB field descriptions

Field	Description
ENB	Enable pin for RGPIO 0 The corresponding package pin is configured for use as a normal GPIO pin, not an RGPIO pin. 1 The corresponding package pin is configured for use as an RGPIO pin.

14.3.4 RGPIO Clear Data Register (RGPIO_CLR)

The RGPIO_CLR register provides a mechanism to clear specific bits in the RGPIO_DATA by performing a simple write. Clearing a bit in RGPIO_CLR clears the corresponding bit in the RGPIO_DATA register. Setting it has no effect. The RGPIO_CLR register is write-only; reads of this address return the RGPIO_DATA register.

Address: C0_0000h base + 6h offset = C0_0006h

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read																
Write	CLR															
Reset	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*

* Notes:
- x = Undefined at reset.

RGPIO_CLR field descriptions

Field	Description
CLR	Clear data 0 Clears the corresponding bit in the RGPIO_DATA register. 1 No effect.

14.3.5 RGPIO Data Direction Register (RGPIO_DIR)

Reading this read-only register returns the value of the RGPIO_DIR register at offset 0h.

Address: C0_0000h base + 8h offset = C0_0008h

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	DIR															
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

RGPIO_DIR field descriptions

Field	Description
DIR	Data direction 0 A properly-enabled RGPIO pin is configured as an input. 1 A properly-enabled RGPIO pin is configured as an output.

14.3.6 RGPIO Set Data Register (RGPIO_SET)

The RGPIO_SET register provides a mechanism to set specific bits in the RGPIO_DATA register by performing a simple write. Setting a bit in RGPIO_SET asserts the corresponding bit in the RGPIO_DATA register. Clearing it has no effect. The RGPIO_SET register is write-only; reads of this address return the RGPIO_DATA register.

Address: C0_0000h base + Ah offset = C0_000Ah

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read																
Write	SET															
Reset	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*

* Notes:
- x = Undefined at reset.

RGPIO_SET field descriptions

Field	Description
SET	Set data 0 No effect. 1 Sets the corresponding bit in the RGPIO_DATA register.

14.3.7 RGPIO Data Direction Register (RGPIO_DIR)

Reading this read-only register returns the value of the RGPIO_DIR register at offset 0h.

Address: C0_0000h base + Ch offset = C0_000Ch

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	DIR															
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

RGPIO_DIR field descriptions

Field	Description
DIR	Data direction 0 A properly-enabled RGPIO pin is configured as an input. 1 A properly-enabled RGPIO pin is configured as an output.

14.3.8 RGPIO Toggle Data Register (RGPIO_TOG)

The RGPIO_TOG register provides a mechanism to invert (toggle) specific bits in the RGPIO_DATA register by performing a simple write. Setting a bit in RGPIO_TOG inverts the corresponding bit in the RGPIO_DATA register. Clearing it has no effect. The RGPIO_TOG register is write-only; reads of this address return the RGPIO_DATA register.

Address: C0_0000h base + Eh offset = C0_000Eh

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read																
Write	TOG															
Reset	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*

* Notes:
- x = Undefined at reset.

RGPIO_TOG field descriptions

Field	Description
TOG	Toggle data 0 No effect 1 Inverts the corresponding bit in RGPIO_DATA

14.4 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).

14.5 Initialization Information

The reset state of the RGPIO module disables the entire 16-bit data port. Prior to using the RGPIO port, software typically:

- Defines the contents of the data register (RGPIO_DATA) if the pin is to be an output
- Configures the pin direction in RGPIO_DIR
- Enables the appropriate pins in RGPIO_ENB

14.6 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).

14.6.1 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. When using back to back BCHG instructions, insert the TPF instruction between the BCHG instructions.
- 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 the following table. 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 14-12. 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) × *f* MHz	2.083 MHz	1.00x	(1/14) × *f* MHz	3.571 MHz	1.71x
set+clr (+toggle)	(1/12) × *f* MHz	4.167 MHz	2.00x	(1/8) × *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.

14.6.2 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 the following figure.

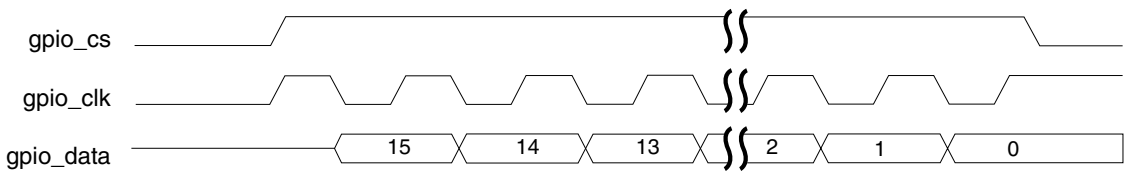


Figure 14-10. GPIO SPI Example Timing Diagram

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 the following example.

```
# subtest: send a 16-bit message via a SPI interface using a RGPIO
                      # the SPI protocol uses a 3-bit value: clock, chip-select, data
                      # the data is centered around the rising-edge of the clock

                              align   16
                      send_16b_spi_message_rgpio:
00510: 4fef fff4               lea     -12(%sp),%sp        # allocate stack space
00514: 48d7 008c               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                         # preserve 50% duty cycle
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   &0xfff5,%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
```

The resulting SPI performance, as measured in the effective Mbps transmission rate for the 16-bit message, is shown in the following table.

Table 14-13. 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 15
Pin Interrupt Function (IRQ)

15.1 Chip-specific information about IRQ

15.1.1 Programming IRQ pin functionality

When pin A4 is programmed as an interrupt, the IRQ function must also be enabled by setting IRQ_SC[IRQPE]. When operated as an interrupt, the pullup/down enable is controlled by IRQ_SC[IRQPDD] instead of PT1PE[PE4]. At the same time, if IRQ_SC[IRQPDD] is enabled and IRQ_SC[IRQEDG] is zero, a pullup is used. If IRQ_SC[IRQEDG] is one, a pulldown is used.

15.1.2 IRQ operation in and exit from stop modes

Subject to settings of the SIM peripheral clock enable registers, the IRQ module remains active in $STOP_{FC}$ and $STOP_{SC}$ modes. For more information, see Stop modes.

The IRQ interrupt can provide a means to exit $STOP_{FC}$, $STOP_{SC}$, and $STOP_{NC}$ modes. For more information, see Exit from Low-Power Modes.

15.2 Introduction

The IRQ (External Interrupt) module provides an interrupt input.

15.2.1 Features

The IRQ includes these distinctive features:

- IP Bus V2.0 compliant
- External interrupt pin (IRQ)
- IRQ pin can be selected as falling edge and low level or rising edge and high level
- Separate IRQ pin enable
- Software enabled interrupt
- Programmable falling edge (or rising edge) only, or both falling edge and low level (or both rising edge and high level) interrupt sensitivity
- Exit from low-power modes
- Software control of whether on-chip pullup/pulldown device is enabled on IRQ pin

15.2.2 Modes of Operation

The IRQ module is mode independent and will continue to operate in all user modes. In the low power STOP mode, the IRQ input becomes an asynchronous path.

15.2.3 Block Diagram

The following is a block diagram of the IRQ module.

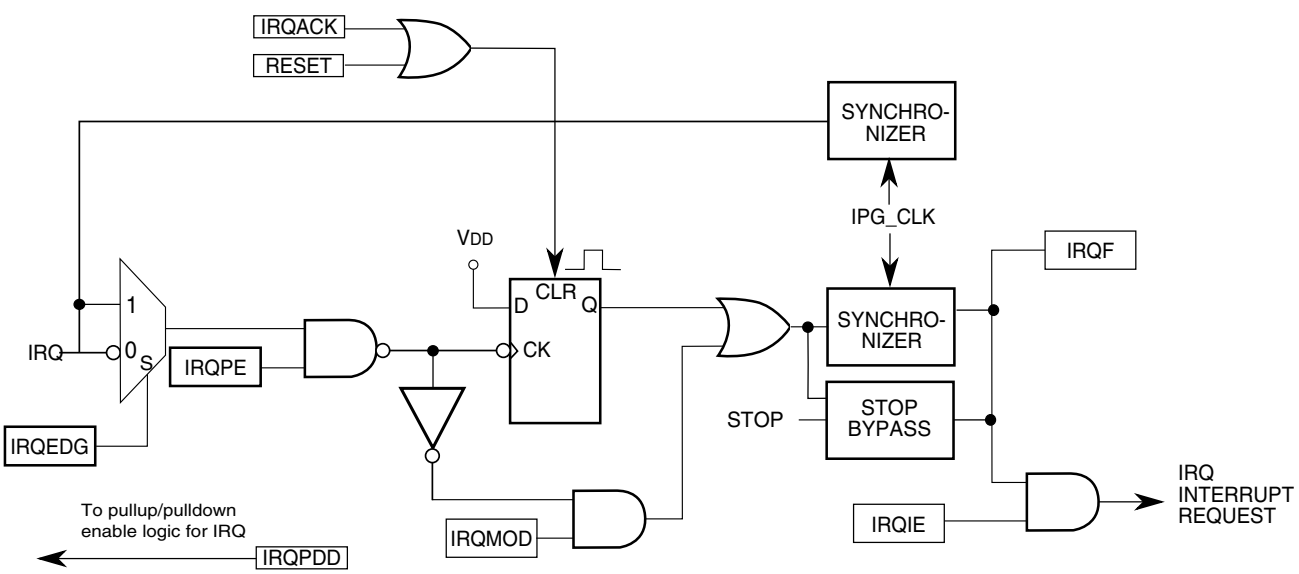


Figure 15-1. IRQ Block Diagram

15.3 Signal Description

The following table shows the user-accessible signal for the IRQ module.

Table 15-1. Signal Properties

Name	Function	Reset State
IRQ	External interrupt pin	input

15.3.1 Detailed Signal Descriptions

This section describes each user-accessible pin.

1. IRQ — External interrupt input pin

   This input pin is used to detect either falling edge, or both falling edge and low level interrupt requests. This input pin can also be used to detect either rising edge, or both rising edge and high level interrupt requests.

15.4 Memory Map and Register Description

This section provides a detailed description of the IRQ register that is accessible to the end user.

IRQ memory map

Address offset (hex)	Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
0	FFFF_80C0	Interrupt status and control register (IRQ_SC)	8	R/W	00h	15.4.1/244

15.4.1 Interrupt status and control register (IRQ_SC)

Address: FFFF_80C0h base + 0h offset = FFFF_80C0h

Bit	7	6	5	4	3	2	1	0
Read	0	IRQPDD	IRQEDG	IRQPE	IRQF	0	IRQIE	IRQMOD
Write						IRQACK		
Reset	0	0	0	0	0	0	0	0

IRQ_SC field descriptions

Field	Description
7 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
6 IRQPDD	IRQ pull device disable Use this bit to disable the on-chip pullup/pulldown device on the IRQ pin. This allows users to have an external device if required for their application. 0 On-chip pullup/pulldown device is enabled 1 On-chip pullup/pulldown device is disabled
5 IRQEDG	IRQ edge select This bit selects the falling edge/low level or rising edge/high level function of the IRQ pin. 0 Falling edge/low level 1 Rising edge/high level
4 IRQPE	IRQ pin enable This bit determines whether the IRQ pin is enabled. 0 IRQ pin not enabled 1 IRQ pin enabled
3 IRQF	IRQ flag This bit indicates when an IRQ interrupt is detected. 0 No IRQ interrupt detected 1 IRQ interrupt detected
2 IRQACK	IRQ acknowledge Writing 1 to this bit is part of the flag clearing mechanism. For more information about flag clearing, see Clearing an IRQ Interrupt Request. This bit always reads as 0.
1 IRQIE	IRQ interrupt enable This bit determines whether an IRQ interrupt request is enabled. 0 IRQ interrupt requests not enabled 1 IRQ interrupt requests enabled
0 IRQMOD	IRQ detection mode This bit (along with the IRQEDG bit) controls the detection mode of the IRQ pin.

Table continues on the next page...

IRQ_SC field descriptions (continued)

Field	Description
	0 IRQ interrupt requests on falling edge only or on rising edge only 1 IRQ interrupt requests on falling edge and low level or on rising edge and high levels

15.5 Functional Description

This section provides a complete functional description of the IRQ module.

15.5.1 External Interrupt Pin

Writing to the IRQPE bit in the SC register, enables or disables the IRQ pin.

15.5.2 IRQ Edge Select

The IRQEDG bit in the SC register determines if the IRQ pin is either falling edge and low level or rising edge and high level sensitive.

15.5.3 IRQ Sensitivity

The IRQMOD bit in the SC register controls the detection mode of the IRQ module.

- If the IRQ interrupt is falling (or rising) edge sensitive only, a falling (or rising) edge on the enabled IRQ pin will set the IRQF bit.
- If the IRQ interrupt is both falling (or rising) edge and low (or high) level sensitive, a falling (or rising) edge on the enabled IRQ sets the IRQF bit. The IRQF bit remains set as long as the IRQ pin remains asserted.

15.5.4 IRQ Interrupts

The IRQ module can provide a source of interrupts. To cause an IRQ module interrupt request, the following must occur:

- The IRQIE bit in the SC register must be set.

- The IRQF bit in the SC register must become set by a triggered IRQ pin. The IRQF bit becomes set by the fifth clock cycle after the IRQ pin has become asserted.
- The IRQ pin must have been in an inactive state for at least one clock cycle before becoming active.
- Changing the IRQMOD or IRQEDG bit while the IRQPE bit is enabled may cause a spurious interrupt and the IRQF bit may be inadvertently cleared.

15.5.5 Clearing an IRQ Interrupt Request

If the IRQ module interrupt pin is either falling edge and low level sensitive or rising edge and high level sensitive, both of the following actions must occur to clear an IRQ interrupt request:

- Software provides an interrupt acknowledge by writing a logic 1 to the IRQACK bit in the SC register.
- And either of the following:
  - The IRQ pin returns to a deasserted logic state.
  - The IRQ pin is disabled using the IRQPE bit.

If the IRQ module interrupt pin is falling (or rising) edge sensitive only, writing a logic 1 to the IRQACK bit in the SC register immediately clears the IRQ interrupt request even if the enabled IRQ pin remains asserted.

15.5.6 Exit from Low-Power Modes

The IRQ interrupt, if enabled, can provide a means to exit CPU low-power modes. If the IRQ pin is enabled and asserted upon entering a low-power mode and the detection mode is set to both falling edge and low level sensitivity or both rising edge and high level sensitivity, an immediate exit from the low-power mode may occur depending on the specific chip implementation. If the detection mode is set to falling or rising edge sensitivity only, an edge must be seen on the enabled IRQ pin to exit the low-power mode.

15.5.6.1 Stop modes

The IRQ module can remain active in stop modes, depending on the chip implementation. Setting the IRQIE bit in the SC register enables the IRQ interrupt request. Any detected IRQ interrupt brings the CPU out of the stop mode.

15.6 Resets

The IRQ interrupt is disabled after reset. The IRQ module cannot cause an MCU reset.

15.7 Interrupts

The IRQ module generates a single interrupt.

The IRQ interrupt is listed in the following table, which shows the interrupt name and the name of the local enable that can be used to disable a IRQ interrupt request.

Table 15-4. Interrupt Summary

Interrupt	Local Enable	Source	Description
IRQF	IRQIE	IRQ input	Software programmable for falling edge only (or rising edge only), or both falling edge and low level detection (or both rising edge and high level detection).

Chapter 16
Read Only Memory (ROM)

16.1 Introduction

The on-chip ROM is used to boot the device, and includes a slave-port command interpreter, utilities for flash and basic device functions, and user-callable functions.

There are several classes of functions stored in ROM:

- A boot program, including a ROM-based, slave-port command interpreter
- A collection of utilities, which can be invoked via the ROM-based slave port interpreter
- ROM functions, which are callable from user code using the call_trap() function

ROM code can only be executed when the CPU is in Supervisor mode. Any attempt to access the ROM while in User mode will result in a privilege violation exception. Error exceptions arising from User-mode attempts to access Supervisor-only resources will cause a soft-reset of the device.

16.2 Boot ROM

The device boots from ROM before handing off control to user code in flash memory. There are six main steps in the boot process. You can also use the ROM utilities to program or erase the flash memory, and additional limited functions are also available.

The FXLC95000 boots from a standard routine in ROM. This boot function (shown in Figure 16-1) is responsible for a number of initialization steps before transferring control to user code in flash memory. The ROM also contains a simple command interpreter capable of running a number of utility and test functions for programming and erasing flash memory, as well as a limited set of other functions.

Individual steps shown in Figure 16-1 are described in more detail in subsequent sections. One common theme is the use of the Flash Options Register (FOPT). This register is not visible to software operating in User mode on the ColdFire core. Normally, it is accessed only by supervisor code operating out of the on-chip ROM.

One of the functions of FOPT is to configure boot options for the device. These are normally fetched once at power-up from the locations 0x0001_FFFE and 0x0001_FFFF. FOPT bits control the security state of the device, such as whether or not a mass-erase operation is pending (required to clear device security) and whether the part is to boot to flash or to the ROM-based slave-port command interpreter. For a flash boot, the FOPT also controls whether a checksum is calculated prior to transferring control to flash and determines what is done if a checksum fails.

Because FOPT[15:8] is initialized only at power-up, it can be manipulated by the slave-port command interpreter and BDM to reconfigure device operation on subsequent reset operations.

For fielded applications, the normal control flow for the boot function is 1-2-3-4-5-10. (See Figure 16-1.) Other options are intended primarily for debug and development purposes.

16.2.1 Boot Step 1: RESET

Any hardware- or software-initiated reset will return the device to this phase (RESET). Hardware logic on the chip is returned to its default state.

During this phase (RESET), FOPT[7:0] (which includes the device's security state) is reloaded from location 0x0001_FFFF in flash memory. If the reset is a result of a power-on sequence, then FOPT[15:8] will be initialized to all 0s, and then BF, MECFB, and CHECKB bits will be loaded from 0x0001_FFFE. FOPT[15:8] register bits are not affected by subsequent reset operations (except for POR); FOPT[15:8] bits are used to coordinate boot and flash operations across reset sequences.

Erased devices eventually boot into the ROM-based command interpreter. Prior to entering that phase, slave port mailboxes are loaded with the equivalent of a CI_DEV_INFO response packet (see CI_DEV_INFO for additional details).

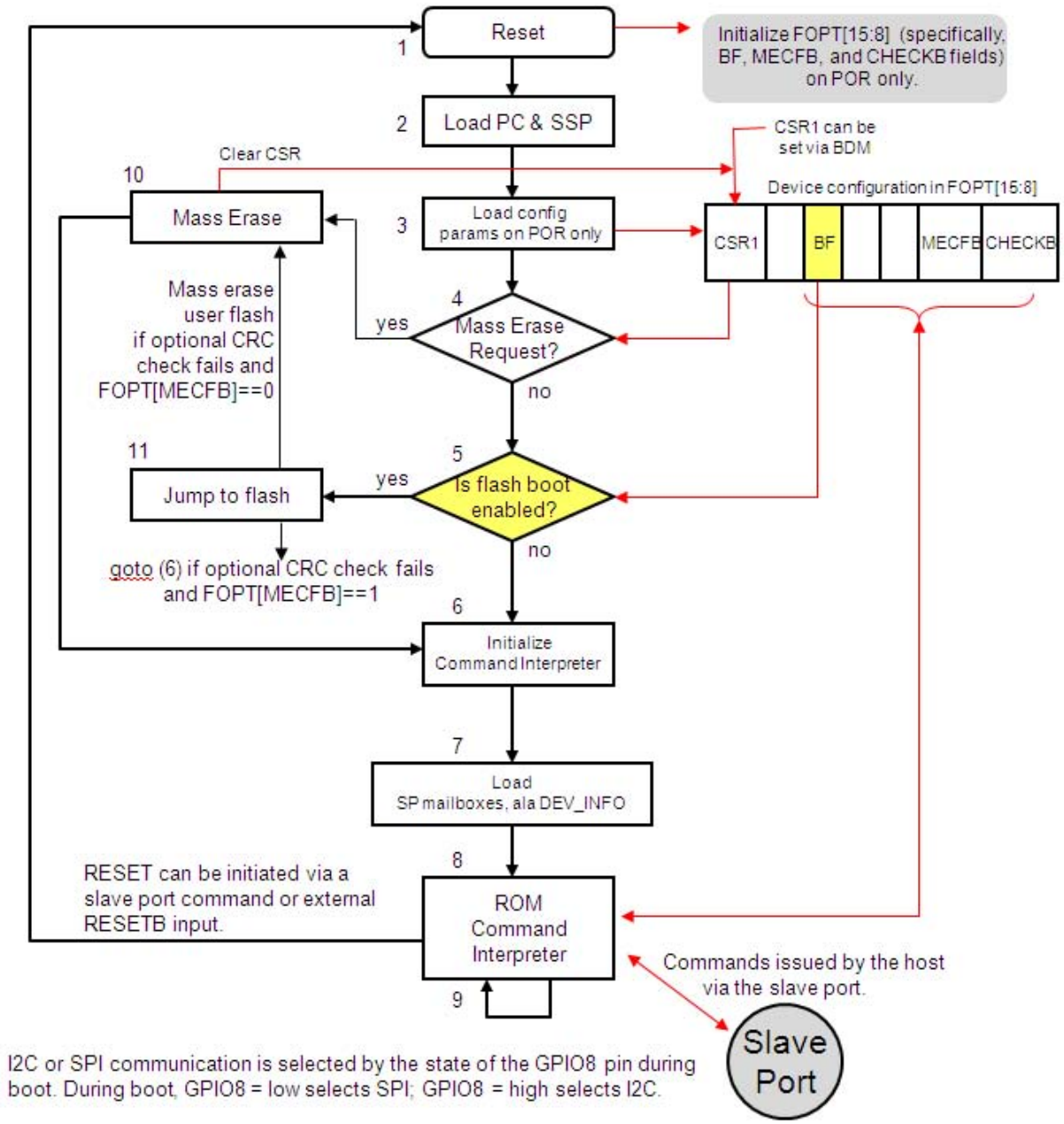


Figure 16-1. Flow diagram for ROM boot routine

16.2.2 Boot Step 2: Load PC and SSP

The Version 1 ColdFire CPU will load the program counter and supervisor-stack pointer from the first two long-words in ROM. The program execution in ROM begins; start-up code initializes the status register to 0x2700 and sets the Vector Base Register (VBR) to point to the beginning of ROM (0x300000).

16.2.3 Boot Step 3: Load configuration parameters

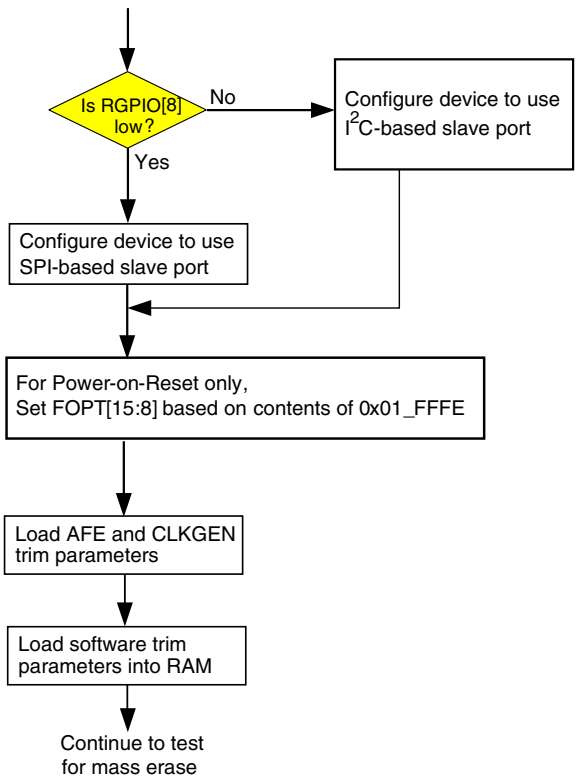


Figure 16-2. Boot Step 3: Load configuration parameters

Subsequent to reset, configuration parameters are read from reserved locations in flash and are stored in a variety of control registers in the memory map. These include the AFE trim and CLKGEN trim registers. Software trim parameters are copied from flash to RAM, in anticipation of later use.

For power-up sequences only:

- FOPT[BF] is set to the inverse of Bit 5 of memory location 0x0001_FFFE in flash. This bit controls whether or not control is transferred to flash in Step 5.
- The FOPT[MECFB,CHECKB] data is loaded from location 0x0001_FFFE in flash.

16.2.4 Boot Step 4: Mass erase request

Flash program/erase operations are implemented via ROM code. It is possible for a debugger to request a mass erase of the device, by setting FOPT[CSR1]. After setting the bit, a soft reset can begin. The boot code sees that FOPT[CSR1] has been set, indicating a

request for mass erase, and then transfers control to step 10 (mass erase). If no request for mass erase is made, then control is transferred to step 5 (check for flash boot request). See Figure 16-3.

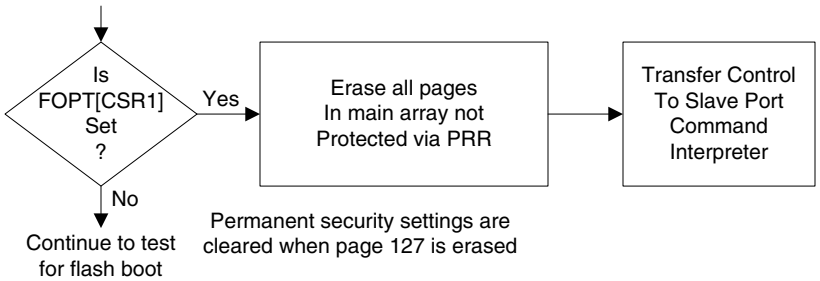


Figure 16-3. Boot Step 4: Mass erase request

16.2.5 Boot Steps 5 and 11: For flash boots, jump to flash

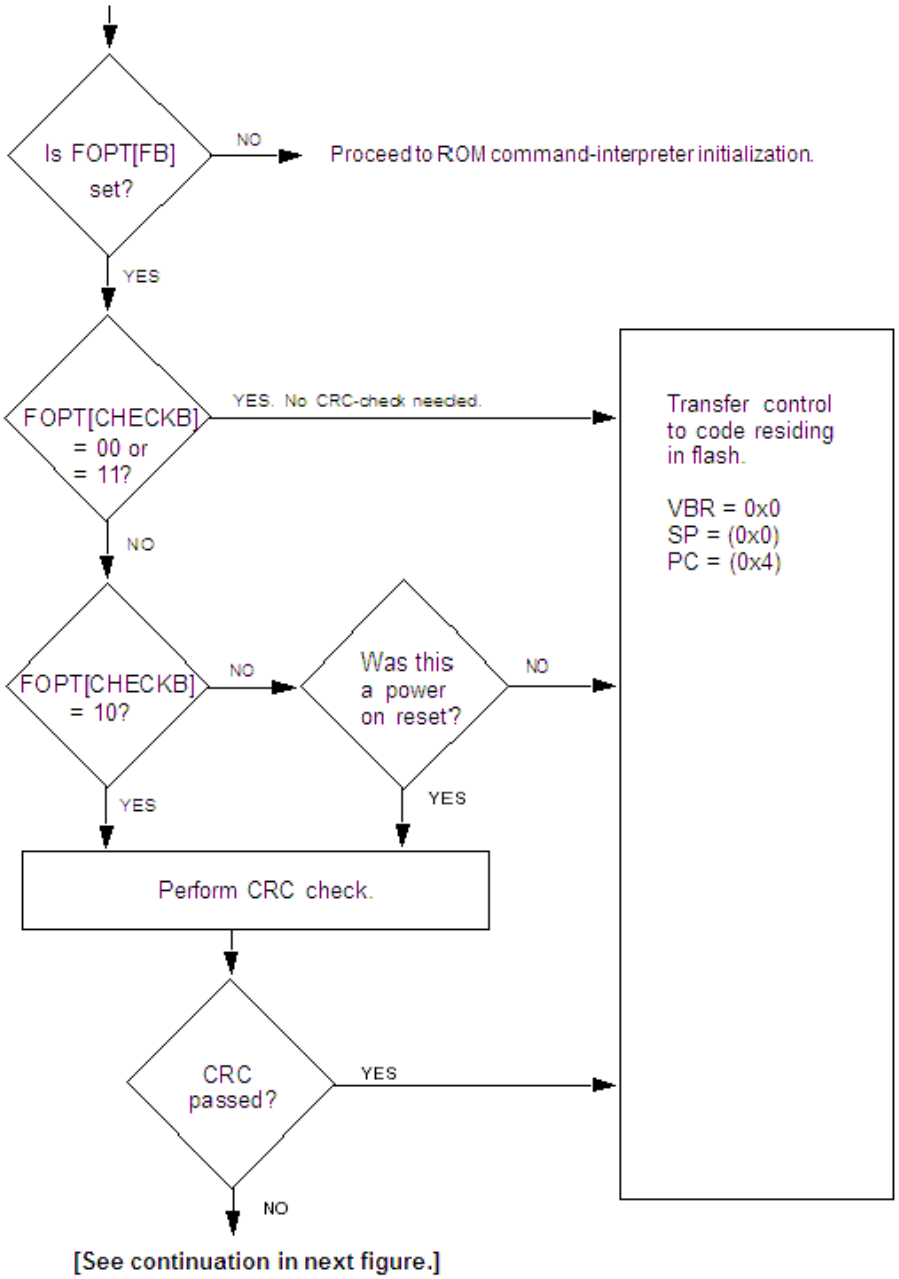


Figure 16-4. Boot-to-flash and associated checks (Part 1)

If FOPT[BF] has been set, then the boot code assumes that the flash is in a programmed state. The boot code checks FOPT[CHECKB] to determine if a CRC check needs to be run (to confirm the flash image). If no check is needed or if a check is run and succeeds, then control is transferred to the address specified at location 0x(00)00_0000 in flash memory.

The supervisor stack pointer is re-initialized to the address contained at location 0x(00)00_0000. The ColdFire Vector Base Register (VBR) is reset to 0x(00)00_0000. If FOPT[FB] has not been set, then control is transferred to Step 5 (Initialize Command Interpreter). If the CRC check fails and FOPT[MECFB] is set, then the device will be subjected to Mass Erase of User Portion Flash, and control is then transferred to Boot Step 6: Initialize command interpreter.

The "transfer control" block (above) transfers control to code located in flash memory, by performing the following functions:

- Resets the Vector Base Register to 0x(00)00_0000
- Reloads the supervisor stack pointer from the value stored at 0x(00)00_0000 in flash
- Reloads the program counter from location 0x(00)00_0004

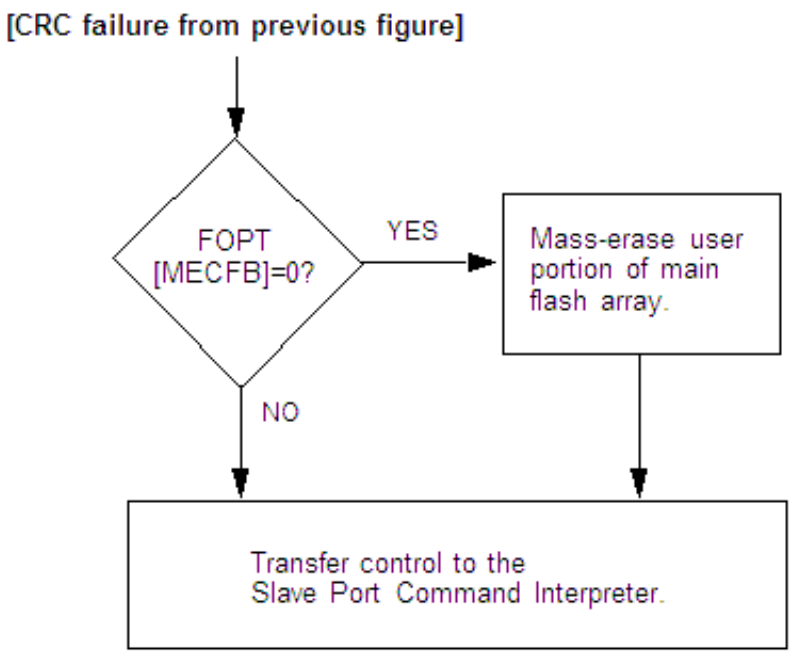


Figure 16-5. Boot-to-flash and associated checks (Part 2)

16.2.6 Boot Step 6: Initialize command interpreter

This step initializes RAM variables and hardware configuration, for use by the ROM command interpreter (Step 6).

The stack pointer is reset on each loop through the command interpreter.

16.2.7 Boot Step 8: Launch ROM command interpreter

This function continuously monitors the slave port for commands submitted serially via that port. The operation is single-threaded.

The port is monitored until a command is entered. An entered command is executed and the interpreter returns to the monitor loop. However, if the entered commands include a reset command, then the state machine restarts at Step 1.

Commands are submitted and statuses returned via the slave-port mailbox registers. See related details in ROM Command Interpreter (CI).

RGPIO8 = LOW at start-up indicates that the SPI should be used as a slave instead of I^2C. This is a function of the application boot code, and not a function of the hardware. The boot routine needs to read the RGPIO8 input value and act accordingly.

16.3 Security

Features in the boot ROM prevent the software that is running from flash from accidentally programming or erasing the flash.

16.3.1 Access and security rules of thumb

- PROTB protects against accidental program/erase by software running on-chip, or via the ROM command interface. PROTB does not prevent mass-erase via BDM.
- The Page-Release Register (PRR) (PRR_F0-3 and PRR_P0-3) allocates the pages of the flash array to be used by Freescale code and by user application code. Pages assigned to Freescale are protected from accidental erasure and can only be erased under tightly controlled conditions.
- Mass erase operation requests supply a starting address of 0x0000_0000 and an ending address of 0x0001_FFFF.
- The following resources are restricted to use in Supervisor mode:
  - ROM code
  - AFE registers
  - Flash-controller registers
  - CLKGEN trim registers
- Asserting security stops almost all access via the BDM and slave ports. The only supported operations in a secure state are RESET, MASS ERASE, and GET DEVICE INFO.

16.3.2 Security

You can secure your code from prying eyes by writing a secure code to NVOPT in the flash array. When the part is subsequently reset, access to the BDM development port is disabled, and ROM-based slave-port access is severely restricted.

Security may be cleared:

- by mass-erasing the device, via BDM by setting XCSR[ERASE] (see Extended Configuration/Status Register (XCSR)), and resetting the device. The ROM boot code will then erase all application pages in flash memory where (PRR=1), regardless of the setting of the flash-protection bit, (FOPT [PROTB]).
- by mass-erasing the device via the slave-port interface. In such cases (as is the case for software running on-chip), it is necessary first to set FOPT[PROTB]= 1 using the flash-unprotect function. See RMF_FLASH_PROTECT and RMF_FLASH_UNPROTECT.

If an attempt is made to read/write any on-chip memory while the device is in a secure state, then the ROM-based slave-port functions will fail and return a security violation.

16.4 Rights management

Use of the on-chip ROM is limited to Freescale use only. User access to restricted functions (in the ROM) is managed using the Device ID (DID) and a Page Release Register (PRR).

16.4.1 Memory-map restrictions

This section describes generic techniques for managing user access to restricted functions.

The FXLC95000 platform is designed to accommodate a varying mix of Freescale and third-party software. On-chip ROM is dedicated to Freescale use. The flash-memory array can be split between Freescale and third-party code.

Table 16-1. NVM memory allocations

Memory	Size	Freescale	Third-Party	Usage
ROM	16 KB	X	-	Boot functions, ROM command interpreter, flash-controller functions, common utilities. ROM code can be accessed only when the CPU is in Supervisor mode.
Main Flash Array	128 KB	X	X	There are 128 1K byte pages of flash memory. Any of these can be assigned to either Freescale or third-party use. All content is visible in User mode.

16.4.2 Rights-management variables

Non-volatile parameters used for rights management are shown in Table 16-2.

Table 16-2. Variables used for rights management

Register Name	Description	Bits 31-24	Bits 23-16	Bits 15-8	Bits 7-0
DID	Device ID	ID[31:0]			
PRR0	Page Release Register	PRR0[31:0] = PE[31:0]			
PRR1	(Factory Settings)	PRR1[31:0] = PE[63:32]			
PRR2		PRR2[31:0] = PE[95:64]			
PRR3		PRR3[31:0] = PE[127:96]			

16.4.2.1 Device ID (DID)

The Device ID provides a relatively unique identifier for any particular device. Freescale does not guarantee every unit to have a unique number. However, the DID field will vary from device to device.

16.4.2.2 Page-Release Register (PRR)

As previously mentioned, the main flash array on this device has 128 pages of 1K each. User programming/erase access to these pages is controlled via a virtual "Page-Release Register." The PRR is dynamically calculated by flash programming/erase firmware routines.

- There is one page-enable (PE) bit in the PRR for each page.
  - If PE is set to "0", then the page is allocated for Freescale use and will not be made available for customer programming.
  - If PE is set to "1", then the page is available for customer use.

- Bit 0 corresponds to the page beginning at address 0x0000_0000.
- Bit 31 corresponds to the page beginning at 0x0000_7C00.
- Bit 127 corresponds to the page beginning at 0x0001_FC00.

16.4.2.3 Hardware restrictions

The flash memory controller contains a non-volatile bit (FOPT[PROTB]), which can be used to protect flash memory from accidental programming/erase operations.

- The FOPT[PROTB] bit is sourced from the NVOPT location in flash memory on reset.
- The FOPT[PROTB] bit can be temporarily switched in and out via software.
- Various mechanisms for manipulating FOPT[PROTB] bit are described in the descriptions of the flash-access functions.

16.5 ROM Command Interpreter (CI)

The device includes a ROM Command Interpreter, so that user code can access a limited set of functions that reside in ROM. These functions support flash programming and erasing, the protection of flash, reset, the reading of device info, and also provide useful error codes.

16.5.1 Callable utilities

Functions available via the ROM Command Interpreter (CI) are summarized in Table 16-3. The Packet transfers and commands overview section provides a general overview of the user model associated with these functions. Subsequent sections provide the details of the individual functions.

Even on secured devices, it is possible to return the device ID and revision numbers and to change the flash-protection status. The latter does *not* waive security at all. Before attempting to mass-erase a secured device via the ROM command interpreter, **it is necessary to unprotect flash memory on that secured device**.

Table 16-3. Functions callable via ROM command interpreter

Command	Description	5-bit command code	Secure Mode Operation	Details
CI_DEV_INFO	Return device information	0x00	Allowed	CI_DEV_INFO

Table continues on the next page...

Table 16-3. Functions callable via ROM command interpreter (continued)

Command	Description	5-bit command code	Secure Mode Operation	Details
CI_READ_WRITE	Read/write memory (including flash programming)	0x01	Operation not performed. Security violation returned.	CI_READ_WRITE
CI_ERASE	Erase flash memory (page and mass-erase)	0x02	Mass-erase only	CI_ERASE
CI_CRC	Calculate CRC over memory range	0x04	Operation not performed. Security violation returned.	CI_CRC
CI_RESET	RESET	0x05	Allowed	CI_RESET
CI_PROTECT	Protect flash memory	0x07	Allowed	CI_PROTECT and CI_UNPROTECT
CI_UNPROTECT	Unprotect flash memory	0x08	Allowed	CI_PROTECT and CI_UNPROTECT
	All other command codes return the RMF_ERROR_COMMAND code (bad command).			

16.5.2 Packet transfers and commands overview

Most ROM-interpreter functions support transfer of two packets of information.

- One packet transfer is from the host to the slave, specifying the command to be executed and any required parameters.
- The second packet transfer is the response from the slave. The second transfer is optional in cases where the response carries only status information.

The Reset command has no return packet.

Mailbox registers on the FXLC95000CL transfer information to and from the command interpreter via the slave port. The following sections specify the function of each of the mailboxes on a per-command basis.

Many of the following sections includes one or more examples of how a specific command might be encoded in the data stream to and from a slave, I^2C port. These examples use a consolidated table format to document I^2C bit sequences.

These commands are easily mapped into standard I^2C waveforms, by noting use of the following notation:

- S - Start bit/Repeated start
- A - Acknowledge bit
- NACK - Not acknowledge bit
- P - Stop bit

In the "example" tables:

- Gray-shaded table cells indicate the bits written by the slave.
- Unshaded bits indicate the bits written by the master.

16.5.3 Common error codes

All command interpreter (CI) response packets utilize the same set of common return codes in the most-significant nibble of Mailbox 1. Bit 7 is used as "Command Complete" or "COCO." COCO is set to 0 when the command interpreter first recognizes the incoming command, and then COCO is set to 1 when the command is complete (with or without errors). COCO = 1 means that the command interpreter has done all it can with the command. Mailbox 1 bits 6-4 hold any applicable error code.

Table 16-4. Common CI error codes

Error Name	Error = Bits 6:4	Mailbox 1 MS Nibble	Description
PENDING	0x0 - 0x7	0x0 - 0x7	The command is still being executed.
RMF_ERROR_NONE	000	0x8	Command completed with no errors
RMF_ERROR_PARAM	001	0x9	An input parameter did not pass muster. Examples include: • Incorrect MEM field is supplied in CI read/write packet. • Erase password does not match RMF_ENABLE_FLASH_ERASE.
RMF_ERROR_PROT	010	0xA	Returned when an attempt is made to program or erase flash while flash protection is active (FOPT[PROTB] = 0). Call the CI function to unprotect flash before attempting to program/erase the flash.
RMF_ERROR_SECURITY	011	0xB	Returned when an attempt has been made to execute a prohibited function. Most CI commands are unavailable when security has been set (FOPT[SSC] = 10).
RMF_ERROR_VERIFY	100	0xC	Returned as a result of a PROGRAM or ERASE command if the final results of the operation do not match expected values. (ERASE values are all Fs. PROGRAM values are the input values.) The address offset of the first found error will be returned in mailboxes 2 and 3. This error only occurs when the VERF bit is set in the command byte.
RMF_ERROR_RIGHTS	101	0xD	Indicates that the user does not have access rights to perform a function, such as attempting to write to ROM.
RMF_ERROR_RANGE	110	0xE	Generally applicable to cases where an input parameter is not within an expected range of values. For example, a write command that attempts to program flash memory across physical rows of the device.
RMF_ERROR_COMMAND	111	0xF	Returned when the command interpreter does not recognize a command code or when an incomplete packet is recognized.

16.5.4 CI_DEV_INFO

This function returns the 32-bit device ID, along with ROM, flash and chip version numbers.

The Error Field of the Response Packet also returns a status code indicating whether or not the device is secure.

16.5.4.1 CI_DEV_INFO command packet format

The five-bit command code for the device info command is 0x00. The extension bits are 0.

Table 16-5. CI_DEV_INFO command packet format at mailbox level

Mailbox #	Description	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Command byte	0	0	0	0	0	0	0	0
1	Parameter byte	0	0	0	0	0	0	0	0
2-31	NOT USED	NOT USED							

16.5.4.2 CI_DEV_INFO response packet format

The first byte of the response packet contains the command packet previously sent.

The second byte is a general status byte. COCO is set to 1 when the command response is complete. The ERR field will be set to RMF_ERROR_SECURITY (0x3) if the device is in a secure state. This should be treated as a status indicator and not as an error, because other packet information will be correct, regardless of the security setting.

Additional mailboxes return:

- 32-bit device ID
- ROM software version number (ROM_MAJOR.ROM_MINOR)
- Freescale flash-based software version number (FT_FLASH_MAJOR.FT_FLASH_MINOR)
- Hardware version number (HW_MAJOR.HW_MINOR)
- Part Number, Reset Status and Security State

Table 16-6. CI_DEV_INFO response packet format at mailbox level

Mailbox #	Description	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Command byte	0	0	0	0	0	0	0	0
1	Status byte	COCO	ERR			0	0	0	0
2	ID MSB	ID[31:24]							
3	ID MSB+1	ID[23:16]							
4	ID MSB+2	ID[15:8]							
5	ID LSB	ID[7:0]							
6	ROM Major Version Number	ROM_MAJOR							
7	ROM Minor Version Number	ROM_MINOR							
8	Freescale Flash Code Major Version Number	FT_FLASH_MAJOR							
9	Freescale Flash Code Minor Version Number	FT_FLASH_MINOR							
10	Sensor Major Version Number	HW_MAJOR							
11	Sensor Minor Version Number	HW_MINOR							
14	Part number in BCD format	PART_NUMBER							
15									
16	RCSR	Value of RCSR at last reset							
17	Security state	0	0	0	0	0	0	FOPT[SSC]	
18-31	Not used	Not used							

NOTE

Values for FOPT[SSC] are: 00/01/11 unsecured, 10 secured.

NOTE

During boot, the ROM code puts the device ID, firmware version and other info into the mailboxes. After boot, the host processor can directly read mailbox registers to get the information, *without having to issue a specific command*.

16.5.4.3 CI_DEV_INFO access/security policies

Table 16-7 details security policies for the CI Return Device Info command.

Table 16-7. Access/Security policies for CI return device info command

Security Enabled	Security Disabled
Available	Available

16.5.5 CI_READ_WRITE

16.5.5.1 CI_READ_WRITE description

This function encapsulates all memory read/write functions, including those required for programming flash memory. Note that prior to any program operation, flash memory must be erased.

Memory-mapped components, RAM, ROM and flash memory can be read/written with a common set of memory-access sequences.

- Read commands require 8 mailbox locations.
- Write commands also require 8 locations, but with an additional payload of 0 to 24 bytes of write data stored in mailboxes 8 through 31.

Payload offsets map to on-chip addresses one-to-one. The first location accessed in the memory map corresponds to the value specified with the MEM and ADDR[15:0] parameters. Addresses are auto-incremented as the payload size increases.

NOTE

The 16-bit peripherals are restricted to word and long-word accesses on read and write. Flash is restricted to long words during programming sequences. The CI read/write commands are not responsible for checking that the packet structure has data packet sizes that are Modulo 2 or 4 for the various types. It is the responsibility of the user to make sure they are correct.

Read response packets are 2 mailboxes plus the payload in length. Write response packets consume 4 mailbox values.

16.5.5.2 CI_READ_WRITE memory command packet format

The five-bit command code for the read/write command is 0x01.

Table 16-8. Command packet format at mailbox level

Mailbox #	Description	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Command byte	0	0	0	0	1	0	VERF	TYPE
1	Parameter byte	MEM			NUMBER				
2	Command interpreter password	CI_PW[31:24]							
3		CI_PW[23:16]							

Table continues on the next page...

Table 16-8. Command packet format at mailbox level (continued)

Mailbox #	Description	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
4		CI_PW[15:8]							
5		CI_PW[7:0]							
6	Address bits [15:8]	ADDR[15:8]							
7	Address bits [7:0]	ADDR[7:0]							
8 - 31	Write data	WDATA							

Table 16-9. Command field descriptions

Field	Description
VERF	Verify Writes (not applicable in Read accesses) 0 Do not verify. 1 Verify that written value matches intent.
TYPE	Type of Access 0 Write 1 Read
MEM	Memory Space. Beyond those listed: All others are reserved. 000 Flash memory lower 64K 001 ROM (Valid CI_PW match required.) 010 RAM 011 RGPIO 100 8-bit peripherals 101 16-bit peripheral (Valid CI_PW match required.) 110 flash memory upper 64K All others are reserved.
NUMBER	Number of bytes to read/write. **NOTE:** Use of values exceeding these ranges may result in unexpected results. 1 to 24 for writes. Other values produce an error in the status packet. 1 to 30 for reads. Other values produce an error in the status packet. 0 NO-OP
CI_PW	Command Interpreter Password Certain restricted functions require a Freescale-supplied password to unlock access. The value of the CI_PW parameter is ignored for non-restricted functions (but the mailboxes still need to be consumed with a CI_PW value - even if that CI_PW value is not valid). See CI_READ_WRITE access/security policies.
ADDR	Address The lower 16-bits of the first memory address to be accessed. The upper bits are implied by the MEM variable.
WDATA	Write Data The NUMBER of bytes of data to be transferred in write command. Flash program packets must contain payloads that are multiples of 4 bytes.

16.5.5.3 CI_READ_WRITE read/write memory response packet format

There are two slightly different forms of the response packet: one form for reads, and one form for writes.

For reads:

- The first byte of the response packet contains the command packet previously sent.
- The second byte is a general status byte.
- Bytes 3 through 32 are optional, and contain data read from the internal memory map of the device.

Table 16-10. Read command response packet format at mailbox level

Mailbox #	Description	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Command byte	0	0	0	0	1	0	VERF	TYPE
1	Status byte	COCO	ERR			0	0	0	0
2-31	Read data	RDATA							

For writes:

- The first byte of the response packet contains the command packet previously sent.
- The second byte is a general status byte.
- Bytes 3 and 4 are optional, and contain data the first address at which a Verify error was detected (if VERF has been set).

Table 16-11. Write command response packet format at mailbox level

Mailbox #	Description	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Command byte	0	0	0	0	1	0	VERF	TYPE
1	Status byte	COCO	ERR			0	0	0	0
2	Verify error addr MSB	VERF_ERR_ADDR[15:8]							
3	Verify error addr LSB	VERF_ERR_ADDR[7:0]							

Table 16-12. Command response field descriptions

Field	Description
VERF	Verify Writes (not applicable in Read accesses) If a verify error is found, then the address at which the first error is detected will be written to mailboxes 2 to 5. 0 Do not verify 1 Verify that written value matches intent.
TYPE	Type of Access

Table continues on the next page...

Table 16-12. Command response field descriptions (continued)

Field	Description
	0 Write 1 Read
COCO	Command Complete Because flash program sequences take quite some time to complete, you may need to repeatedly poll the port before the operation completes. (The ERR flag will be set for aborted sequences.) 0 Previous command has not completed. 1 Previous command has completed or aborted.
ERR	Error Flag For the set of common CI error codes, see Table 16-4.
RDATA	Read Data If ERR = 000, then RDATA is the NUMBER of bytes of data transferred in the read command. If ERR is any other value, then the data contained in RDATA is not guaranteed.
VERF_ADDR	Verify Address[15:0] For write operations with verify, VERF_ADDR is the lower 16 bits of the first location in which a verify error was detected.

16.5.5.4 CI_READ_WRITE access/security policies

Table 16-13 details security policies for the CI Read/Write command.

Table 16-13. Access/Security policies for CI read/write memory command

	Security Enabled		Security Disabled	
	Read	Write	Read	Write
Main array of flash memory	No access		Allowed	Subject to PRR
RAM	No access		Allowed	
ROM	No access		CI_PW match required	Not allowed
16-bit peripherals	No access		CI_PW match required	
8-bit peripherals and RGPIO	No access		Allowed	

Policy descriptions are:

- Subject to PRR, writes to flash memory are restricted to those in which the PRR[page number] bit is "1." Flash protection must be disabled prior to any attempt at programming.
- CI_PW match is required. A valid command interpreter password must be supplied.

16.5.5.5 CI_READ_WRITE read/write memory example

This example does the following:

- Reads 4 bytes from RAM
- Starts at location 0x(00)80_0008
- Uses the I²C slave port, mapped to location 0x03 on the I²C bus

The Read packet must write 4 mailbox registers in the slave port.

Table 16-14. Command to read 4 bytes from RAM (starting at offset 0x08 for device 3 on I²C bus)

Start/Stop	7	6	5	4	3	2	1	0	ACK/ NACK
S	Slave address = 0x03							R/W = 0	A
	Register address = Mailbox #0 = 0x00								A
	Mailbox 0 = READ command = 0x09								A
	Mailbox 1 = "4 bytes from RAM" = 0x44								A
	Mailbox 2 = CI_PW[31:24]								A
	Mailbox 3 = CI_PW[23:16]								A
	Mailbox 4 = CI_PW[15:8]								A
	Mailbox 5 = CP_PW[7:0]								A
	Mailbox 6 = MSB of starting address = 0x00								A
	Mailbox 7 = LSB of starting address = 0x08								A
P									

The response packet uses the I²C "combined format" that is described in I²C message format for reading. This format combines a write (to establish the slave address and the first register address) and a read (of the six mailbox registers), to transfer the required data.

Table 16-15. Response to previous read command on I²C bus

Start/Stop	7	6	5	4	3	2	1	0	ACK/ NACK
S	Slave address = 0x03							R/W = 0	A
	Register address= 0x00								A
S	Slave address = 0x03							R/W = 1	A
	Mailbox 0 = Read command = 0x09								A
	Mailbox 1 = Status = 0x80 (command complete)								A
	Mailbox 2 = Data Byte from RAM Location (00)80_0008								A
	Mailbox 3 = Data Byte from RAM Location (00)80_0009								A

Table continues on the next page...

Table 16-15. Response to previous read command on I²C bus (continued)

Start/Stop	7	6	5	4	3	2	1	0	ACK/ NACK
	Mailbox 4 = Data Byte from RAM Location (00)80_000A								A
	Mailbox 5 = Data byte from RAM location (00)80_000B								NACK
P									

16.5.6 CI_ERASE

16.5.6.1 CI_ERASE erase flash function description

This function encapsulates all functions for page-erase and mass-erase actions of the flash memory.

- The command packet is 10 mailboxes in length.
- The response packets are 2 to 6 mailboxes in length.

User requests for mass-erase will honor protection provided by the PRR. Only pages whose PRR bit is 1 will be erased. Effectively, the mass-erase operation is translated on-the-fly to a series of page-erase operations.

NOTE

Page-erase *requests* are not supported for secured devices. A mass-erase must be *requested*.

The same function call encapsulates both page- and mass-erase operations. The ROM software will use mass-erase when possible, and use page-erase when mass-erase is not possible.

16.5.6.2 CI_ERASE erase command packet format

The five-bit command code for the erase command is 0x02.

Table 16-16. Command packet format at mailbox level

Mailbox #	Description	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Command byte	0	0	0	1	0	0	VERF	0
1	Parameter byte	PB = 0xC5							
2	Starting Address	START_ADDR[31:24]							
3		START_ADDR[23:16]							

Table continues on the next page...

Table 16-16. Command packet format at mailbox level (continued)

Mailbox #	Description	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
4		START_ADDR[15:8]							
5		START_ADDR[7:0]							
6	Ending Address	END_ADDR[31:24]							
7		END_ADDR[23:16]							
8		END_ADDR[15:8]							
9		END_ADDR[7:0]							
10-31	NOT USED	NOT USED							

Table 16-17. Command field descriptions

Field	Description
VERF	Verify Erase If a verify error is found, then the address at which the first error is detected will be written to mailboxes 2, 3, 4 and 5. 0 Do not verify 1 Verify that written value matches intent.
PB	Parameter Byte Constant value = 0xC5 Values other than 0xC5 will trigger a security error.
START_ADDR	The address for the first flash long word to be erased. • Legal values are those at the beginning of a page (ending in 0x000, 0x400, 0x800, 0xC00). • Erase operations are subject to usage rights previously established for the device. Some pages in flash memory may be dedicated to Freescale developed code. Erase requests for those pages will normally be rejected. See Page-Release Register (PRR).
END_ADDR	The address for last flash long word to be erased. • Legal values are those at the end of a page (ending in 0x3FF, 0x7FF, 0xBFF and 0xFFF). The main flash array on this device is composed of 128 1K-byte pages of memory. • A page is the minimum amount of flash memory that can be erased in a single operation. • Erase operations are subject to usage rights previously established for the device. Some pages in flash memory may be dedicated to Freescale developed code. Erase requests for those pages will normally be rejected. See Page-Release Register (PRR).

16.5.6.3 CI_ERASE erase command response packet format

The first byte of the response packet contains the command packet previously sent.

The second byte is a general status byte.

Table 16-18. Response packet format at mailbox level

Mailbox #	Description	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Command byte	0	0	0	1	0	0	VERF	0
1	Status byte	COCO	ERR			0	0	0	0
2	Verify error addr MSB	VERF_ERR_ADDR[31:24]							
3		VERF_ERR_ADDR[23:16]							
4		VERF_ERR_ADDR[15:8]							
5		VERF_ERR_ADDR[7:0]							
6 - 31	Not Used	Not Used							

Table 16-19. Command field descriptions

Field	Description
VERF	Verify Erase If a verify error is found, then the address at which the first error is detected will be written to mailboxes 2 to 5. 0 Do not verify. 1 Verify that written value matches intent.
COCO	Command Complete 0 Previous command was not completed. Because the flash program sequences take quite some time to complete, you may need to repeatedly poll the port before the operation completes. 1 Previous command has completed or aborted. (The ERR flag will be set for aborted sequences.)
ERR	Error Flag See Table 16-4 for the set of common CI error codes. Of those, the following error code interpretations apply to this device: RMF_ERROR RIGHTS - Issued when pages specified by the starting and ending addresses are not available to the user (i.e., they are protected by PRR). RMF_ERROR_SECURITY - The only erase operation allowed on a secured device is mass erase. RMF_ERROR_VERIFY - Some portion of the erasure was incomplete. RMF_ERROR_PARAMETER - If the internal password passed to FSL_flash_erase was invalid. RMF_ERROR_PROT - FOPT[PROTB] needs to be reset to 1 before erasing. RMF_ERROR_RANGE - Out-of-range addresses (starting or ending) were specified in the command packet.
VERF_ADDR	Verify Address[31:0] This is the 32-bit address of the first location in which a verify error was detected. This is only applicable if VERF is set and RMF_ERROR_VERIFY is returned in the ERR field.

16.5.6.4 CI_ERASE access/security policies

Table 16-20 details security policies for the CI Erase command.

Table 16-20. Access/security policies for CI erase flash command

	Security Enabled		Security Disabled	
	Page Erase	Mass Erase	Page Erase	Mass Erase
Upper portion of flash memory array	Not supported	Erased when (PB=0xC5 and Start Address = 0x00000000 and End Address = 0x0001FFFF)	Subject to PRR	Erased when (PB = 0xC5 and Start Address = 0x00000000 and End Address = 0x0001FFFF)

Policy descriptions are:

- Subject to PRR, erasures of flash memory are restricted to those in which the PRR[page number] bit is "1." The flash protection must be disabled prior to any attempts to page-erase.

16.5.6.5 CI_ERASE example

This example performs a mass-erase of the upper portion of the main array in flash memory.

The command packet must write 10 mailbox registers in the slave port.

Table 16-21. Command to mass erase flash on Device 3 on I^2C bus

Start/Stop	7	6	5	4	3	2	1	0	ACK/ NACK
S	Slave address = 0x03							R/W = 0	A
	Register address = Mailbox 0 = 0x00								A
	Mailbox 0 = Mass erase main-array-only command = 0x12								A
	Mailbox 1 = Parameter byte = 0xC5								A
	Mailbox 2 = 0x00								A
	Mailbox 3 = 0x00								A
	Mailbox 4 = 0x00								A
	Mailbox 5 = 0x00								A
	Mailbox 6 = 0x00								A
	Mailbox 7 = 0x01								A
	Mailbox 8 = 0xFF								A
	Mailbox 9 = 0xFF								A
P									

The response packet uses the I²C "combined format," which is described in I²C message format for reading This format combines a write (to establish the slave address and first register address) and a read (of mailbox registers) to transfer the required data. The table "Table 16-22" shows the case where only status information was retrieved. The diagnostic information in mailbox 2 and 3 was ignored.

Table 16-22. Response to previous mass erase command on I²C bus

Start/Stop	7	6	5	4	3	2	1	0	
S	Slave address = 0x03							R/W = 0	A
	Register address= 0x00								A
S	Slave address = 0x03							R/W = 1	A
	Mailbox 0 = Mass erase main-array-only command = 0x12								A
	Mailbox 1 = Status = 0x80 (command complete, no errors)								NACK
P									

16.5.7 CI_CRC

CodeWarrior has the ability to calculate a CRC over a range of code and then include it (the CRC) as part of the flash or ROM image. This function replicates the same algorithm, which can be used to confirm code integrity over time.

If the device has security enabled, then the CRC function will fail with a security violation.

16.5.7.1 CI_CRC checksum command packet format

The 5-bit command code for the CRC command is 0x04. The command packet requires 8 mailboxes and is shown in Table 16-23.

Table 16-23. Command packet format (RANGE=1, CS=1) at mailbox level

Mailbox #	Description	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Command byte	0	0	1	0	0	0	0	0
1	Parameter byte	MEM			RESERVED				
2	CRC Seed [15:8]	SEED[15:8]							
3	CRC seed [7:0]	SEED[7:0]							
4	Starting offset [15:8]	OFFL[15:8]							
5	Starting offset [7:0]	OFFL[7:0]							
6	Ending offset [15:8]	OFFH[15:8]							

Table continues on the next page...

Table 16-23. Command packet format (RANGE=1, CS=1) at mailbox level (continued)

Mailbox #	Description	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
7	Ending offset [7:0]	OFFH[7:0]							
8 - 31	NOT USED	NOT USED							

Table 16-24. Command field descriptions

Field	Description
MEM	Memory Space 000 Flash memory lower 64kB (Base Address 0x(00)00_0000) 001 ROM (Base Address 0x(00)30_0000) 010 RAM (Base Address 0x(00)80_0000) 110 Flash memory upper 64kB (Base Address 0x(00)01_0000) All others are reserved.
RESERVED	Reserved Bit Field Write as 0x00
SEED[15:0]	CRC Seed Value CRC calculations start with a known seed value. The recommended seed is 0x1D0F, although any value may be used.
OFFL[15:0]	Low Address Offset The base address of the memory + OFFL represents the first location in memory that will be accessed for the CRC calculation.
OFFH[15:0]	High Address Offset The base address of the memory + OFFH represents the last location in memory that will be accessed for the CRC calculation. OFFH must be greater than OFFL.

16.5.7.2 CI_CRC response packet format

The response packet for the CRC calculation has a length of 4 mailboxes.

These include:

- The first byte of the response packet, which contains the command packet previously sent.
- The second byte is a general status byte.
- Bytes 3 and 4 contain the signature calculated by the CRC function.

Table 16-25. Write command response packet format at mailbox level

Mailbox #	Description	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Command byte	0	0	1	0	0	0	1	1
1	Status byte	COCO	ERR			0	0	0	0
2	MSB of signature	SIG[15:8]							
3	LSB of signature	SIG[7:0]							

Table 16-26. Command field descriptions

Field	Description
COCO	Command Complete 0 Previous command was not completed. Test sequences take quite some time to complete. You may need to repeatedly poll the port before the operation completes. 1 Previous command has completed or aborted. For aborted sequences, the ERR flag will be set.
ERR	Error Flag If the number of bytes to perform the CRC on is less than 4 (end addr – start addr + 1), then the CRC function returns the RMF_ERROR_RANGE error. For the set of common CI error codes, see Table 16-4.
SIG	Signature[15:0] 16-bit signature calculated by the CRC function.

16.5.7.3 CI_CRC access/security policies

Table 16-27 details security policies for the CI_CRC command.

Table 16-27. Access/Security policies for CI_CRC command

Security Enabled	Security Disabled
Not Available	Available

16.5.7.4 CI_CRC example

This example calculates a CRC across the entire range of the ROM.

The command packet must write 8 mailbox registers in the slave port.

Table 16-28. CI_CRC I²C command packet to calculate the ROM CRC

Start/Stop	7	6	5	4	3	2	1	0	ACK/ NACK
S	Slave address = 0x03							R/W=0	A
	Register address = Mailbox 0 = 0x00								A
	Mailbox 0 = CRC command = 0x20								A
	Mailbox 1 = Test ROM = 0x20								A
	Mailbox 2 = MSB of Seed = 0x1D								A
	Mailbox 3 = LSB of Seed = 0x0F								A
	Mailbox 4 = 0x00								A
	Mailbox 5 = 0x00								A
	Mailbox 6 = 0x40								A
	Mailbox 7 = 0x00								A
P									

The minimum response packet uses the I²C "combined format", which is described in I²C message format for reading. This format combines a write (to establish the slave address and first register address) and a read (of the six mailbox registers) to transfer the required data.

Table 16-29. Response to previous read command on I²C bus

Start/Stop	7	6	5	4	3	2	1	0	ACK/ NACK
S	Slave address = 0x03							R/W = 0	A
	Register address= 0x00								A
S	Slave address = 0x03							R/W = 1	A
	Mailbox 0 = CRC command = 0x20								A
	Mailbox 1 = Status = 0x80 (command complete, no errors)								A
	Mailbox 2 = SIG[15:8]								A
	Mailbox 3 = SIG[7:0]								NACK
P									

16.5.8 CI_RESET

The Reset command configures FOPT[BF] to control flash/ROM Command Interpreter boot options, and initiates a reset by writing RCSR[SW] = 1. Because a hardware reset results from this operation, the RESET command has no response packet *unless* an error is encountered. In error cases, the "standard" 2-mailbox response packet is generated.

16.5.8.1 CI_RESET command packet format

The command packet requires 2 mailboxes.

The 5-bit command code for the reset command is 0x05.

Table 16-30. CI_RESET command packet format at mailbox level

Mailbox #	Description	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Command byte	0	0	1	0	1	DR	0	FL
1	RESERVED	0	0	0	0	0	0	0	0
2 - 31	NOT USED	NOT USED							

Table 16-31. Reset Command field descriptions

Field	Description
DR	DRIVE 0 Set RCSR[DR] = 0; RESETB pin is input only. 1 Set RCSR[DR] = 1; RESETB pin is driven low on device reset.
FL	Boot to Flash 0 Do not boot to flash. 1 Boot to flash.

The FL bit determines which address that the device boots with, on reset.

Table 16-32. Reset boot options

FL	Memory	Base Address
0	ROM	0x(00) 30_0000
1	Flash	0x(00) 00_0000

16.5.8.2 CI_RESET response packet format

The response packet for the CI_RESET command has a length of 2 mailboxes. These include:

- The first byte of the response packet contains the command packet previously sent.
- The second byte is a general status byte.

The response packet is only available when an error condition is found. If there are no error conditions, then the device performs a software reset without issuing a response packet.

Table 16-33. CI_RESET command response packet format at mailbox level

Mailbox #	Description	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Command byte	0	0	1	0	1	DR	0	FL
1	Status byte	COCO	ERR			0	0	0	0

In addition to the command byte parameters already described, the response packet includes standard COCO and ERR fields.

Table 16-34. CI_RESET response packet field descriptions

Field	Description
COCO	Command complete 0 Command has not finished 1 Command has finished
ERR	Error Flag For the set of common CI error codes, see Table 16-4.

16.5.8.3 CI_RESET access/security policies

Table 16-35 details security policies for the CI_RESET command.

Table 16-35. Access/Security policies for CI_RESET command

Security Enabled	Security Disabled
Available	Available

16.5.9 CI_PROTECT and CI_UNPROTECT

These complementary functions are used for toggling the state of the FOPT[PROTB] control bit. Flash programing/erase checks the status of the FOPT[PROTB] bit before undertaking any changes to the flash array. If FOPT[PROTB] = 0, then the device is considered in a "protected" state and (except for BDM-initiated mass-erase) the flash memory will not be modified.

Any calls to CI_READ_WRITE or CI_ERASE to modify the flash memory should be preceded by a call to CI_UNPROTECT, and followed by a call to CI_PROTECT. In other words and in order, CI_UNPROTECT --> CI_READ_WRITE (or CI_ERASE) --> CI_PROTECT.

16.5.9.1 CI_PROTECT command packet format

The 5-bit command code for the CI_PROTECT command is 0x07. The extension bits are 0.

Table 16-36. CI_PROTECT command packet format at mailbox level

Mailbox #	Description	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Command byte	0	0	1	1	1	0	0	0
1	Parameter byte	0	0	0	0	0	0	0	0
2-31	NOT USED	NOT USED							

16.5.9.2 CI_UNPROTECT command packet format

The 5-bit command code for the CI_UNPROTECT command is 0x08. The extension bits are 0.

Table 16-37. CI_PROTECT command packet format at mailbox level

Mailbox #	Description	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Command byte	0	1	0	0	0	0	0	0
1	Parameter byte	0	0	0	0	0	0	0	0
2-31	NOT USED	NOT USED							

16.5.9.3 CI_PROTECT and CI_UNPROTECT response packets format

The first byte of the response packet contains the command packet previously sent.

The second byte is a general status byte. When the command response is complete, COCO is set to 1.

16.5.9.4 CI_PROTECT and CI_UNPROTECT access/security policies

Table 16-38 details security policies for the CI _PROTECT and CI_UNPROTECT commands.

Table 16-38. Access/Security policies for CI Return Device Info command

Security Enabled	Security Disabled
Available	Available

16.6 User-Callable ROM functions

To access available ROM functions for flash program/erase and basic device functions, function calls are trapped and then handled using a function table.

The primary function of the FXLC95000 ROM is to provide storage for flash programming/erase firmware and perform some basic management of device functions. A small number of ROM functions are accessible from user code. The call hierarchy is shown in Figure 16-6.

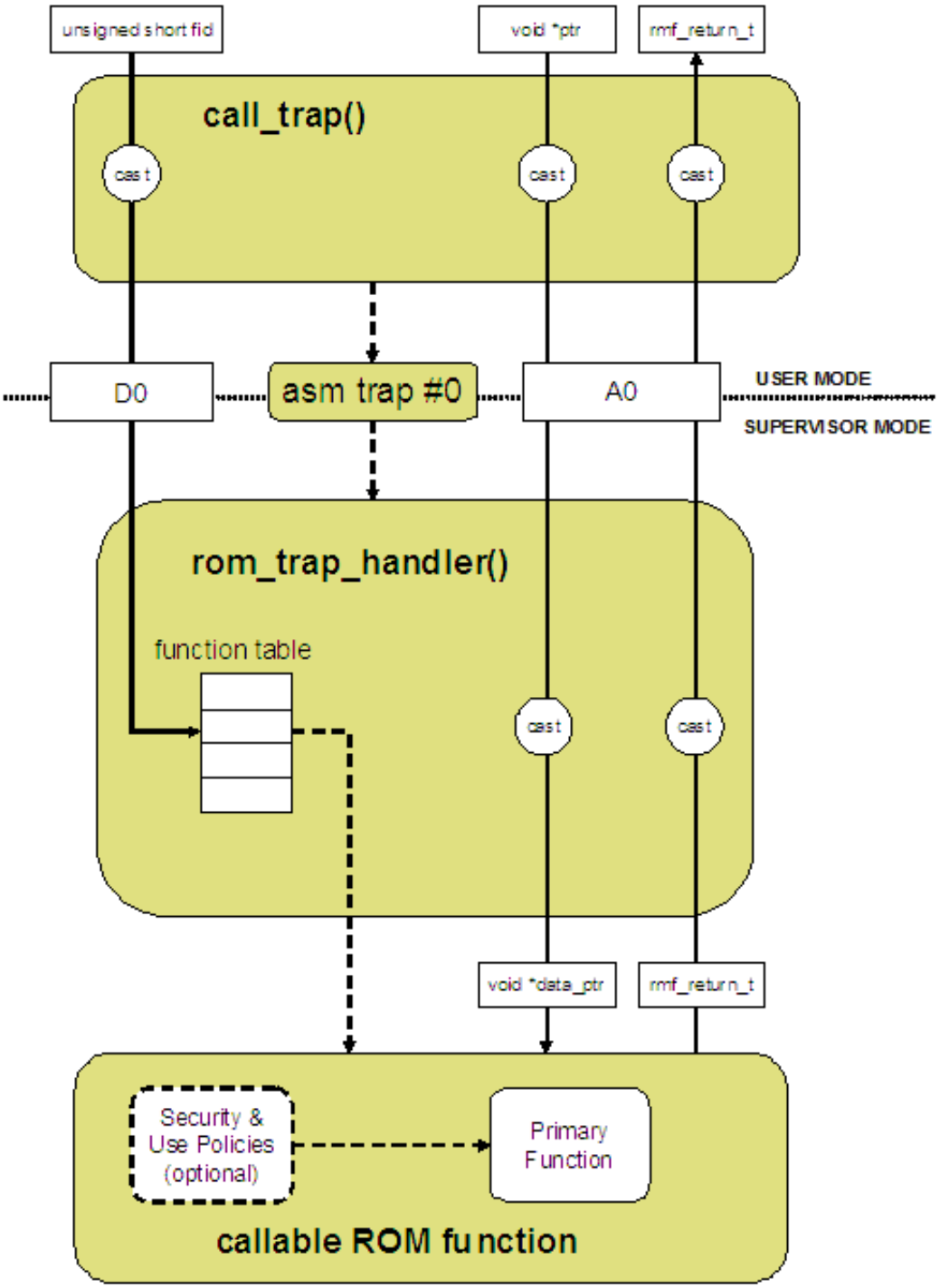


Figure 16-6. Call hierarchy for ROM functions

All user-callable ROM functions are invoked via the call_trap() function. The C prototype for this function is:

```
typedef union {
void * ptr;
```

```
unsigned long val;
} rmf_return_t;
rmf_return_t call_trap(unsigned short fid, void *ptr);
```

Input parameters are a function ID code (fid) and a void pointer (which is internally recast to a structure type specific to the function being called).

There are 32 bits of data returned. Depending on the function, these may be a pointer to a structure or simply an unsigned long. When using the rmf_return_t C language data type, you will need to specify varName.ptr or varName.val, depending on the data type into which you want to cast the result.

It is possible to call ROM functions directly in assembler. An inspection of the call_trap() source code in the example (below) shows how this is done. Simply load register D0 with the function ID and load register A0 with a structure pointer. Then run the assembler "trap #0" instruction, to transfer control to the supervisor routine associated with that instruction. The result will be returned in register A0.

Example: call_trap()

```
rmf_return_t call_trap(unsigned short fid, void *ptr)
{
rmf_return_t result;
asm {
move.w  fid,d0      // D0 contains function ID   (16 bits)
move.l  ptr,a0      // A0 contains pointer to structure  (32 bits)
trap #0
move.l  a0,result   // store in local 'result' variable
}
return result;
}
```

Only predefined, Freescale functions can be called via call_trap(). These functions are defined in Table 16-39.

Table 16-39. ROM functions callable via call_trap()

Function ID	Description	Input/Output Structures
RMF_DEV_INFO	Retrieve 32-bit device identifier	In: NULL Out: rmf_device_info_sts_t
RMF_FLASH_PROGRAM	Program flash	In: rmf_flash_prog_params_t Out: rmf_flash_op_sts_t
RMF_FLASH_ERASE	Erase flash	In: rmf_flash_erase_params_t Out: rmf_flash_op_sts_t
RMF_CRC	Calculate a checksum over a range of memory.	In: rmf_crc_params_t Out: rmf_crc_sts_t
RMF_CI	Transfer Control to ROM-based command interpreter.	In: NULL No return
RMF_FLASH_PROTECT	Protect flash from accidental program/erase.	In: NULL Out: NULL

Table continues on the next page...

Table 16-39. ROM functions callable via call_trap() (continued)

Function ID	Description	Input/Output Structures
(see RMF_FLASH_PROTECT and RMF_FLASH_UNPROTECT)		
RMF_FLASH_UNPROTECT	Enable program/erase. To enable program/erase operations, first unprotect flash.	In: NULL Out: NULL
RMF_FLASH_UNSECURE	Temporarily unsecure the device.	In: NULL Out: NULL

Function details are provided in the following sub-sections.

RMF functions use the same error codes found earlier in the description of the ROM command interpreter, but the function of the COCO bit is different. For RMF calls:

- COCO = 0 indicates that an error has occurred.
- COCO = 1 indicates that the function has completed properly.

Table 16-40. Common RMF error codes

Error Name	Error = Bits 6:4	Description
RMF_ERROR_PARAM	001	An input parameter did not pass muster. Examples include: • The erase password does not match RMF_ENABLE_FLASH_ERASE.
RMF_ERROR_PROT	010	Returned when an attempt is made to program or erase flash while flash protection is active (FOPT[PROTB]=0). Before attempting to program/erase the flash, call RMF_FLASH_UNPROTECT (to unprotect the flash).
RMF_ERROR_SECURITY	011	Most CI commands are unavailable when security has been set (FOPT[SSC]=10). The RMF_ERROR_SECURITY error code will be returned when an attempt has been made to execute a prohibited function.
RMF_ERROR_VERIFY	100	Returned as a result of a PROGRAM or ERASE command, if the final results of the operation do not match expected values. (ERASE values are all Fs. PROGRAM values are the input values.) • The address offset of the first found error will be returned in *first_err (and not in mailboxes). • RMF_ERROR_VERIFY error only occurs when the verify bit is set in the command byte.
RMF_ERROR_RIGHTS	101	The user does not have access rights to some feature. For example, writing to the ROM.
RMF_ERROR_RANGE	110	Generally applicable to cases where an input parameter is not within an expected range of values. An example would be a write command that attempted to program flash memory across physical rows of the device.
RMF_ERROR_COMMAND	111	RMF_ERROR_COMMAND code is returned when: • The command interpreter does not recognize a command code. • An incomplete packet is recognized.

NOTE

To conserve RAM, all RMF functions return data in a common "consolidated return structure". This structure can be thought of similar to a C union of the various return structures shown later. Structure values will be lost each time that an RMF call is made. Any structures that are required for later use should first be saved.

NOTE

Note that interrupts are disabled by the trap handler, and STOP mode operation is temporarily disabled. System clocks will remain in their high speed states (16 MHz) during program operations. The call_trap() function is NOT re-entrant.

16.6.1 RMF_DEV_INFO

16.6.1.1 RMF_DEV_INFO description

RMF_DEV_INFO returns the 32-bit device ID (DID), and hardware and software version numbers. The device ID is a part of the "rights management" system described in Security.

Typically, a master controller will request this information of the FXLC95000CL as the first step in initiating an upgrade.

16.6.1.2 RMF_DEV_INFO input structure syntax

No parameters are necessary. However, when invoking RMF_DEV_INFO, you must supply a NULL structure pointer to the call_trap() function.

16.6.1.3 RMF_DEV_INFO output structure syntax

```
typedef struct {
  unsigned long id;                    // Device ID
  unsigned char rom_major;            // ROM version identifier is rom_major.rom_minor
  unsigned char rom_minor;
  unsigned char ft_flash_major;     // Freescale flash version identifier is
ft_flash_major.ft_flash_minor
  unsigned char ft_flash_minor;
  unsigned char hw_major;             // ASIC version identifier is hw_major.hw_minor
  unsigned char hw_minor;
  unsigned short RESERVED;
  unsigned char part_number_0;     // Part Number MSB
```

```
  unsigned char part_number_1;    // Part Number LSB
  unsigned char rscr;                // Reset Control and Status Register
  unsigned char security_state;    // Security state of device
} rmf_device_info_sts_t;
```

16.6.1.4 RMF_DEV_INFO error codes

RMF_DEV_INFO always succeeds. There are no error codes.

16.6.1.5 RMF_DEV_INFO operation

RMF_DEV_INFO returns useful information about the device.

Table 16-41. Return parameters for RMF_DEV_INFO

Variable	Function
id	32-bit relatively unique identifier for this unit
rom_major	Major version number for ROM software
rom_minor	Minor version number for ROM software
ft_flash_major	Major version number for Freescale flash content
ft_flash_minor	Minor version number for Freescale flash content
hw_major	Major version number for this device type
hw_minor	Minor version number for this device type
part_number[1:0]	Part number in BCD format
rcsr	Reset Control and Status Register
security_state	Security state of this device

NOTE

To conserve RAM, all RMF functions return data in a common "consolidated return structure." The structure pointer returned by the call_trap() function can be cast to type rmf_flash_prog_params_t (shown above). Structure values will be lost.

16.6.1.6 RMF_DEV_INFO access/security policies

RMF_DEV_INFO may be called at any time.

16.6.1.7 RMF_DEV_INFO example

```
rmf_device_info_t device_data;
device_data =  (call_trap(RMF_DEV_INFO, NULL)).val;
```

16.6.2 RMF_FLASH_PROGRAM

16.6.2.1 RMF_FLASH_PROGRAM description

All user access to the flash-controller programming functions is via the RMF_FLASH_PROGRAM function. Interrupts are disabled when RMF_FLASH_PROGRAM executes and STOP-mode operation is temporarily disabled. System clocks will remain in their high-speed states (16 MHz) during program operations.

Because flash operations interfere with STOP-mode operations, their use is not consistent with normal sensor operation (as described in Modes of operation). Frame operations will need to be suspended during the use of the RMF_FLASH_PROGRAM function.

NOTE

The flash-controller hardware is not accessible outside of Supervisor mode. Accessing flash-controller hardware by a Supervisor-mode method other than RMF_FLASH_PROGRAM (and companion functions, listed herein) is strongly discouraged.

Primary input attributes are an array of values to be programmed and the address at which the first value should be programmed in flash. Addresses are automatically incremented for each value in the input array.

16.6.2.2 RMF_FLASH_PROGRAM input structure syntax

```
typedef struct {
unsigned long pw;
unsigned long addr;
unsigned short num_lwords;
unsigned short reserved; // Write as 0x0000;
unsigned short reserved2; // Write as 0x0000;
unsigned short verify;
unsigned long *data;
} rmf_flash_prog_params_t;
```

NOTE

To conserve RAM, all RMF functions return data in a common “consolidated return structure”. The structure pointer returned

by the call_trap() function can be cast to type rmf_flash_prog_params_t (shown above). Structure values will be lost.

16.6.2.3 RMF_FLASH_PROGRAM input parameters

Table 16-42. rmf_flash_prog_params_t parameters

Parameter	Description
pw	This is a constant password required to enable program operation. It is used only to limit the possibility of runaway code accidentally enabling this function. If any value other than RMF_ENABLE_FLASH_PROGRAM is used as the value for this parameter, then the RMF_FLASH_PROGRAM function will return with a failed status code. **NOTE:** Any program operation must be 256 bytes or less, and fit within a single row of the flash array. The value of num_lwords should not exceed 64 words, and the range of words should not cross row boundaries.
addr	First address to be programmed. Because we are programming 4 bytes at a time, addr[1:0] must be 0.
num_lwords	This is the number of 32-bit words to be programmed. Any program operation must be 256 bytes or less and fit within a single row of the flash array. The value of num_lwords should not exceed 64, and the range of words should not cross row boundaries. It is OK to program as few as one 32-bit long word within a row. Only that one long word would be programmed.
reserved	Write as 0x0000
reserved2	Write as 0x0000
verify	TRUE = Once program operation is complete, run a verification of programmed values. FALSE = Do not run verify check.
data	This is a pointer to an array of values to be programmed, starting at addr.

16.6.2.4 RMF_FLASH_PROGRAM output structure syntax

```
typedef struct {
unsigned short coco;        // Command complete (TRUE/FALSE)
unsigned short err;         // Error code, if any
unsigned long  *first_err;  // Address of first error found in any verify operations
} rmf_flash_op_sts_t;
```

NOTE

To conserve RAM, all RMF functions return data in a common “consolidated return structure”. The structure pointer returned by the call_trap() function can be cast to type rmf_flash_op_sts_t (shown above). Structure values will be lost each time that an RMF call is made. Any structure values that are required for later use should first be saved.

16.6.2.5 RMF_FLASH_PROGRAM output parameters

Table 16-43. rmf_flash_prog_params_t parameters

Parameter	Description
COCO	COmmand COmplete TRUE if command completed without errors. FALSE if command did not complete and/or errors were found. Check "err" field for details.
err	Error Code See Table 16-40 for possible values.
first_error	Address of first error found in any verify operations.

16.6.2.6 RMF_FLASH_PROGRAM access/security policies

Flash program operations are allowed only on those pages in which the associated PRR bit is 1.

16.6.2.7 RMF_FLASH_PROGRAM example

This example attempts to write four, 32-bit long-words to flash memory starting at address 0x(00)00_2000. After programming those words, the function will perform a verify operation and leave the flash in protected state

```
static rmf_flash_prog_params_t pparams;
rmf_flash_op_sts_t *psts;
static unsigned long words[4] = {0x01234567, 0x89ABCDEF, 0x55555555, 0xCCCCCCCC};
pparams.pw = RMF_ENABLE_FLASH_PROGRAM;
pparams.addr = 0x00002000;
pparams.num_lwords = 4;
pparams.reserved = 0;
pparams.reserved2 = 0;
pparams.verify = TRUE;
pparams.data = words;
psts = (call_trap(RMF_FLASH_PROGRAM, &pparams)).ptr;
if (psts->coco == TRUE) {
// Proceed you like.
} else {
// Process errors from failed program operation
}
```

16.6.3 RMF_FLASH_ERASE

16.6.3.1 RMF_FLASH_ERASE description

All user access to the flash-controller erase functions (both page-erase and mass-erase) is via the RMF_FLASH_ERASE function. When RMF_FLASH_ERASE executes and STOP-mode operation is temporarily disabled, interrupts are disabled. System clocks will remain in their high speed states (16 MHz) during erase operations.

Because flash operations interfere with STOP-mode operation, their use is not consistent with normal sensor operation (as described in Modes of operation). Frame operation will need to be suspended during the use of the RMF_FLASH_ERASE function.

NOTE

The flash-controller hardware is not accessible outside of Supervisor mode. Accessing flash-controller hardware by a Supervisor-mode method other than the RMF_FLASH_ERASE function (and companion functions, listed herein) is strongly discouraged.

This main flash array is composed of 128 pages of 1K bytes each. RMF_FLASH_ERASE requires (as parameters) the starting long word address and ending long word addresses of the range to be erased. Erasure is subject to page protections afforded by the PRR.

Before calling RMF_FLASH_ERASE, flash protection must be disabled (by calling RMF_FLASH_UNPROTECT). After the erase operation finishes, re-assert flash protection (by calling RMF_FLASH_UNPROTECT).

16.6.3.2 RMF_FLASH_ERASE input structure syntax

```
typedef struct {
unsigned long pw;
unsigned long start; // starting address
unsigned long end; // ending address
unsigned short reserved; // write as 0
unsigned short verify;
} rmf_flash_erase_params_t;
```

16.6.3.3 RMF_FLASH_ERASE input parameters

Table 16-44. rmf_flash_erase_params_t parameters

Parameter	Description
pw	Password This is a constant password required to enable program operation. It is used only to limit the possibility of runaway code accidentally enabling the RMF_ENABLE_FLASH function. If any other value than

Table continues on the next page...

Table 16-44. rmf_flash_erase_params_t parameters (continued)

Parameter	Description
	RMF_ENABLE_FLASH_ERASE is used as the value for this parameter, then RMF_ENABLE_FLASH will return with a failed status code.
start	Starting address Lowest address to be erased. Must be on a page start boundary (evenly divisible by 0x400)
end	Ending address Highest address to be erased. Must be on a page end boundary (ending in 0x3FF, 0x7FF, 0xBFF, 0xFFF)
verify	TRUE = After program operation finishes, verify that the erased area is "all 1s". FALSE = Do not run verify check.
reserved	Write as 0x0000.

16.6.3.4 RMF_FLASH_ERASE output structure syntax

```
typedef struct {
unsigned short coco;        // Command complete (TRUE/FALSE)
unsigned short err;         // Error code, if any
unsigned long  *first_err;  // address of first error found in any verify operations
} rmf_flash_op_sts_t;
```

NOTE

To conserve RAM, all RMF functions return data in a common "consolidated return structure". The structure pointer returned by the call_trap() function can be cast to type rmf_flash_op_sts_t (shown above). Structure values will be lost each time that an RMF call is made. Any structure values required for later use should first be saved.

16.6.3.5 RMF_FLASH_ERASE output parameters

Table 16-45. rmf_flash_erase_params_t Parameters

Parameter	Description
COCO	COmmand COmplete TRUE if command completed without errors. FALSE if command did not complete and/or errors were found. Check "err" field for details.
err	Error Code 0x0000 if no errors encountered. See Table 16-40 for additional values. **NOTE:** ROM code will return RMF_ERROR_RANGE in the err field, if out-of-range addresses are supplied when attempting an erase via CI or RMF.
first_error	Address of first error found in any verify operations

16.6.3.6 RMF_FLASH_ERASE access/security policies

Table 16-46 details the security policies for RMF_FLASH_ERASE.

Table 16-46. Access/Security policies for RMF_FLASH_ERASE

Main Flash Array	
Security Enabled	Security Disabled
Subject to PRR	

Subject to PRR, erase operations on flash memory are restricted to those in which the PRR[page number] bit is 1. See Page-Release Register (PRR).

16.6.3.7 RMF_FLASH_ERASE example

This example attempts to perform a mass erase of the main flash array.

```
static rmf_flash_erase_params_t eparams;
rmf_flash_op_sts_t *ests;
eparams.pw = RMF_ENABLE_FLASH_ERASE;
eparams.start = 0x0;   // mass erase
eparams.end = 0x1FFFF;
eparams.verify = TRUE;

eparams.reserved = 0;

ests = (call_trap(RMF_FLASH_ERASE, &eparams)).ptr;

if (ests->coco == TRUE) {
// Program flash memory if you like.
} else {
// Process errors from failed erase operation
}
```

16.6.4 RMF_FLASH_PROTECT and RMF_FLASH_UNPROTECT

16.6.4.1 RMF_FLASH_PROTECT and RMF_FLASH_UNPROTECT description

These complementary functions allow temporary (until the next reset sequence) changes in the protection status of flash memory. Their sole function is to manipulate the protection functions in the flash options register (FOPT[PW, PROTB]) to their proper states.

Separating changes in protection from programming/erase operations improves the odds against accidental programming/erasure of flash.

The NVOPT byte in the main flash array must be reprogrammed to make any permanent change in the protection state of the part. See Flash Options Register (FLASH_FOPT) Flash Options Register (FOPT).

16.6.4.2 RMF_FLASH_PROTECT and RMF_FLASH_UNPROTECT input structure syntax

No parameters are necessary for either function. When invoking RMF_FLASH_PROTECT and RMF_FLASH_UNPROTECT functions, supply a NULL structure pointer to the call_trap() function.

16.6.4.3 RMF_FLASH_PROTECT and RMF_FLASH_UNPROTECT output structure syntax

RMF_FLASH_PROTECT and RMF_FLASH_UNPROTECT functions always succeed. They both return a NULL pointer.

16.6.4.4 RMF_FLASH_PROTECT and RMF_FLASH_UNPROTECT access/security policies

RMF_FLASH_PROTECT and RMF_FLASH_UNPROTECT functions may be called at any time.

16.6.4.5 RMF_FLASH_PROTECT and RMF_FLASH_UNPROTECT example

```
call_trap(RMF_FLASH_UNPROTECT, NULL); // unprotect flash
call_trap(RMF_FLASH_PROTECT, NULL);   // protect flash
```

16.6.5 RMF_FLASH_UNSECURE

16.6.5.1 RMF_FLASH_UNSECURE description

RMF_FLASH_UNSECURE allows temporary (until the next reset sequence) changes in the security status of the device. Its sole function is to manipulate the security functions in the flash options register (FOPT[SSW, SSC]) to their proper states.

The NVOPT byte in the main flash array must be reprogrammed to make any permanent change in the security state of the part. See Flash Security.

16.6.5.2 RMF_FLASH_UNSECURE input structure syntax

No parameters are necessary. When invoking RMF_FLASH_UNSECURE, supply a NULL structure pointer to the call_trap() function.

16.6.5.3 RMF_FLASH_UNSECURE output structure syntax

RMF_FLASH_UNSECURE always succeeds. It returns a NULL pointer.

16.6.5.4 RMF_FLASH_UNSECURE access/security policies

RMF_FLASH_UNSECURE may be called at any time.

To clear security, external parties (slave port and BDM port) normally mass erase the device.

16.6.5.5 RMF_FLASH_UNSECURE example

```
call_trap(RMF_FLASH_UNSECURE, NULL); // clear security until next reset operation
```

16.6.6 RMF_CRC

16.6.6.1 RMF_CRC description

The RMF_CRC function uses a Cyclic Redundancy Check (CRC) function to generate a CRC value over a specified range of memory. The 16-bit CRC-CCITT polynomial ($x^{16} + x^{12} + x^5 + 1$) is used to generate the CRC code.

RMF_CRC features:

- CRC16-CCITT compliancy with $x^{16} + x^{12} + x^5 + 1$ polynomial
- Error detection for all single, double, odd, and most multi-bit errors
- Programmable, initial-seed value (for sensors use, we recommend using 0x1D0F)

16.6.6.2 RMF_CRC input structure syntax

```
typedef struct {
unsigned long seed;
unsigned long starting_addr;
unsigned long ending_addr;
unsigned short reserved;  // write as 0
} rmf_crc_params_t;
```

16.6.6.3 RMF_CRC input parameters

Table 16-47. rmf_crc_params_t parameters

Parameter	Description
seed	The "seed" for the CRC algorithm
starting_addr	Address of the first location in the memory map to be checked
ending_addr	Address of the last location in the memory map to be checked
Reserved	Write as 0x0.

16.6.6.4 RMF_CRC output structure syntax

```
typedef struct {
unsigned short coco;
unsigned short sts;
unsigned long crc;
} rmf_crc_sts_t;
```

The CRC function returns an error (RMF_ERROR_RANGE), if the number of bytes to perform the CRC on is less than 4 (end addr - start addr + 1).

NOTE

To conserve RAM, all RMF functions return data in a common “consolidated return structure”. The structure pointer returned by the call_trap() function can be cast to type rmf_crc_sts_t (shown above). Structure values will be lost each time that an RMF call is made. Any structure values required for later use should first be saved.

16.6.6.5 RMF_CRC error codes

RMF_CRC always returns a value. If the number of bytes in the requested CRC range is less than 4, then the RMF_CRC function returns RMF_ERROR_RANGE in the sts field.

16.6.6.6 RMF_CRC example

```
rmf_crc_params_t crc_params;
rmf_crc_sts_t *crc_results;
unsigned long crc;
crc_params.seed = 0x1D0F;
crc_params.reserved = 0;
crc_params.starting_addr = 0x00000000;
crc_params.ending_addr   = 0x0001FFFF;
crc_results = (call_trap(RMF_CRC, &crc_params)).ptr;
if (crc_results->sts == RMF_ERROR_NONE) {
crc = crc_results->crc;
}
```

16.6.6.7 RMF_CRC access/security policies

RMF_CRC may be called at any time.

Chapter 17
ColdFire v1 Interrupt Controller (CF1_INTC)

17.1 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.

The following table 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 17-1. Exception Processing Comparison

Attribute	HCS08	V1 ColdFire
Exception Vector Table	32 two-byte entries, fixed location at upper end of memory	115 four-byte entries, located at lower end of memory at reset, relocatable with the VBR

Table continues on the next page...

Table 17-1. Exception Processing Comparison (continued)

Attribute	HCS08	V1 ColdFire
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

17.1.1 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 ... > 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 × vector number). After the exception vector has been fetched, the contents of the vector serves 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 × 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:

```
ColdFire Vector Number = 62 + 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 following simplified block diagram.

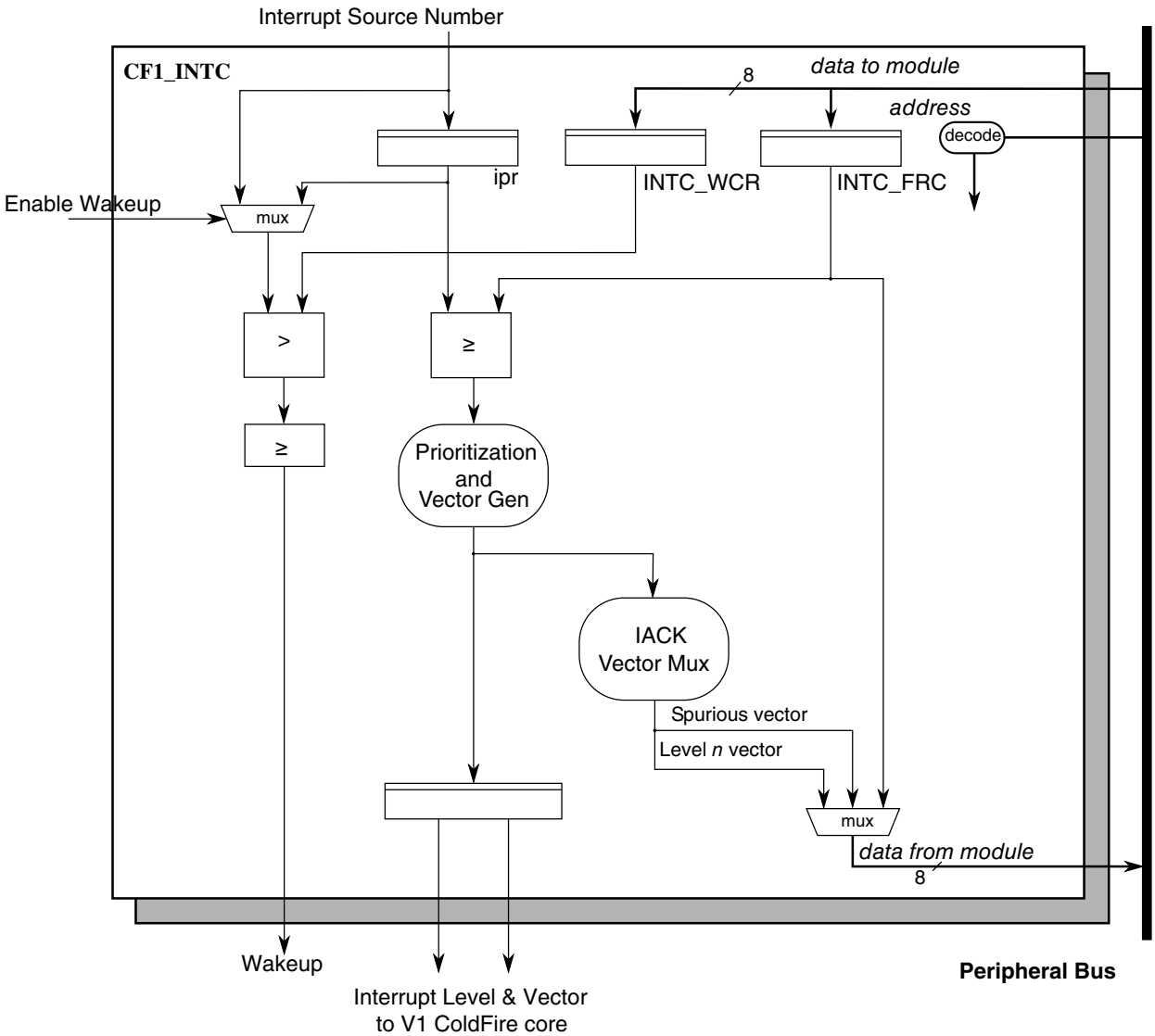


Figure 17-1. CF1_INTC Block Diagram

17.1.2 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 30 peripheral I/O interrupt requests plus seven software (one per level) interrupt requests

- Fixed association between interrupt request source and level plus priority
  - 30 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

17.1.3 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.

17.2 External Signal Description

The CF1_INTC module does not include any external interfaces.

17.3 Interrupt Request Level and Priority Assignments

The CF1_INTC module implements a sparsely populated 7 × 9 matrix of levels (7) and priorities within each level (9).

For details, see the chip-specific information about interrupt vector assignments.

Note

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

17.4 Memory Map and Registers

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.

In the programming model, 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 memory map.

INTC memory map

Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
FFFF_FFD0	Force Interrupt Register (INTC_FRC)	8	R/W	00h	17.4.1/304
FFFF_FFD8	INTC Programmable Level 6 Priority Registers (INTC_PL6P7)	8	R/W	00h	17.4.2/305
FFFF_FFD9	INTC Programmable Level 6 Priority Registers (INTC_PL6P6)	8	R/W	00h	17.4.2/305
FFFF_FFDB	INTC Wakeup Control Register (INTC_WCR)	8	R/W	80h	17.4.3/306
FFFF_FFDE	INTC Set Interrupt Force Register (INTC_SFRC)	8	W	00h	17.4.4/307
FFFF_FFDF	INTC Clear Interrupt Force Register (INTC_CFRC)	8	W	00h	17.4.5/308
FFFF_FFE0	INTC Software IACK Register (INTC_SWIACK)	8	R	00h	17.4.6/309
FFFF_FFE4	INTC Level-n IACK Registers (INTC_LVL1IACK)	8	R	18h	17.4.7/309
FFFF_FFE8	INTC Level-n IACK Registers (INTC_LVL2IACK)	8	R	18h	17.4.7/309
FFFF_FFEC	INTC Level-n IACK Registers (INTC_LVL3IACK)	8	R	18h	17.4.7/309
FFFF_FFF0	INTC Level-n IACK Registers (INTC_LVL4IACK)	8	R	18h	17.4.7/309
FFFF_FFF4	INTC Level-n IACK Registers (INTC_LVL5IACK)	8	R	18h	17.4.7/309
FFFF_FFF8	INTC Level-n IACK Registers (INTC_LVL6IACK)	8	R	18h	17.4.7/309
FFFF_FFFC	INTC Level-n IACK Registers (INTC_LVL7IACK)	8	R	18h	17.4.7/309

17.4.1 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

For compatibility with other ColdFire interrupt controllers, this register's bit numbers are actually 63-56 (not 7-0 as shown in the register diagram).

Address: FFFF_FFC0h base + 10h offset = FFFF_FFD0h

Bit	7	6	5	4	3	2	1	0
Read	0	LVL1	LVL2	LVL3	LVL4	LVL5	LVL6	LVL7
Write								
Reset	0	0	0	0	0	0	0	0

INTC_FRC field descriptions

Field	Description
7 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
6 LVL1	Force level 1 interrupt 0 Negates the forced level 1 interrupt request. 1 Forces a level 1 interrupt request.
5 LVL2	Force level 2 interrupt 0 Negates the forced level 2 interrupt request. 1 Forces a level 2 interrupt request.
4 LVL3	Force level 3 interrupt 0 Negates the forced level 3 interrupt request. 1 Forces a level 3 interrupt request.
3 LVL4	Force level 4 interrupt 0 Negates the forced level 4 interrupt request. 1 Forces a level 4 interrupt request.

Table continues on the next page...

INTC_FRC field descriptions (continued)

Field	Description
2 LVL5	Force level 5 interrupt 0 Negates the forced level 5 interrupt request. 1 Forces a level 5 interrupt request.
1 LVL6	Force level 6 interrupt 0 Negates the forced level 6 interrupt request. 1 Forces a level 6 interrupt request.
0 LVL7	Force level 7 interrupt 0 Negates the forced level 7 interrupt request. 1 Forces a level 7 interrupt request.

17.4.2 INTC Programmable Level 6 Priority Registers (INTC_PL6P*n*)

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 Using INTC_PL6P{7,6} Registers.

Address: FFFF_FFC0h base + 18h offset + (1d × i), where i=0d to 1d

Bit	7	6	5	4	3	2	1	0
Read	0		REQN					
Write								
Reset	0	0	0	0	0	0	0	0

INTC_PL6P*n* field descriptions

Field	Description
7–6 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
REQN	Request number Defines the peripheral IRQ number to be remapped as the level 6, priority 7 (for INTC_PL6P7) request and level 6, priority 6 (for INTC_PL6P6) request. **NOTE:** The value must be a valid interrupt number. Unused or reserved interrupt numbers are ignored. The selected vector number is derived from the decimal value of the REQN field, which is *n* in the following mapping formula: for vectors 64-102, *n* = *Vector_Number - 64*, while for vectors 110-114, *n* = *Vector_Number - 71*. In other words, the REQN field's minimum value is 4d (selecting vector 68) and maximum value is 43d (selecting vector 114).

17.4.3 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 wakeup signal is defined as:

```
wakeup = INTC_WCR[ENB] + (level of any asserted_int_request > INTC_WCR[MASK])
```

Address: FFFF_FFC0h base + 1Bh offset = FFFF_FFDBh

Bit	7	6	5	4	3	2	1	0
Read	ENB	0					MASK	
Write								
Reset	1	0	0	0	0	0	0	0

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	This field is reserved. This read-only field is reserved and always has the value 0.
MASK	Interrupt mask level

17.4.4 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.

Address: FFFF_FFC0h base + 1Eh offset = FFFF_FFDEh

Bit	7	6	5	4	3	2	1	0
Read	Reserved							
Write			SET					
Reset	0	0	0	0	0	0	0	0

INTC_SFRC field descriptions

Field	Description
7–6 Reserved	This field is reserved.
SET	For data values within the 56–62 range, the corresponding bit in the INTC_FRC register is set, as defined below. **NOTE:** 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. 111000 Bit 56, INTC_FRC[LVL7] is set 111001 Bit 57, INTC_FRC[LVL6] is set

Table continues on the next page...

INTC_SFRC field descriptions (continued)

Field	Description
	111010 Bit 58, INTC_FRC[LVL5] is set 111011 Bit 59, INTC_FRC[LVL4] is set 111100 Bit 60, INTC_FRC[LVL3] is set 111101 Bit 61, INTC_FRC[LVL2] is set 111110 Bit 62, INTC_FRC[LVL1] is set

17.4.5 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.

Address: FFFF_FFC0h base + 1Fh offset = FFFF_FFDFh

Bit	7	6	5	4	3	2	1	0
Read								
Write	Reserved		CLR					
Reset	0	0	0	0	0	0	0	0

INTC_CFRC field descriptions

Field	Description
7–6 Reserved	This field is reserved.
CLR	For data values within the 56–62 range, the corresponding bit in the INTC_FRC register is cleared, as defined below. **NOTE:** 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. 111000 Bit 56, INTC_FRC[LVL7] is cleared 111001 Bit 57, INTC_FRC[LVL6] is cleared 111010 Bit 58, INTC_FRC[LVL5] is cleared 111011 Bit 59, INTC_FRC[LVL4] is cleared 111100 Bit 60, INTC_FRC[LVL3] is cleared 111101 Bit 61, INTC_FRC[LVL2] is cleared 111110 Bit 62, INTC_FRC[LVL1] is cleared

17.4.6 INTC Software IACK Register (INTC_SWIACK)

Refer to the description of the Level-*n* IACK registers.

Address: FFFF_FFC0h base + 20h offset = FFFF_FFE0h

Bit	7	6	5	4	3	2	1	0
Read	0	VECN						
Write								
Reset	0	0	0	0	0	0	0	0

INTC_SWIACK field descriptions

Field	Description
7 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
VECN	Vector number Indicates the appropriate vector number. 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.

17.4.7 INTC Level-*n* IACK Registers (INTC_LVL*n*IACK)

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.

Address: FFFF_FFC0h base + 24h offset + (4d × i), where i=0d to 6d

Bit	7	6	5	4	3	2	1	0
Read	0	VECN						
Write								
Reset	0	0	0	1	1	0	0	0

INTC_LVL*n*IACK field descriptions

Field	Description
7 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
VECN	Vector number Indicates the appropriate vector number. It is the highest priority request within the specified level-n. If there are no pending requests within the level, VECN is 0x18 (24) to signal a spurious interrupt.

17.5 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 7 interrupt requests and the module's interfaces.

17.5.1 Handling of Non-Maskable Level 7 Interrupt Requests

In this context of this discussion, the non-maskable level 7 interrupt requests refer only to the masking capability provided by the processor's SR[I] field. 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.

The CPU treats level 7 interrupts as non-maskable, edge-sensitive requests, while levels 1 through 6 are maskable, level-sensitive requests. As a result of this definition, level 7 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 7 (regardless of the SR[I] field) and each time the SR[I] mask changes from 7 to a lower value while the encoded request level remains at 7.

17.6 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.

17.7 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.

17.7.1 Emulation of the HCS08's 1-Level IRQ Handling

As noted in Table 10-7, 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.

17.7.2 Using INTC_PL6P{7,6} Registers

The INTC Programmable Level 6, Priority {7,6} registers (INTC_PL6P{7,6}) 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 13 (0Dh) = vector 77 = level 24, priority 6
- sci1_tx = interrupt source 14 (0Eh) = vector 78 = level 24, 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 13 (0Dh), remaps sci1_rx as level 6, priority 7.
- Setting INTC_PL6P6 to 14 (0Eh), remaps sci1_tx as level 6, priority 6.

The reset state of the INTC_PL6P{7,6} registers disables any request remapping.

17.7.3 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 17-18.

```
                    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.l*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 17-18. 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 18
Programmable Delay Block (PDB)

18.1 Introduction

The PDB can be used to generate two independent digital pulse waveform outputs on pins PDB_A and PDB_B, with programmable width and delay, and synchronized to internal event triggers.

The programmable delay block provides a controllable delay between an internal trigger input signal and two digital output signals generated on pins PDB_A and PDB_B. Two different output waveforms are available:

- Fixed duration (short) pulse with programmable delay
- Programmable duration pulse with programmable delay

The output signal, once triggered, can be either a single shot or repetitive (with programmable periodicity).

In addition to generating the 2 output signals, the PDB can be used to schedule events *within a frame* via its output interrupts, which is a primary purpose of the PDB module.

18.1.1 Features

PDB features include:

- 16-bit resolution counter with prescaler. The counter is clocked by the peripheral bus clock.
- Positive transition of a trigger_input will initiate the counter. The trigger source is software-programmable to be one of the following:
  - Φ_A started
  - Φ_D started
  - Software trigger
- Supports two output signals. Each output signal has an independently controlled delay from the trigger_input.

- Digital comparator outputs can be used to schedule precise edge placement for a pulsed output of desired delay and width.
- Continuous-trigger or single-shot mode supported.
- Each output is independently enabled/configured.

18.1.2 Modes of operation

The Programmable Delay Block supports three modes of operation.

The three modes of operation include:

- Disabled: Counter is off and both A and B outputs are low.
- Enabled One Shot: Upon receiving a positive edge on the input trigger, the Counter is enabled and restarted at count zero. A and B will see only one output transition per input trigger.
- Enabled Continuous: Upon receiving a positive edge on the input trigger, the Counter is enabled and restarted at count zero. When the count reaches the value specified in the PDB_MOD register, the counter will be rolled over to zero again, and the counting will be restarted. This enables a continuous stream of output pulses from a single-trigger input.

18.1.3 Block diagram

Figure 18-1 illustrates the basic structure of the PDB block. It contains a single counter whose output is compared against three different digital values. The delayA and delayB determine the time between assertion of the trigger input to the time that changes in the output signals are initiated. These times are defined as:

- Trigger input to A output time = (prescaler x delayA) + 1 peripheral bus clock cycle.
- Trigger input to B output time = (prescaler x delayB) + 1 peripheral bus clock cycle.
- When using both A and B comparators to schedule both edges on an output pulse, add one additional peripheral bus-clock cycle.

The third digital value (modulus) is used to reset the counter back to zero at the end of the count. If PDB_CSR[CONT] is set, then the counter will then resume a new count. Otherwise, the timer operation will cease until the next trigger input event occurs.

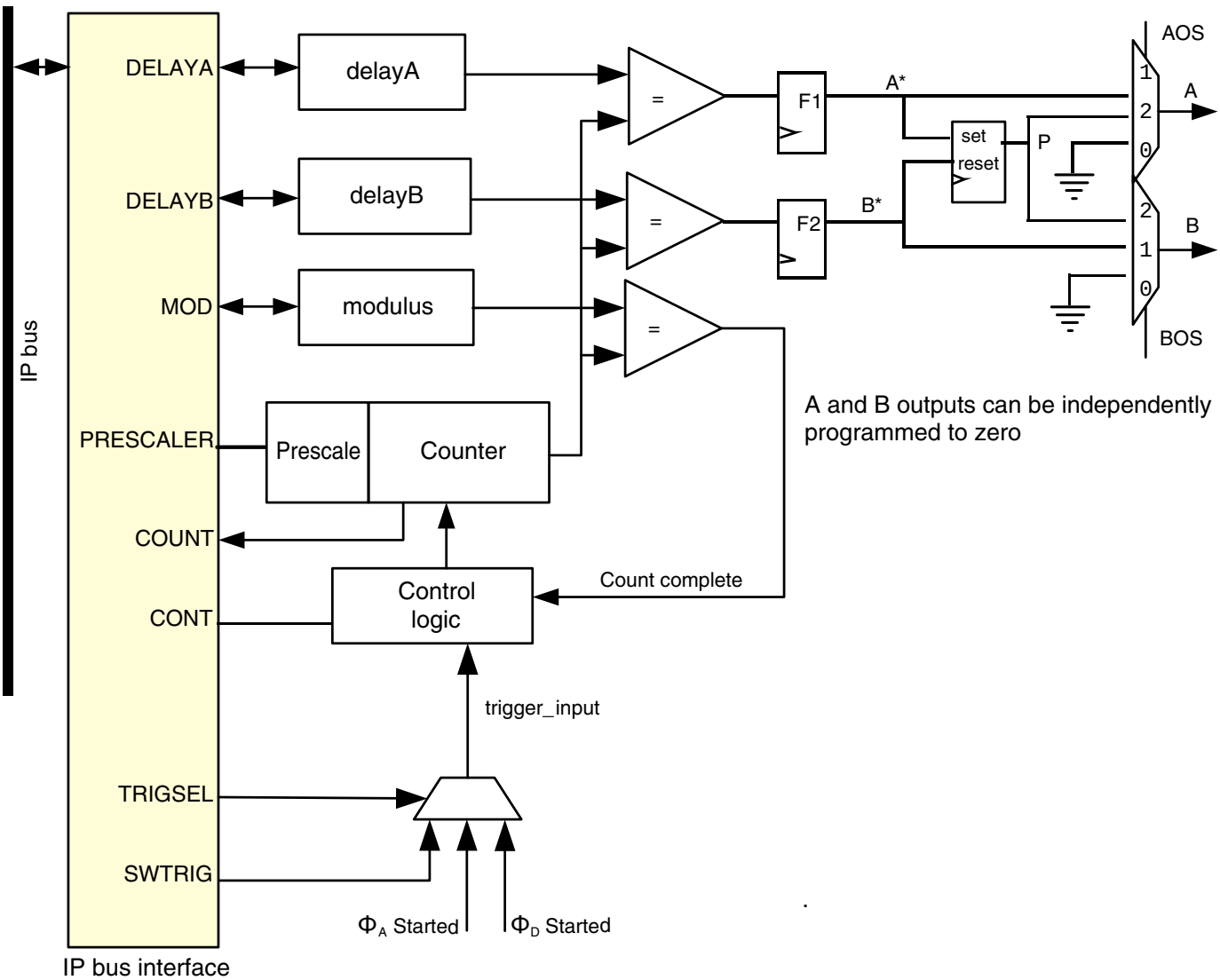


Figure 18-1. FXLC95000L PDB block diagram

The programmable duration pulse mode is shown in Figure 18-2. In this case, A* and B* are used to precisely schedule the rising and falling edges of the output waveform.

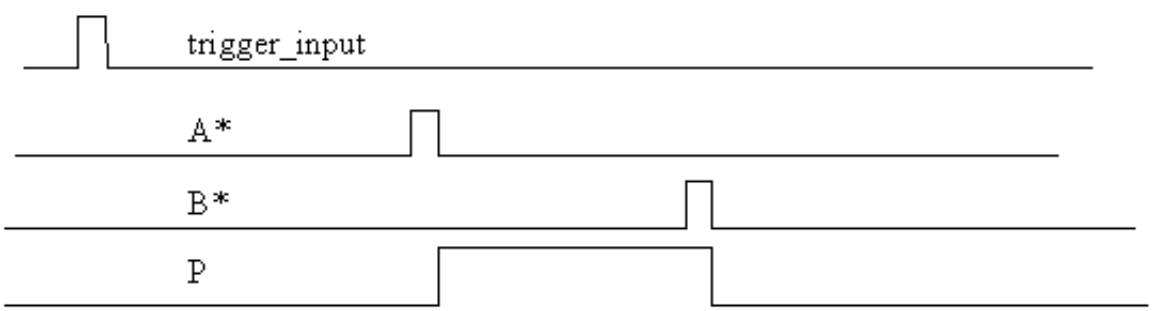


Figure 18-2. Trigger-pulsed output operation

18.1.4 PDB memory map/register definition

PDB memory map

Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
FFFF_EC00	PDB Control and Status Register (PDB_CSR)	16	R/W	0000h	18.1.5/320
FFFF_EC02	PDB Delay A Register (PDB_DELAYA)	16	R/W	0000h	18.1.6/322
FFFF_EC04	PDB Delay B Register (PDB_DELAYB)	16	R/W	0000h	18.1.7/322
FFFF_EC06	PDB Counter Modulus Register (PDB_MOD)	16	R/W	FFFFh	18.1.8/323
FFFF_EC08	PDB Counter Value (PDB_COUNT)	16	R	FFFFh	18.1.9/323

18.1.5 PDB Control and Status Register (PDB_CSR)

This register contains status and control bits for the Programmable Delay Block. The counter is enabled if EN has been set to 1. In general, you should reconfigure the module only when the module is not enabled (EN = 0).

Address: FFFF_EC00h base + 0h offset = FFFF_EC00h

Bit	15	14	13	12	11	10	9	8
Read	PRESCALER			SB	SA	IENB	IENA	BOS
Write								
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
Read	BOS	AOS		CONT	0	TRIGSEL		EN
Write					SWTRIG			
Reset	0	0	0	0	0	0	0	0

PDB_CSR field descriptions

Field	Description
15–13 PRESCALER	Clock Prescaler Select This value should be changed only when the module is not enabled. The pulse width of outputs A and B is also impacted by this field. Larger prescalers result in longer pulse widths as would be expected. 000 Timer uses peripheral clock. 001 Timer uses peripheral clock/2. 010 Timer uses peripheral clock/4. 011 Timer uses peripheral clock/8. 100 Timer uses peripheral clock/16.

Table continues on the next page...

PDB_CSR field descriptions (continued)

Field	Description
	101 Timer uses peripheral clock/32. 110 Timer uses peripheral clock/64. 111 Timer uses peripheral clock/128.
12 SB	Sticky B This bit is sticky. It will remain at 1 once set, even after B goes low. Clear this bit by writing a 1 to this location. SB is the source for the interrupt enabled by IENB. 0 B* has not triggered. 1 B* has triggered.
11 SA	Sticky A This bit is sticky. It will remain at 1 once set, even after A goes low. Clear this bit by writing a 1 to this location. SA is the source for the interrupt enabled by IENA. 0 A* has not triggered. 1 A* has triggered.
10 IENB	Interrupt Enable B The interrupt is cleared by writing a 1 to SB. 0 Interrupt B is not enabled. 1 Assert an interrupt when B* triggers and SB goes high.
9 IENA	Interrupt Enable A The interrupt is cleared by writing a 1 to SA. 0 Interrupt A is not enabled 1 Assert an interrupt when A* triggers and SA goes high.
8–7 BOS	B Output Select 00 B output is zero. 01 B=B* 10 PulseOut (P) 11 RESERVED
6–5 AOS	A Output Select 00 A output is zero. 01 A=A* 10 PulseOut (P) 11 RESERVED
4 CONT	Continuous Mode Enable 0 Module is in OneShot mode 1 Module is in continuous mode
3 SWTRIG	Software Trigger - When TRIGSEL=2'b00 and the module is enabled, writing a 1 to this field will trigger a reset and restart of the counter. This bit always reads as 0.
2–1 TRIGSEL	Input Trigger Select The Φ_D started option is independent of the interrupt enable for Φ_D in the SIM Frame Control and Status Register. It can be used even when that interrupt is not enabled. The timing is the same either way.

Table continues on the next page...

PDB_CSR field descriptions (continued)

Field	Description
	00 Software trigger 01 Φ_A started 10 Φ_D started 11 RESERVED
0 EN	Module Enable 0 Module is not enabled. Outputs are 0. 1 Module is enabled for use.

18.1.6 PDB Delay A Register (PDB_DELAYA)

These registers are used to specify the delay from assertion of Trigger Input to assertion of A and B outputs. The delay is in terms of prescaled peripheral clock cycles. These registers should only be changed when the module is not enabled.

Address: FFFF_EC00h base + 2h offset = FFFF_EC02h

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read Write	DELAYA															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

PDB_DELAYA field descriptions

Field	Description
DELAYA	Delay is in terms of prescaled peripheral clock cycles from assertion of Trigger Input to assertion of A output. A delay value of $0000 will never be reached and no output trigger will occur since the counter goes from $0001 to $FFFF.

18.1.7 PDB Delay B Register (PDB_DELAYB)

Address: FFFF_EC00h base + 4h offset = FFFF_EC04h

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read Write	DELAYB															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

PDB_DELAYB field descriptions

Field	Description
DELAYB	Delay is in terms of prescaled peripheral clock cycles from assertion of Trigger Input to assertion of B output.

PDB_DELAYB field descriptions (continued)

Field	Description
	A delay value of $0000 will never be reached and no output trigger will occur since the counter goes from $0001 to $FFFF.

18.1.8 PDB Counter Modulus Register (PDB_MOD)

This register specifies the period of the counter in terms of prescaled peripheral bus cycles. When the counter reaches this value, it will be reset back to all 0s. If PDB_CSR[CONT] is set to one, the count will begin anew.

This register should be changed only when the module is not enabled.

Address: FFFF_EC00h base + 6h offset = FFFF_EC06h

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read Write	MOD															
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

PDB_MOD field descriptions

Field	Description
MOD	Specifies the period of the counter in terms of prescaled peripheral bus cycles

18.1.9 PDB Counter Value (PDB_COUNT)

This register can be used to read the current value of the counter. It is READ ONLY.

Address: FFFF_EC00h base + 8h offset = FFFF_EC08h

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	COUNT															
Write																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

PDB_COUNT field descriptions

Field	Description
COUNT	This register read content reflects the current value of the counter. When counting starts, the register switches to 0x0001. When a rollover occurs, it will roll over to 0x0001.

18.1.10 Considerations

- A and B outputs are defined to be glitch-free.
- Additional trigger events -after the first, but before the timer times out- will cause the counter to restart.
- Using the prescaler impacts the timing resolution.

  Use of prescalers > 1 limit the count/delay accuracy in terms of peripheral clocks (to the modulus of the prescaler value). If the prescaler is set to div 2 then the only values of total peripheral clocks that can be detected are even values, if div is set to 4 then the only values of total peripheral clocks that can be decoded as detected are mod(4) and so forth. If a user requires a long delay value and used div 128, then the resolution is limited to 128 bus clocks.

  Therefore, use the lowest possible prescaler for a given application.

- In general, change or reconfigure all R/W registers only when the module is not enabled.

18.2 Resets

This module has a single reset input, corresponding to the chip-wide peripheral reset.

18.3 Clocks

This module has a single clock input, the IP Bus peripheral clock.

18.4 Interrupts

This module has two possible interrupts:

- One interrupt that is associated with the A output
- One interrupt that is associated with the B output

Chapter 19
Modulo Timer 16-Bit (MTIM16)

19.1 Chip-specific information about MTIM16

The MTIM is a simple, 16-bit timer with several software-selectable clock sources and a programmable interrupt. This module is incorporated on numerous Freescale ICs. On FXLC95000CL, it is connected as follows:

- BUSCLK = IP bus clock
- XCLK = FFCLK from the CLKGEN module. This is nominally 1/8 X $F_{osc\text{-}low}$.
- TCLK = GROUND

Clock options are limited on the FXLC95000CL. All clocks are derived from the same oscillator source, and for power and noise reasons, it is imperative that clocking during STOP modes be extremely localized. Use of XCLK is discouraged for applications that require utmost sensor accuracy. In these cases, FFCLK should be disabled (by setting OSCTRL[FFSEN] to 0).

Restricting the MTIM to use BUSCLK implies, that for applications that strictly follow the frame structure suggested in Frame-based sampling, MTIM activity is restricted to Φ_D.

19.2 Introduction

The MTIM (or MTIM16) module is a simple 16-bit timer with several software selectable clock sources and a programmable interrupt.

19.3 Features

Timer system features include:

- 16-bit up-counter

- Free-running or 16-bit modulo limit
- Software controllable interrupt on overflow
- Counter reset bit (TRST)
- Counter stop bit (TSTP)

- Four software selectable clock sources for input to prescaler:
  - System bus clock — rising edge
  - Fixed frequency clock (XCLK) — rising edge
  - External clock source on the TCLK pin — rising edge
  - External clock source on the TCLK pin — falling edge
- Nine selectable clock prescale values:
  - Clock source divide by 1, 2, 4, 8, 16, 32, 64, 128, or 256

19.3.1 Block Diagram

The following figure is a block diagram of the modulo timer module.

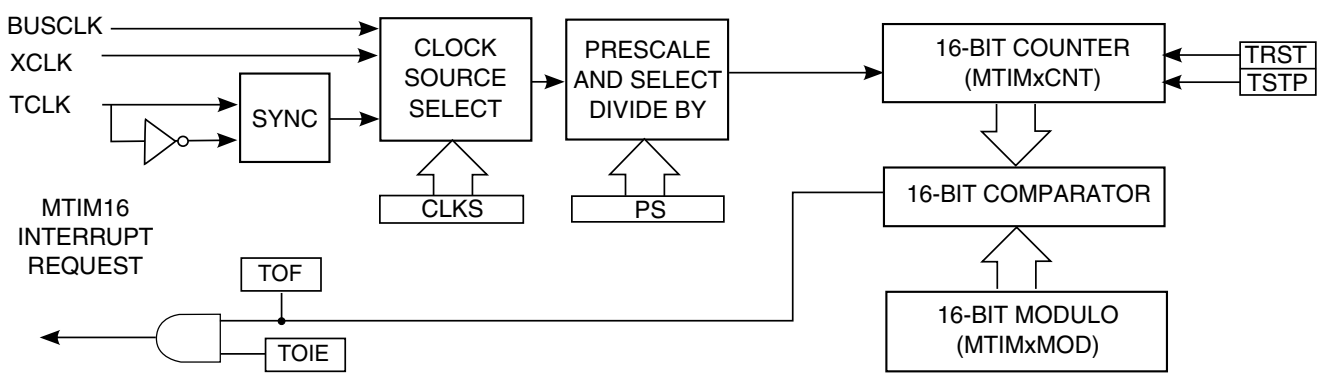


Figure 19-1. Modulo Timer (MTIM16) Block Diagram

19.3.2 Modes of Operation

This section defines MTIM16 operation in stop, wait, and background debug modes.

19.3.2.1 MTIM16 in Wait Mode

The MTIM16 continues to run in wait mode if enabled prior to the execution of the WAIT instruction. The timer overflow interrupt brings the MCU out of wait mode if it is enabled. For lowest possible current consumption, the MTIM16 should be stopped by software if it is not needed as an interrupt source during wait mode.

19.3.2.2 MTIM16 in Stop Modes

MTIM operation in stop modes is chip-specific. See details about MCU power modes and clocking.

- If the MTIM is unclocked in any MCU stop mode, it is disabled in that mode, regardless of the module settings before the STOP instruction was executed. It cannot be used as a wakeup source in that mode.
- If the MTIM is clocked and enabled in an MCU stop mode, it can be used as a wakeup source in that mode.

Upon waking from very low-power stop modes, the MTIM enters its reset state.

For low-power stop modes:

- If the device exits any of these modes with a reset, the MTIM module enters its reset state.
- If the device exits any of these modes with an interrupt, the MTIM module continues from its state in the low-power stop mode.
- If the counter was active upon entering any of these modes, the count resumes from the current value.

19.3.2.3 MTIM16 in Active Background Mode

The MTIM16 stops all counting until the microcontroller returns to normal user operating mode. Counting resumes from the suspended value as long as an MTIM16 reset did not occur (TRST written to a 1).

19.4 External Signal Description

This module does not have any external signals.

19.5 Memory Map and Register Descriptions

Each MTIM16 module includes six registers.

If a chip has more than one MTIM16 module, register names include placeholder characters.

MTIM memory map

Address offset (hex)	Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
0	FFFF_80A0	MTIM16 status and control register (MTIM_SC)	8	R/W	10h	19.5.1/328
1	FFFF_80A1	MTIM16 clock configuration register (MTIM_CLK)	8	R/W	00h	19.5.2/329
2	FFFF_80A2	MTIM16 counter register high (MTIM_CNTH)	8	R	00h	19.5.3/330
3	FFFF_80A3	MTIM16 counter register low (MTIM_CNTL)	8	R	00h	19.5.4/331
4	FFFF_80A4	MTIM16 modulo register high (MTIM_MODH)	8	R/W	00h	19.5.5/332
5	FFFF_80A5	MTIM16 modulo register low (MTIM_MODL)	8	R/W	00h	19.5.6/333

19.5.1 MTIM16 status and control register (MTIM_SC)

This register contains the overflow status flag and control bits. Use them to configure the interrupt enable, reset the counter, and stop the counter.

Address: FFFF_80A0h base + 0h offset = FFFF_80A0h

Bit	7	6	5	4	3	2	1	0
Read	TOF	TOIE	0	TSTP	0			
Write	0		TRST					
Reset	0	0	0	1	0	0	0	0

MTIM_SC field descriptions

Field	Description
7 TOF	MTIM16 overflow flag This bit is set when the MTIM16 counter register overflows to 0x0000 after reaching the value in the MTIM16 modulo register. Clear TOF by reading the SC register while TOF is set and then by writing 0 to TOF. Writing 1 has no effect. TOF is also cleared when 1 is written to TRST.

Table continues on the next page...

MTIM_SC field descriptions (continued)

Field	Description
	0 MTIM16 counter has not reached the overflow value in the MTIM16 modulo register. 1 MTIM16 counter has reached the overflow value in the MTIM16 modulo register.
6 TOIE	MTIM16 overflow interrupt enable This read/write bit enables MTIM16 overflow interrupts. If TOIE is set, then an interrupt is generated when TOF = 1. Reset clears TOIE. Do not set TOIE if TOF = 1; instead, clear TOF first, and then set TOIE. 0 TOF interrupts are disabled. Use software polling. 1 TOF interrupts are enabled.
5 TRST	MTIM16 counter reset When 1 is written to this write-only bit, the MTIM16 counter register resets to 0x0000 and TOF is cleared. Writing 1 to this bit also causes the modulo value to take effect at once. Reading this bit always returns 0. 0 No effect. MTIM16 counter remains in its current state. 1 MTIM16 counter is reset to 0x0000.
4 TSTP	MTIM16 counter stop When set, this read/write bit stops the MTIM16 counter at its current value. Counting resumes from the current value when TSTP is cleared. Reset sets TSTP to prevent the MTIM16 from counting. 0 MTIM16 counter is active. 1 MTIM16 counter is stopped.
Reserved	This field is reserved. This read-only field is reserved and always has the value 0.

19.5.2 MTIM16 clock configuration register (MTIM_CLK)

This register contains the clock select bits (CLKS) and the prescaler select bits (PS).

Address: FFFF_80A0h base + 1h offset = FFFF_80A1h

Bit	7	6	5	4	3	2	1	0
Read	0		CLKS		PS			
Write								
Reset	0	0	0	0	0	0	0	0

MTIM_CLK field descriptions

Field	Description
7–6 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
5–4 CLKS	Clock source select These two read/write bits select one of four different clock sources as the input to the MTIM16 prescaler. Changing the clock source while the counter is active does not clear the counter. The count continues with the new clock source. Reset clears CLKS to 00.

Table continues on the next page...

MTIM_CLK field descriptions (continued)

Field	Description
	00 Encoding 0. Bus clock (BUSCLK) 01 Encoding 1. Fixed-frequency clock (XCLK) 10 Reserved 11 Reserved
PS	Clock source prescaler These four read/write bits select one of nine outputs from the 8-bit prescaler. Changing the prescaler value while the counter is active does not clear the counter. The count continues with the new prescaler value. Reset clears PS to 0000. 0000 Encoding 0. MTIM16 clock source / 1 0001 Encoding 1. MTIM16 clock source / 2 0010 Encoding 2. MTIM16 clock source / 4 0011 Encoding 3. MTIM16 clock source / 8 0100 Encoding 4. MTIM16 clock source / 16 0101 Encoding 5. MTIM16 clock source / 32 0110 Encoding 6. MTIM16 clock source / 64 0111 Encoding 7. MTIM16 clock source / 128 1xxx Encoding 8+. MTIM16 clock source / 256

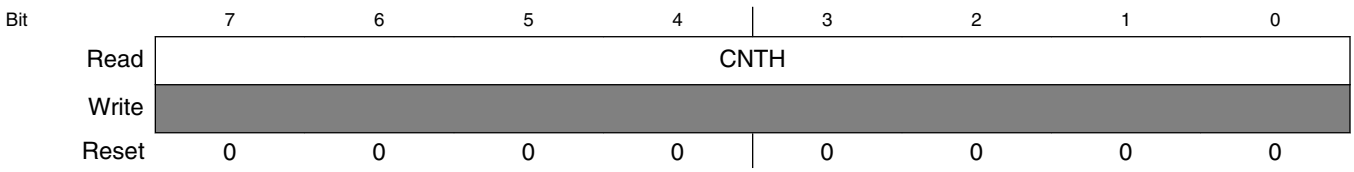
19.5.3 MTIM16 counter register high (MTIM_CNTH)

This register is the read-only value of the high byte of the current MTIM16 16-bit counter.

When either the CNTH or CNTL register is read, the content of the two registers is latched into a buffer where they remain latched until the other register is read. This allows the coherent 16-bit value to be read in both big-endian and little-endian compile environments and ensures the 16-bit counter is unaffected by the read operation. The coherency mechanism is automatically restarted by an MCU reset or by setting the TRST bit of the SC register (whether BDM mode 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 BDM became active, even if one or both halves of the counter 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, the appropriate value from the other half of the 16-bit value is read after returning to normal execution. The value read from the CNTH and CNTL registers in BDM mode is the value of these registers and not the value of their read buffer.

Address: FFFF_80A0h base + 2h offset = FFFF_80A2h



MTIM_CNTH field descriptions

Field	Description
CNTH	MTIM16 count (high byte) These 8 read-only bits contain the current high byte value of the 16-bit counter. Writing has no effect on this register. Reset clears the register to 0x00.

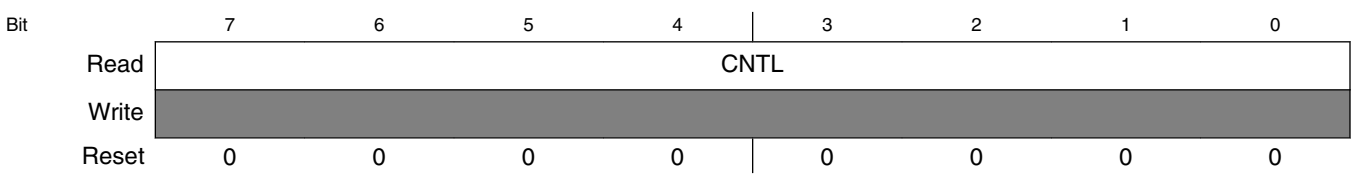
19.5.4 MTIM16 counter register low (MTIM_CNTL)

This register is the read-only value of the low byte of the current MTIM16 16-bit counter.

When either the CNTH or CNTL register is read, the content of the two registers is latched into a buffer where they remain latched until the other register is read. This allows the coherent 16-bit value to be read in both big-endian and little-endian compile environments and ensures the 16-bit counter is unaffected by the read operation. The coherency mechanism is automatically restarted by an MCU reset or by setting the TRST bit of the SC register (whether BDM mode 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 BDM became active, even if one or both halves of the counter 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, the appropriate value from the other half of the 16-bit value is read after returning to normal execution. The value read from the CNTH and CNTL registers in BDM mode is the value of these registers and not the value of their read buffer.

Address: FFFF_80A0h base + 3h offset = FFFF_80A3h



MTIM_CNTL field descriptions

Field	Description
CNTL	MTIM16 count (low byte)

MTIM_CNTL field descriptions (continued)

Field	Description
	These 8 read-only bits contain the current low byte value of the 16-bit counter. Writing has no effect on this register. Reset clears the register to 0x00.

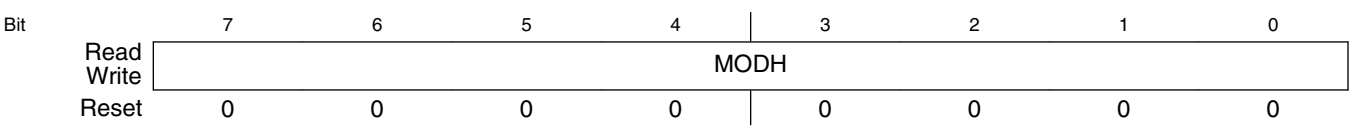
19.5.5 MTIM16 modulo register high (MTIM_MODH)

A value of 0x0000 in MODH:MODL puts the MTIM16 in free-running mode. Writing to either MODH or MODL latches the value into a buffer and the latched buffers are updated after the second byte writing. The updated values take effect and reload to the MODH:MODL registers in the next MTIM16 counter cycle, except for the first writing of modulo after a chip reset or in BDM mode. However, after a software reset, the MODH:MODL takes effect at once even if it did not take effect before the reset. On the first writing of MODH:MODL after chip reset, the counter is reset and the modulo takes effect immediately. The latching mechanism may be manually reset by setting the TRST bit of the SC register (whether BDM is active or not).

When BDM is active, the coherency mechanism is frozen so 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 writing to the modulo registers bypasses the buffer latches and writes directly to the modulo register while BDM is active, and also the counter is cleared at the same time.

Reading MODH:MODL returns the modulo values that are taking effect whenever in normal run mode or in BDM mode.

Address: FFFF_80A0h base + 4h offset = FFFF_80A4h



MTIM_MODH field descriptions

Field	Description
MODH	MTIM16 modulo (high byte) These 8 read/write bits contain the modulo high byte value used to reset the counter and set TOF. Reset sets the register to 0x00.

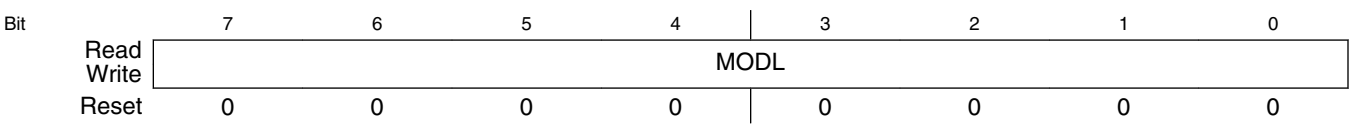
19.5.6 MTIM16 modulo register low (MTIM_MODL)

A value of 0x0000 in MODH:MODL puts the MTIM16 in free-running mode. Writing to either MODH or MODL latches the value into a buffer and the latched buffers are updated after the second byte writing. The updated values take effect and reload to the MODH:MODL registers in the next MITIM16 counter cycle, except for the first writing of modulo after a chip reset or in BDM mode. However, after a software reset, the MODH:MODL takes effect at once even if it did not take effect before the reset. On the first writing of MODH:MODL after chip reset, the counter is reset and the modulo takes effect immediately. The latching mechanism may be manually reset by setting the TRST bit of the SC register (whether BDM is active or not).

When BDM is active, the coherency mechanism is frozen so 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 writing to the modulo registers bypasses the buffer latches and writes directly to the modulo register while BDM is active, and also the counter is cleared at the same time.

Reading MODH:MODL returns the modulo values that are taking effect whenever in normal run mode or in BDM mode.

Address: FFFF_80A0h base + 5h offset = FFFF_80A5h



MTIM_MODL field descriptions

Field	Description
MODL	MTIM16 modulo (low byte) These 8 read/write bits contain the modulo low byte value used to reset the counter and set TOF. Reset sets the register to 0x00.

19.6 Functional Description

The MTIM16 is composed of a main 16-bit up-counter with 16-bit modulo register, a clock source selector, and a prescaler block with nine selectable values. The module also contains software selectable interrupt logic.

The MTIM16 counter (CNTH:CNTL registers) has three modes of operation: stopped, free-running, and modulo. The counter is stopped out of reset. If the counter starts without writing a new value to the modulo registers, it will be in free-running mode. The counter is in modulo mode when a value other than 0x0000 is in the modulo registers.

After an MCU reset, the counter stops and resets to 0x0000, and the modulo also resets to 0x0000. The bus clock functions as the default clock source, and the prescale value is divided by 1. To start the MTIM16 in free-running mode, write to the MTIM16 status and control (SC) register and clear the MTIM16 stop (TSTP) bit.

Clock sources are software selectable:

- the internal bus clock
- the fixed frequency clock (XCLK)

The MTIM16 clock select (CLKS) field in the CLK register selects the desired clock source. If the counter is active (the TSTP bit is 0) when a new clock source is selected, the counter continues counting from the previous value using the new clock source.

Nine prescale values are software selectable: clock source divided by 1, 2, 4, 8, 16, 32, 64, 128, or 256. The prescaler select bits (PS[3:0]) in the CLK register select the desired prescale value. If the counter is active (TSTP = 0) when a new prescaler value is selected, the counter continues counting from the previous value using the new prescaler value.

The MTIM16 modulo register (MODH:MODL) allows the overflow compare value to be set to any value from 0x0001 to 0xFFFF. Reset clears the modulo value to 0x0000, which results in a free-running counter.

When the counter is active (the TSTP bit is 0), it increases at the selected rate until the count matches the modulo value. When these values match, the counter overflows to 0x0000 and continues counting. The MTIM16 overflow flag (TOF) is set whenever the counter overflows. The flag sets on the transition from the modulo value to 0x0000.

Clearing TOF is a two-step process. The first step is to read the SC register while TOF is set. The second step is to write a 0 to TOF. If another overflow occurs between the first and second steps, the clearing process is reset and TOF stays set after the second step is performed. This will prevent the second occurrence from being missed. TOF is also cleared when a 1 is written to TRST.

The MTIM16 module allows for an optional interrupt to be generated whenever TOF is set. To enable the MTIM16 overflow interrupt, set the MTIM16 overflow interrupt enable (TOIE) bit in the SC register. The TOIE bit should never be written to be 1 while TOF is 1. Instead, TOF should be cleared first, and then the TOIE bit can be set to 1.

19.6.1 MTIM16 Operation Example

This section shows an example of the MTIM16 module's operation as the counter reaches a matching value from the modulo register.

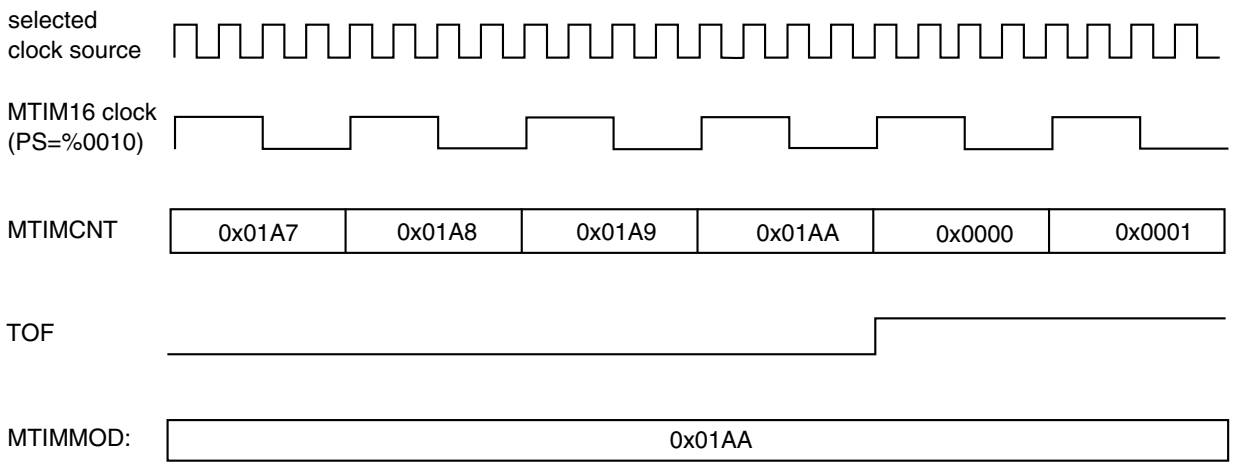


Figure 19-8. MTIM16 Counter Overflow Example

In this figure, the selected clock source could be any of the four possible choices. The prescaler is set to divide-by-4 (the PS field is 0010). The modulo value in the MODH:MODL register is set to 01AAh. When the counter (CNTH:CNTL registers) reaches the modulo value of 01AAh, the counter overflows to 0000h and continues counting. The timer overflow flag, TOF, sets when the counter value changes from 01AAh to 0000h. An MTIM16 overflow interrupt is generated when TOF is set, if the TOIE bit is 1.

Chapter 20
Timer/Pulse-Width Modulator (TPM)

20.1 Introduction

The TPM is a one-to-eight-channel timer system that supports traditional input capture, output compare, or edge-aligned PWM on each channel.

A control bit configures the TPM so all channels are used for center-aligned PWM functions. Timing functions are based on a 16-bit counter with prescaler and modulo features to control the time reference's frequency and range (period between overflows). 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.

The FXLC95000 includes a single, two-channel instance of the standard Freescale Timer/PWM module (TPM). This module can be used to create delay sequences (useful during flash programming) and measure external events (for proximity sensor functions). The module can also generate PWM output signals, although that operation can negatively impact sensor accuracy and power consumption.

The TPM has several software-selectable clock sources and three programmable interrupts. This module is incorporated on numerous Freescale ICs. On this device, it is connected as follows:

- BUSCLK = IP bus clock
- Fixed System Clock = FFCLK from the CLKGEN module. This is nominally 1/8 x $F_{osc\text{-}low}$.
- External Clock = Ground

Clock options are limited on the FXLC95000. All clocks are derived from the same oscillator source. For power and noise reasons, it is imperative that clocking during STOP modes be extremely localized. For applications that require the utmost sensor accuracy, using the fixed system clock is discouraged. In such cases, FFCLK should be disabled by setting OSCTRL[FFSEN] to 0.

Restricting the TPM to use BUSCLK implies that, for applications that strictly follow the frame structure suggested in Frame-based sampling, TPM activity is restricted to Φ_D.

20.1.1 Features

The TPM includes these features:

- One to eight Channels, with each channel having:
  - Input capture, output compare, or edge-aligned PWM
  - A rising-edge, falling-edge or any-edge input-capture trigger
  - Set, clear, or toggle output compare action
  - Selectable polarity on PWM outputs
  - Buffered, center-aligned Pulse-Width-Modulation (CPWM)
- Timer clock source selectable as bus clock, fixed frequency clock or an external clock
  - Prescale taps for divide-by 1, 2, 4, 8, 16, 32, 64 or 128 for any clock input selection
  - An additional, fixed-frequency clock input for selecting an on-chip clock source other than the bus clock
- 16-bit free-Running or Modulus count with Up/down selection
- One interrupt per channel and one interrupt for TPM counter overflow

20.1.2 Modes of operation

In general, TPM channels are independently configured to operate in input capture, output compare or edge-aligned PWM modes. A control bit allows the whole TPM (all channels) to switch to center-aligned, PWM mode. When center-aligned PWM mode is selected, input capture, output compare and edge-aligned PWM functions are not available on any channels of the TPM module.

When the MCU is in active BDM background or BDM foreground mode, the TPM temporarily suspends all counting until the MCU returns to normal user operating mode. During stop mode, all TPM input clocks are stopped, so the TPM is effectively disabled until clocks resume.

20.1.2.1 Input capture

When a selected edge event occurs on the associated MCU pin, the current value of the 16-bit, timer counter is captured into the channel value register and an interrupt flag bit is set. Rising edges, falling edges, any edge or no edge (disable channel) are selected as the active edge that triggers the input capture.

20.1.2.2 Output compare

When the value in the timer-counter register matches the channel value register, an interrupt flag bit is set and a selected output action is forced on the associated MCU pin. The output-compare action is selected to force the pin to 0, force the pin to 1, toggle the pin or ignore the pin (used for software-timing functions).

20.1.2.3 Edge-aligned PWM

The period of the PWM output signal is set as the value of the 16-bit, modulo register plus 1. The channel-value register sets the duty cycle of the PWM output signal. You can also choose the polarity of the PWM output signal.

Interrupts are available at the end of the period and at the duty-cycle transition point. This type of PWM signal is called edge-aligned because the leading edges of all PWM signals are aligned with the beginning of the period that is same for all channels within a TPM.

20.1.2.4 Center-aligned PWM

The period of the PWM output is set as twice the value of a 16-bit, modulo register. 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.

20.1.3 Block diagram

The TPM uses one input/output (I/O) pin per channel, TPMxCH*n* (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. (For the specific chip implementation, see .)

Figure 20-1 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.

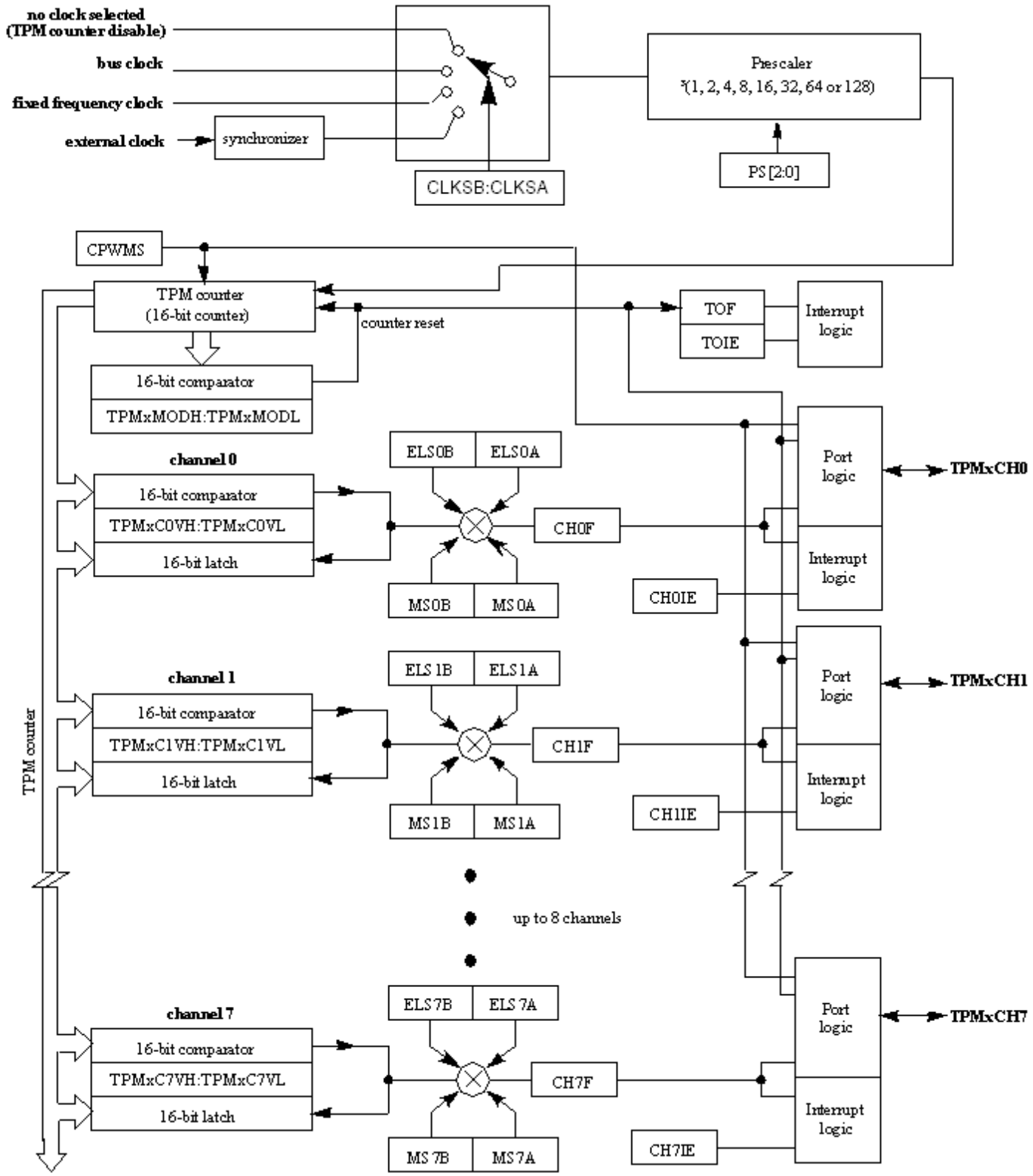


Figure 20-1. TPM Block Diagram

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.

20.2 Signal descriptions

There are 8 possible TPM channels that you can configure, and each channel can have a device I/O pin associated with it.

Table 20-1 shows the user-accessible signals for the TPM. The number of channels can 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. For the specific chip implementation, see .

Table 20-1. Signal Properties

Name	Function
TPMxCH*n*	I/O pin associated with TPM channel *n*.

20.2.1 TPMxCH*n* - TPM channel *n* I/O pins

The TPM channel does not control the I/O pin when ELS*n*B:ELS*n*A or CLKSB:CLKSA are cleared so it normally reverts to general-purpose I/O control. When CPWMS is set and ELS*n*B:ELS*n*A are not cleared, all TPM channels are configured for center-aligned PWM and the TPMxCH*n* pins are all controlled by TPM. When CPWMS is cleared, the MS*n*B:MS*n*A 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, MS*n*B:MS*n*A = 0:0, and ELS*n*B:ELS*n*A different from 0:0), the TPMxCH*n* pin is forced to act as an edge-sensitive input to the TPM. ELS*n*B:ELS*n*A control bits determine what polarity edge or edges trigger input capture events. The channel input signal is synchronized on the bus clock. This implies the minimum pulse width-that can be reliably detected-on an input capture pin is four bus clock periods. (With ideal clock pulses as near as two bus clocks can be detected.)

When a channel is configured for output compare (CPWMS = 0, MS*n*B:MS*n*A = 0:1 and ELS*n*B:ELS*n*A different from 0:0), the TPMxCH*n* pin is an output controlled by the TPM. The ELS*n*B:ELS*n*A bits determine whether the TPMxCH*n* pin is toggled, cleared or set each time the 16-bit channel value register matches the TPM counter.

When the output compare toggle mode is initially selected, the previous value on the pin is driven out until the next output compare event. The pin is then toggled.

When a channel is configured for edge-aligned PWM (CPWMS = 0, MS*n*B = 1 and ELS*n*B:ELS*n*A different from 0:0), the TPMxCH*n* pin is an output controlled by the TPM and the ELS*n*B:ELS*n*A bits control the polarity of the PWM output signal. When ELS*n*B is set and ELS*n*A is cleared, the TPMxCH*n* pin is forced high at the start of each new period (TPMxCNT=0x0000). The TPMxCH*n* pin is forced low when the channel value register matches the TPM counter. When ELS*n*A is set, the TPMxCH*n* pin is forced low at the start of each new period (TPMxCNT=0x0000). The TPMxCH*n* pin is forced high when the channel value register matches the TPM counter.

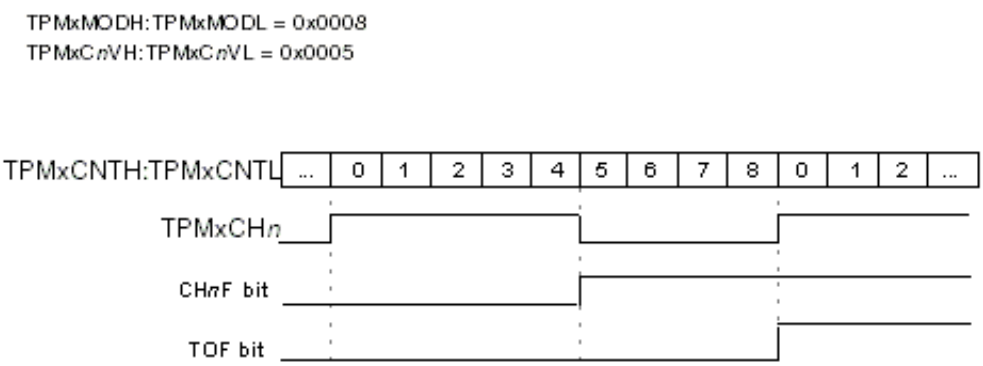


Figure 20-2. High-True pulse of an edge-aligned PWM

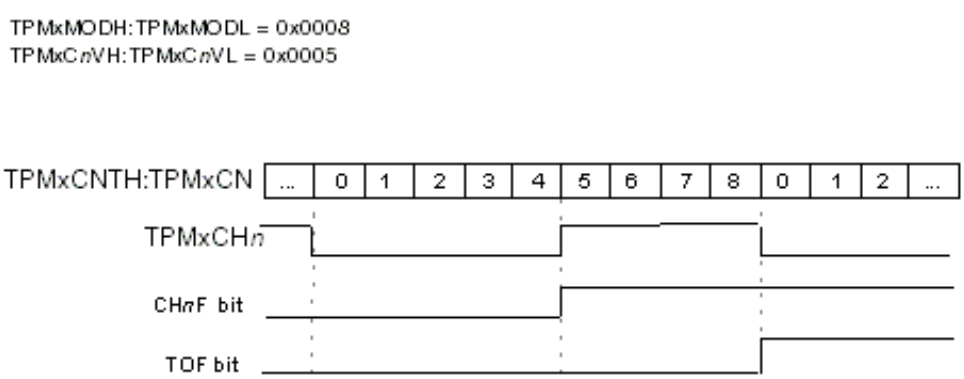


Figure 20-3. Low-True pulse of an edge-aligned PWM

When the TPM is configured for center-aligned PWM (CPWMS = 1 and ELS*n*B:ELS*n*A different from 0:0), the TPMxCH*n* pins are outputs controlled by the TPM and the ELS*n*B:ELS*n*A bits control the polarity of the PWM output signal. If ELS*n*B is set and ELS*n*A is cleared, the corresponding TPMxCH*n* pin is cleared when the TPM counter is counting up and the channel value register matches the TPM counter.

The corresponding TPMxCH*n* pin is set when the TPM counter is counting down and the channel value register matches the TPM counter. If ELS*n*A is set, the corresponding TPMxCH*n* pin is set when the TPM counter is counting up and the channel value register matches the TPM counter. The corresponding TPMxCH*n* pin is cleared when the TPM counter is counting down and the channel value register matches the TPM counter.

TPMxMODH:TPMxMODL = 0x0008
TPMxC*n*VH:TPMxC*n*VL = 0x0005

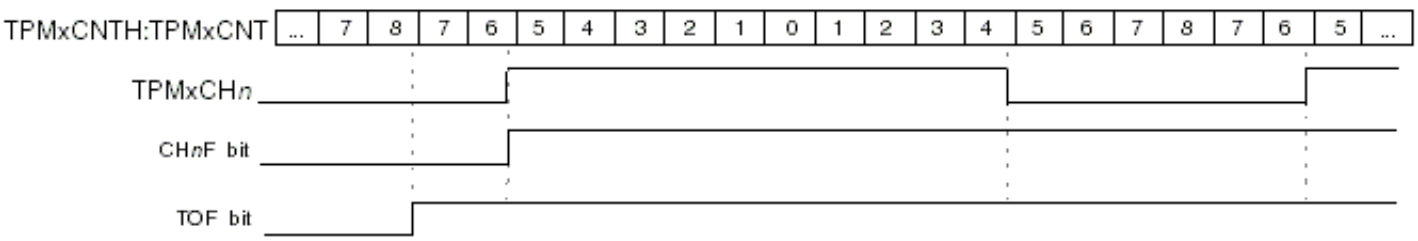


Figure 20-4. High-True pulse of a center-aligned PWM

TPMxMODH:TPMxMODL = 0x0008
TPMxC*n*VH:TPMxC*n*VL = 0x0005

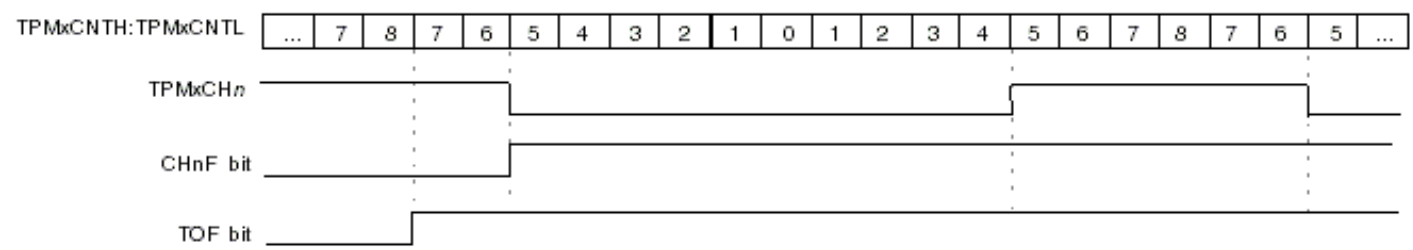


Figure 20-5. Low-True pulse of a center-aligned PWM

20.3 TPM memory map/register definition

TPM memory map

Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
FFFF_8120	TPM Status and Control Register (TPM_TPMxSC)	8	R/W	00h	20.3.1/345
FFFF_8121	TPM Counter Register High (TPM_TPMxCNTH)	8	R	00h	20.3.2/347
FFFF_8122	TPM Counter Register Low (TPM_TPMxCNTL)	8	R	00h	20.3.3/348
FFFF_8123	TPM Counter Modulo Register High (TPM_TPMxMODH)	8	R/W	00h	20.3.4/349
FFFF_8124	TPM Counter Modulo Register Low (TPM_TPMxMODHL)	8	R/W	00h	20.3.5/350
FFFF_8125	TPM Channel n Status and Control Register (TPM_TPMxCnSC)	8	R/W	00h	20.3.6/351
FFFF_8128	TPM Channel Value Register High (TPM_TPMxCnVH)	8	R/W	00h	20.3.7/352
FFFF_812B	TPM Channel Value Register Low (TPM_TPMxCnVL)	8	R/W	00h	20.3.8/354

20.3.1 TPM Status and Control Register (TPM_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.

Address: FFFF_8120h base + 0h offset = FFFF_8120h

Bit	7	6	5	4	3	2	1	0
Read Write	TOF	TOIE	CPWMS	CLKSB	CLKSA	PS2	PS1	PS0
Reset	0	0	0	0	0	0	0	0

TPM_TPMxSC field descriptions

Field	Description
7 TOF	Timer overflow flag. This read/write flag is set when the TPM counter resets to 0x0000 after reaching the modulo value programmed in the TPM counter modulo registers. Clear TOF by reading the TPM status and control register when TOF is set and then writing a logic 0 to TOF. If another TPM overflow occurs before the clearing sequence is completed, the sequence is reset so TOF remains set after the clear sequence was completed for the earlier TOF. This is done so that 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 1. Reset clears TOIE. 0 TOF interrupts inhibited (use for software polling). 1 TOF interrupts enabled.
5 CPWMS	Center-aligned PWM select. This read/write bit selects CPWM operating mode. By default, the TPM operates in up-counting mode for input capture, output compare and edge-aligned PWM functions. Setting

Table continues on the next page...

TPM_TPMxSC field descriptions (continued)

Field	Description
	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 MS*n*B:MS*n*A control bits in each channel's status and control register. 1 All channels operate in center-aligned PWM mode.
4 CLKSB	Clock source selection bits. This two-bit field is used to disable the TPM counter or select one of three clock sources to TPM counter and counter prescaler. 00 TPM Clock to Prescaler Input: No clock selected (TPM counter disable 01 TPM Clock to Prescaler Input: Bus clock 10 TPM Clock to Prescaler Input: Fixed frequency clock 11 TPM Clock to Prescaler Input: UNAVAILABLE
3 CLKSA	Clock source selection bits. This two-bit field is used to disable the TPM counter or select one of three clock sources to TPM counter and counter prescaler. 00 TPM Clock to Prescaler Input: No clock selected (TPM counter disable 01 TPM Clock to Prescaler Input: Bus clock 10 TPM Clock to Prescaler Input: Fixed frequency clock 11 TPM Clock to Prescaler Input: UNAVAILABLE
2 PS2	Prescale factor select. This three-bit field selects one of eight division factors for the TPM clock. This prescaler is located after any clock synchronization or clock selection, so it affects the clock selected to drive the TPM counter. The new prescale factor affects the selected clock on the next bus clock cycle after the new value is updated into the register bits. 000 TPM clock divided by 1 001 TPM clock divided by 2 010 TPM clock divided by 4 011 TPM clock divided by 8 100 TPM clock divided by 16 101 TPM clock divided by 32 110 TPM clock divided by 64 111 TPM clock divided by 128
1 PS1	Prescale factor select. This three-bit field selects one of eight division factors for the TPM clock. This prescaler is located after any clock synchronization or clock selection, so it affects the clock selected to drive the TPM counter. The new prescale factor affects the selected clock on the next bus clock cycle after the new value is updated into the register bits. 000 TPM clock divided by 1 001 TPM clock divided by 2 010 TPM clock divided by 4 011 TPM clock divided by 8 100 TPM clock divided by 16 101 TPM clock divided by 32 110 TPM clock divided by 64 111 TPM clock divided by 128
0 PS0	Prescale factor select. This three-bit field selects one of eight division factors for the TPM clock. This prescaler is located after any clock synchronization or clock selection, so it affects the clock selected to drive the TPM counter. The new prescale factor affects the selected clock on the next bus clock cycle after the new value is updated into the register bits.

Table continues on the next page...

TPM_TPMxSC field descriptions (continued)

Field	Description
	000 TPM clock divided by 1 001 TPM clock divided by 2 010 TPM clock divided by 4 011 TPM clock divided by 8 100 TPM clock divided by 16 101 TPM clock divided by 32 110 TPM clock divided by 64 111 TPM clock divided by 128

20.3.2 TPM Counter Register High (TPM_TPMxCNTH)

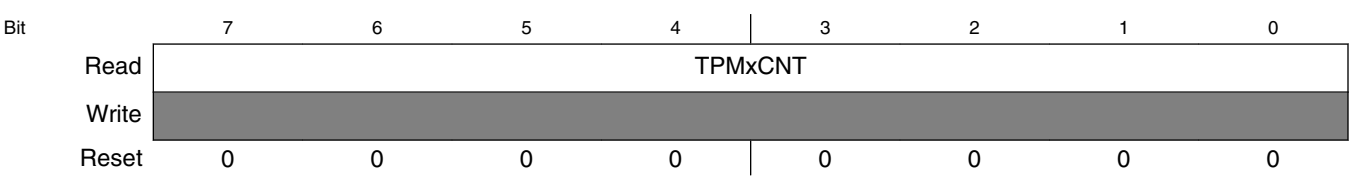
The two read-only TPM counter registers contain the high and low bytes of the value in the TPM counter. Reading either byte (TPMxCNTH or TPMxCNTL) latches the contents of both bytes into a buffer where they remain latched until the other half is read. This allows coherent, 16-bit reads in big-endian or little-endian order which makes reads 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.

When BDM is active, the timer counter is frozen. (This is the value you read.) The coherency mechanism is frozen so 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 you were in the middle of reading a 16-bit register when BDM became active, it reads the appropriate value from the other half of the 16-bit value after returning to normal execution.

In BDM mode, writing any value to TPMxSC, TPMxCNTH, or TPMxCNTL registers resets the read coherency mechanism of the TPMxCNTH:TPMxCNTL registers, regardless of the data involved in the write.

Address: FFFF_8120h base + 1h offset = FFFF_8121h



TPM_TPMxCNTH field descriptions

Field	Description
TPMxCNT	See full description above.

20.3.3 TPM Counter Register Low (TPM_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 which makes reads 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.

When BDM is active, the timer counter is frozen. (This is the value you read.) The coherency mechanism is frozen so 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 you were in the middle of reading a 16-bit register when BDM became active, it reads the appropriate value from the other half of the 16-bit value after returning to normal execution.

In BDM mode, writing any value to TPMxSC, TPMxCNTH, or TPMxCNTL registers resets the read coherency mechanism of the TPMxCNTH:TPMxCNTL registers, regardless of the data involved in the write.

Address: FFFF_8120h base + 2h offset = FFFF_8122h

Bit	7	6	5	4	3	2	1	0
Read	TPMxCNTL							
Write								
Reset	0	0	0	0	0	0	0	0

TPM_TPMxCNTL field descriptions

Field	Description
TPMxCNTL	See full description above.

20.3.4 TPM Counter Modulo Register High (TPM_TPMxMODH)

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 which results in a free-running timer counter (modulo disabled).

Writes to any of the registers TPMxMODH and TPMxMODL actually write to buffer registers and the registers are updated with the value of their write buffer according to the value of CLKSB:CLKSA bits:

- If CLKSB and CLKSA are cleared, the registers are updated when the second byte is written.
- If CLKSB and CLKSA are not cleared, the registers are updated after both bytes were written and the TPM counter changes from (TPMxMODH:TPMxMODL - 1) to (TPMxMODH:TPMxMODL). If the TPM counter is a free-running counter, the update is made when the TPM counter changes from 0xFFFE to 0xFFFF.

The latching mechanism is manually reset by writing to the TPMxSC address (whether BDM is active or not).

When BDM is active, the coherency mechanism is frozen (unless reset by writing to TPMxSC register) so the buffer latches remain in the state they were in when the BDM became active, even if one or both halves of the modulo register are written while BDM is active. Any write to the modulo registers bypasses the buffer latches and directly writes to the modulo register while BDM is active.

To avoid confusion about when the first counter overflow occurs, reset the TPM counter before writing to the TPM modulo registers.

Address: FFFF_8120h base + 3h offset = FFFF_8123h

Bit	7	6	5	4	3	2	1	0
Read Write	TPMxMODH							
Reset	0	0	0	0	0	0	0	0

TPM_TPMxMODH field descriptions

Field	Description
TPMxMODH	See full description above.

20.3.5 TPM Counter Modulo Register Low (TPM_TPMxMODHL)

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 which results in a free-running timer counter (modulo disabled).

Writes to any of the registers TPMxMODH and TPMxMODL actually write to buffer registers and the registers are updated with the value of their write buffer according to the value of CLKSB:CLKSA bits:

- If CLKSB and CLKSA are cleared, the registers are updated when the second byte is written.
- If CLKSB and CLKSA are not cleared, the registers are updated after both bytes were written and the TPM counter changes from (TPMxMODH:TPMxMODL - 1) to (TPMxMODH:TPMxMODL). If the TPM counter is a free-running counter, the update is made when the TPM counter changes from 0xFFFE to 0xFFFF.

The latching mechanism is manually reset by writing to the TPMxSC address (whether BDM is active or not).

When BDM is active, the coherency mechanism is frozen (unless reset by writing to TPMxSC register) so the buffer latches remain in the state they were in when the BDM became active, even if one or both halves of the modulo register are written while BDM is active. Any write to the modulo registers bypasses the buffer latches and directly writes to the modulo register while BDM is active.

To avoid confusion about when the first counter overflow occurs, reset the TPM counter before writing to the TPM modulo registers.

Address: FFFF_8120h base + 4h offset = FFFF_8124h

Bit	7	6	5	4	3	2	1	0
Read Write	TPMxMODL							
Reset	0	0	0	0	0	0	0	0

TPM_TPMxMODHL field descriptions

Field	Description
TPMxMODL	See full description above.

20.3.6 TPM Channel n Status and Control Register (TPM_TPMxCnSC)

TPMxC*n*SC contains the channel-interrupt-status flag and control bits that configure the interrupt enable, channel configuration and pin function.

Table 20-9. Mode, Edge, and Level Selection

CPWMS	MS*n*B:MS*n*A	ELS*n*B:ELS*n*A	Mode	Configuration
X	XX	00	Pin is not controlled by TPM. It is reverted to general-purpose I/O or other peripheral control	
0	00	01	Input capture	Capture on Rising Edge Only
		10		Capture on Falling Edge Only
		11		Capture on Rising or Falling Edge
	01	00	Output compare	Software-compare only
		01		Toggle output on channel match
		10		Clear output on channel match
		11		Set output on channel match
	1X	10	Edge-aligned PWM	High-true pulses clear output on channel match)
		X1		Low-true pulses (set output on channel match)
1	XX	10	Center-aligned PWM	High-true pulses (clear output on channel match when TPM counter is counting up)
		X1		Low-true pulses (set output on channel match when TPM counter is counting up)

Address: FFFF_8120h base + 5h offset = FFFF_8125h

Bit	7	6	5	4	3	2	1	0
Read Write	CHnF	CHnIE	MSnB	MSnA	ELSnB	ELSnA	Reserved	
Reset	0	0	0	0	0	0	0	0

TPM_TPMxCnSC field descriptions

Field	Description
7 CHnF	Channel *n* flag. When channel *n* is an input capture channel, this read/write bit is set when an active edge occurs on the channel *n* input. When channel *n* is an output compare or edge-aligned/center-aligned PWM channel, CH*n*F 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 percent or 100 percent, CH*n*F is not set even when the value in the TPM counter registers matches the value in the TPM channel *n* value registers. A corresponding interrupt is requested when this bit is set and channel *n* interrupt is enabled (CH*n*IE = 1). Clear CH*n*F by reading TPMxC*n*SC while this bit is set and then writing a logic 0 to it. If another interrupt request occurs before the clearing sequence is completed, CH*n*F remains set. This is done so a CH*n*F interrupt request is not lost due to clearing a previous CH*n*F.

Table continues on the next page...

TPM_TPMxCnSC field descriptions (continued)

Field	Description
	Reset clears this bit. Writing a logic 1 to CH*n*F has no effect. 0 No input capture or output compare event occurred on channel *n*. 1 Input capture or output-compare event on channel *n*.
6 CHnIE	Channel *n* interrupt enable. This read/write bit enables interrupts from channel *n*. Reset clears this bit. 0 Channel *n* interrupt requests disabled (Use this for software polling.) 1 Channel *n* interrupt requests enabled.
5 MSnB	Mode select B for TPM channel *n*. When CPWMS is cleared, setting the MS*n*B bit configures TPM channel *n* for edge-aligned PWM mode. Refer to the summary of channel mode and setup controls in the Mode, Edge and Level Selection Table.
4 MSnA	Mode select A for TPM channel *n*. When CPWMS and MS*n*B are cleared, the MS*n*A bit configures TPM channel *n* for input capture mode or output compare mode. Refer to the Mode, Edge and Level Selection Table for a summary of channel mode and setup controls. **NOTE:** 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 ELSnB	Edge/level select bits. Depending upon the operating mode for the timer channel as set by CPWMS:MS*n*B:MS*n*A and shown in the Mode, Edge and Level Selection Table, these bit select the polarity of the input edge that triggers an input capture event, select the level that is driven in response to an output compare match or select the polarity of the PWM output. If ELS*n*B and ELS*n*A bits are cleared, the channel pin is not controlled by TPM. This configuration can be used by software compare only because it does not require the use of a pin for the channel.
2 ELSnA	Edge/level select bits. Depending upon the operating mode for the timer channel as set by CPWMS:MS*n*B:MS*n*A and shown in the Mode, Edge and Level Selection Table, these bits select the polarity of the input edge that triggers an input capture event, select the level that is driven in response to an output compare match or select the polarity of the PWM output. If ELS*n*B and ELS*n*A bits are cleared, the channel pin is not controlled by TPM. This configuration can be used by software compare only because it does not require the use of a pin for the channel.
R	This field is reserved. Reserved or Unimplemented

20.3.7 TPM Channel Value Register High (TPM_TPMxCnVH)

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.

In input-capture mode, reading either byte (TPMxC*n*VH or TPMxC*n*VL) 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 TPMxC*n*SC register is written (whether BDM mode is active or not). Any write to the channel registers is ignored during the input-capture mode.

When BDM is active, the coherency mechanism is frozen (unless reset by writing to TPMxC*n*SC register) so the buffer latches remain in the state they were in when the BDM became active, even if one or both halves of the channel register are read while BDM is active. This assures that if you were in the middle of reading a 16-bit register when BDM became active, it reads the appropriate value from the other half of the 16-bit value after returning to normal execution. The value read from the TPMxC*n*VH and TPMxC*n*VL 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 (TPMxC*n*VH or TPMxC*n*VL) latches the value into a buffer. After both bytes were written, they are transferred as a coherent, 16-bit value into the timer-channel registers according to the value of CLKSB:CLKSA bits and the selected mode:

- If CLKSB and CLKSA are cleared, the registers are updated when the second byte is written.
- If CLKSB and CLKSA are not cleared and in output-compare mode, the registers are updated after the second byte is written and on the next change of the TPM counter (end of the prescaler counting).
- If CLKSB and CLKSA are not cleared and in the EPWM or CPWM mode, 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 0xFFFE to 0xFFFF.

The latching mechanism is manually reset by writing to the TPMxC*n*SC register (whether BDM mode is active or not). This latching mechanism allows coherent, 16-bit writes in either big-endian or little-endian order that is friendly to various compiler implementations.

When BDM is active, the coherency mechanism is frozen so 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 writes directly to the channel register while BDM is active. The values written to the channel register while BDM is active are used for PWM and output-compare operation after normal execution resumes. Writes to the channel registers while BDM is active do not interfere with partial completion of a coherency sequence. After the coherency mechanism is fully exercised, the channel registers are updated using the buffered values (while BDM was not active).

Address: FFFF_8120h base + 8h offset = FFFF_8128h

Bit	7	6	5	4	3	2	1	0
Read Write	TPMxCnVH							
Reset	0	0	0	0	0	0	0	0

TPM_TPMxCnVH field descriptions

Field	Description
TPMxCnVH	See full description above.

20.3.8 TPM Channel Value Register Low (TPM_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.

In input-capture mode, reading either byte (TPMxC*n*VH or TPMxC*n*VL) 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 TPMxC*n*SC register is written (whether BDM mode is active or not). Any write to the channel registers is ignored during the input-capture mode.

When BDM is active, the coherency mechanism is frozen (unless reset by writing to TPMxC*n*SC register) so the buffer latches remain in the state they were in when the BDM became active, even if one or both halves of the channel register are read while BDM is active. This assures that if you were in the middle of reading a 16-bit register when BDM became active, it reads the appropriate value from the other half of the 16-bit value after returning to normal execution. The value read from the TPMxC*n*VH and TPMxC*n*VL 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 (TPMxC*n*VH or TPMxC*n*VL) latches the value into a buffer. After both bytes were written, they are transferred as a coherent, 16-bit value into the timer-channel registers according to the value of CLKSB:CLKSA bits and the selected mode:

- If CLKSB and CLKSA are cleared, the registers are updated when the second byte is written.
- If CLKSB and CLKSA are not cleared and in output-compare mode, the registers are updated after the second byte is written and on the next change of the TPM counter (end of the prescaler counting).
- If CLKSB and CLKSA are not cleared and in the EPWM or CPWM mode, 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 0xFFFE to 0xFFFF.

The latching mechanism is manually reset by writing to the TPMxC*n*SC register (whether BDM mode is active or not). This latching mechanism allows coherent, 16-bit writes in either big-endian or little-endian order that is friendly to various compiler implementations.

When BDM is active, the coherency mechanism is frozen so 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 writes directly to the channel register while BDM is active. The values written to the channel register while BDM is active are used for PWM and output-compare operation after normal execution resumes. Writes to the channel registers while BDM is active do not interfere with partial completion of a coherency sequence. After the coherency mechanism is fully exercised, the channel registers are updated using the buffered values (while BDM was not active).

Address: FFFF_8120h base + Bh offset = FFFF_812Bh

Bit	7	6	5	4	3	2	1	0
Read Write	TPMxCnVL							
Reset	0	0	0	0	0	0	0	0

TPM_TPMxCnVL field descriptions

Field	Description
TPMxCnVL	See full description above.

20.4 Functional description

The following sections describe TPM counter and each of the timer operating modes (input capture, output compare, edge-aligned PWM and center-aligned PWM).

All TPM functions are associated with a central, 16-bit counter that allows flexible selection of the clock and prescale factor. There is also a 16-bit, modulo register associated with this counter.

The CPWMS control bit chooses between center-aligned PWM operation for all channels in the TPM (CPWMS = 1) or general-purpose timing functions (CPWMS = 0) where each channel can independently be configured to operate in input capture, output compare or edge-aligned PWM mode. The CPWMS control bit is located in the TPM status and control register because it affects all channels within the TPM and influences the way the main counter operates. (In CPWM mode, the counter changes to an up/down mode rather than the up-counting mode used for general-purpose timer functions.)

Note that details of pin operation and interrupt activity depend upon the operating mode, and so these topics are described in the associated mode explanation sections.

20.4.1 Counter

All timer functions are based on the main, 16-bit counter (TPMxCNTH:TPMxCNTL). This section discusses selection of the clock, end-of-count overflow, up-counting versus up/down counting and manual counter reset.

20.4.1.1 Counter clock source

The two-bit field CLKSB:CLKSA, in the timer status and control register (TPMxSC), disables the TPM counter or selects one of three clock sources to TPM counter (CLKSA). After any MCU reset, CLKSB and CLKSA are cleared so no clock is selected and the TPM counter is disabled (when the TPM is in a very-low-power state).

You can read or write these control bits at any time. Disabling the TPM counter by writing 00 to CLKSB:CLKSA bits does not affect the values in the TPM counter or other registers.

The fixed-frequency clock is an alternative clock source for the TPM counter that allows the selection of a clock other than the bus clock or external clock. This clock input is defined by chip integration. For further information, see Introduction. Due to TPM hardware implementation limitations, the frequency of the fixed-frequency clock must not exceed the bus clock frequency. The fixed-frequency clock has no limitations for low frequency operation.

20.4.1.2 Counter overflow and modulo reset

An interrupt flag and enable are associated with the 16-bit, main counter. The flag (TOF) is a software-accessible indication that the timer counter has overflowed. The enable signal selects between

- software polling (TOIE = 0), where no interrupt is generated
- or interrupt-driven operation (TOIE = 1), where the interrupt is generated whenever the TOF is set.

The conditions causing TOF to become set depend on whether the TPM is configured for center-aligned PWM (CPWMS = 1).

- If CPWMS is cleared (CPWMS = 0) and there is no modulus limit, then the 16-bit timer counter counts from 0x0000 through 0xFFFF, and overflows to 0x0000 on the next counting clock. TOF is set at the transition from 0xFFFF to 0x0000. If CPWMS is cleared (CPWMS = 0) and a modulus limit is set, then TOF is set at the transition from the value set in the modulus register to 0x0000.
- When the TPM is in center-aligned PWM mode (CPWMS = 1), then the TOF flag is set as the counter changes direction at the end of the count value that is 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.)

20.4.1.3 Counting modes

The main timer counter has two counting modes:

- Up/down counting mode
- Up-counting mode

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.

- When center-aligned PWM operation is specified, the counter counts up from 0x0000 through its terminal count, and then counts down to 0x0000 (where it changes back to up-counting). The terminal count value and 0x0000 are normal length counts (one timer clock period long). In this mode, the Timer Overflow Flag (TOF) is set at the end of the terminal-count period (as the count changes to the next-lower count value).
- As an up-counter, the timer counter counts from 0x0000 through its terminal count and continues with 0x0000. The terminal count is 0xFFFF or a modulus value in TPMxMODH:TPMxMODL.

20.4.1.4 Manual counter reset

The main timer counter can be manually reset at any time, by writing any value to TPMxCNTH or TPMxCNTL. Resetting the counter in this manner also resets the coherency mechanism (in case only half of the counter was read before resetting the count).

20.4.2 Channel mode selection

If CPWMS is cleared, then the MS*n*B and MS*n*A bits determine the basic mode of operation for the corresponding channel. Choices of operating modes for channels include input capture, output compare and edge-aligned PWM.

20.4.2.1 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 (TPMxC*n*VH:TPMxC*n*VL). Rising edges, falling edges or any edge is chosen as the active edge that triggers an input capture.

In input capture mode, the TPMxC*n*VH and TPMxC*n*VL registers are read-only.

When either half of the 16-bit capture register is read, the other half is latched into a buffer to support coherent, 16-bit accesses in big-endian or little-endian order. The coherency sequence can be manually reset by writing to TPMxC*n*SC.

An input capture event sets a flag bit (CH*n*F) that optionally generates a CPU interrupt request.

While in BDM, the input-capture function works as configured. When an external event occurs, the TPM latches the contents of the TPM counter (frozen because of the BDM mode) into the channel-value registers and sets the flag bit.

20.4.2.2 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 TPMxC*n*VH:TPMxC*n*VL registers of an output-compare channel, the TPM can set, clear or toggle the channel pin.

Writes to any of TPMxC*n*VH and TPMxC*n*VL registers actually write to buffer registers. In output-compare mode, the TPMxC*n*VH:TPMxC*n*VL registers are updated with the value of their write buffer only after both bytes were written and according to the value of CLKSB:CLKSA bits:

- If CLKSB and CLKSA are cleared, the registers are updated when the second byte is written.
- If CLKSB and CLKSA are not cleared, the registers are updated at the next change of the TPM counter (the 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 (TPMxC*n*SC).

An output compare event sets a flag bit (CHnF) that optionally generates a CPU interrupt request.

20.4.2.3 Edge-aligned PWM mode

This type of PWM output uses the normal up-counting mode of the timer counter (CPWMS = 0) and can be used when other channels in the same TPM are configured for input-capture or output-compare functions.

- The period of this PWM signal is determined by the value of the modulus register (TPMxMODH:TPMxMODL) plus 1.
- The duty cycle is determined by the value of the timer-channel register (TPMxC*n*VH:TPMxC*n*VL). Zero-percent and 100-percent duty-cycle cases are possible.
- The polarity of this PWM signal is determined by ELS*n*A bit.

The time between the modulus overflow and the channel match value (TPMxC*n*VH:TPMxC*n*VL) is the pulse width or duty cycle (Figure 20-14).

- If ELS*n*A is cleared, then the counter overflow forces the PWM signal high, and the channel match forces the PWM signal low.
- If ELS*n*A is set, then the counter overflow forces the PWM signal low, and the channel-match forces the PWM signal high.

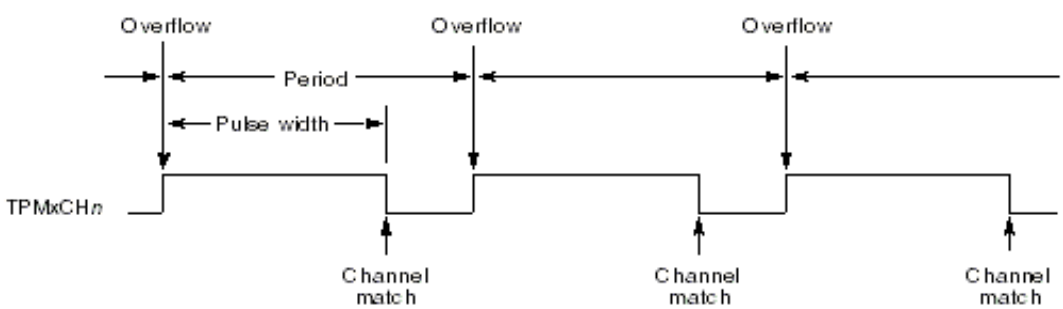


Figure 20-14. EPWM Period and Pulse Width (ELSnA = 0)

When the channel-value register is set to 0x0000, the duty cycle is 0 percent. A 100-percent duty cycle is achieved by setting the timer-channel register (TPMxC*n*VH:TPMxC*n*VL) to a value greater than the modulus setting. This implies that to get a 100-percent duty cycle, the modulus setting must be less than 0xFFFF.

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 TPMxC*n*VH and TPMxC*n*VL actually write to buffer registers. In edge-aligned PWM mode, the TPMxC*n*VH:TPMxC*n*VL registers are updated with the value of their write buffer, according to the value of the CLKSB:CLKSA bits:

- If CLKSB and CLKSA are cleared, then the registers are updated when the second byte is written.
- If CLKSB and CLKSA are not cleared, 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, then the update is made when the TPM counter changes from 0xFFFE to 0xFFFF.

20.4.2.4 Center-aligned PWM mode

This type of PWM output uses the up/down counting mode of the timer counter (CPWMS = 1). The channel-match value in TPMxC*n*VH:TPMxC*n*VL determines the pulse width (duty cycle) of the PWM signal; the value in TPMxMODH:TPMxMODL determines the period.

TPMxMODH:TPMxMODL must be kept in the range of 0x0001 to 0x7FFF, because values outside this range can produce ambiguous results. ELS*n*A determines the polarity of the CPWM signal.

pulse width = 2 x (TPMxC*n*VH:TPMxC*n*VL)

period = 2 x (TPMxMODH:TPMxMODL); TPMxMODH:TPMxMODL = 0x0001-0x7FFF

- If TPMxC*n*VH:TPMxC*n*VL is zero or negative (Bit 15 is set), then the duty cycle is 0 percent.
- If TPMxC*n*VH:TPMxC*n*VL is a positive value (Bit 15 clear) and greater than the non-zero modulus setting, then the duty cycle is 100 percent because the channel-match never occurs.

This implies the usable range of periods set by the modulus register is 0x0001 through 0x7FFE (or 0x7FFF, if you do not need to generate a 100-percent duty cycle). Note that this is not a significant limitation, because the resulting period is much longer than required for normal applications.

All 0s in TPMxMODH:TPMxMODL is a special case that must not be used with center-aligned PWM mode.

- When CPWMS is cleared, this case corresponds to the counter running free from 0x0000 through 0xFFFF.
- When CPWMS is set, the counter needs a valid match to the modulus register somewhere other than at 0x0000, to change from up-counting to down-counting.

The channel-match value in the TPM channel registers (times two) determines the pulse width (duty cycle) of the CPWM signal (Figure 20-15). If ELS*n*A is cleared, then

- a channel match occurring while counting up clears the CPWM output signal,
- and a channel match occurring while counting down sets the CPWM output signal.

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.

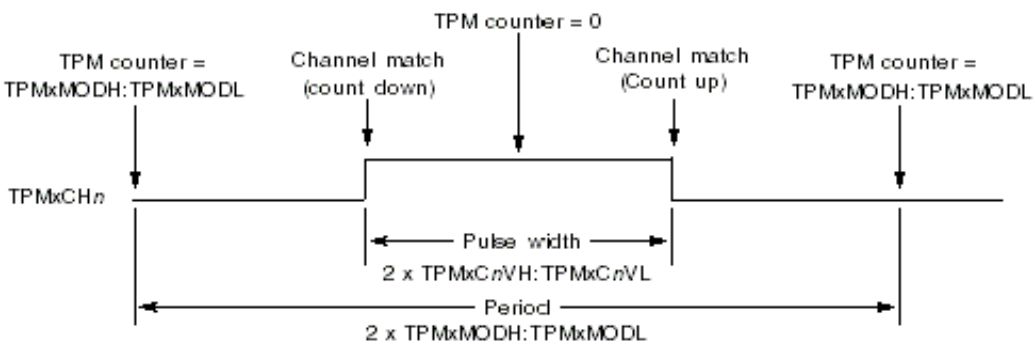


Figure 20-15. CPWM Period and Pulse Width (ELS*n*A = 0)

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 required for some types of motor drives.

Input capture, output compare and edge-aligned PWM functions do not make sense when the counter is operating in up/down-counting mode, so this implies that all active channels within a TPM must be used in CPWM mode when CPWMS is set.

The timer-channel registers are buffered to ensure coherent, 16-bit updates and avoid unexpected PWM pulse widths. Writes to any of the registers TPMxC*n*VH and TPMxC*n*VL actually write to buffer registers. In center-aligned PWM mode, the TPMxC*n*VH:TPMxC*n*VL registers are updated with the value of their write buffer, according to the value of the CLKSB:CLKSA bits:

- If CLKSB and CLKSA are cleared, then the registers are updated when the second byte is written.
- If CLKSB and CLKSA are not cleared, 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 0xFFFE to 0xFFFF.

When TPMxCNTH:TPMxCNTL equals TPMxMODH:TPMxMODL, the TPM can optionally generate a TOF interrupt (at the end of this count).

20.5 Reset

The TPM is reset whenever any MCU reset occurs. Reset clears TPMxSC, which disables the TPM counter clock and overflow interrupt (TOIE=0). CPWMS, MSnB, MSnA, ELSnB and ELSnA are all cleared. This configures all TPM channels for input capture operations, and the associated pins are not controlled by TPM.

20.6 Interrupts

When interrupts occur and what they do depends on the mode (input capture, output capture, PWM) selected for the channel.

20.6.1 Two types of interrupts

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, then 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, then the interrupt flag is set each time that the main timer counter matches the value in the 16-bit, channel-value register.

All TPM interrupts are listed in Table 20-12.

Table 20-12. Interrupt Summary

Interrupt	Local Enable	Source	Description
TOF	TOIE	Counter Overflow	Set each time the TPM counter reaches its terminal count (at transition to its next count value)
CH*n*F	CH*n*IE	Channel Event	An input capture event or channel match took place on channel *n*

The TPM module provides high-true interrupt signals.

20.6.2 Interrupt operation

For each interrupt source in the TPM, a flag bit is set upon recognition of the interrupt condition (such as timer overflow, channel input-capture or output-compare events). This flag is read (polled) by software to determine that the action has occurred or that an associated enable bit (TOIE or CH*n*IE) can be set to enable the interrupt generation. While the interrupt-enable bit is set, the interrupt is generated whenever the associated interrupt flag is set. Before returning from the interrupt-service routine, software must perform a sequence of steps to clear the interrupt flag.

TPM interrupt flags are cleared by a two-step process including a read of the flag bit (while the flag bit is set), followed by a write of 0 to the flag bit. If a new event is detected between these two steps, then the sequence is reset, and the interrupt flag remains set after the second step (to avoid the possibility of missing the new event).

20.6.2.1 Timer Overflow interrupt (TOF)

The meaning and details of operation for TOF interrupts varies slightly, depending on the mode of operation of the TPM system (general-purpose timing functions or center-aligned PWM operation). The flag is cleared by the two-step sequence described previously.

20.6.2.1.1 Normal case

When CPWMS is cleared, TOF is set when the timer counter changes from the terminal count (the value in the modulo register) to 0x0000. If the TPM counter is a free-running counter, the update is made when the TPM counter changes from 0xFFFF to 0x0000.

20.6.2.1.2 Center-aligned PWM case

When CPWMS is set, TOF is set when the timer counter changes direction from up-counting to down-counting at the end of the terminal count (the value in the modulo register).

20.6.2.2 Channel event interrupt

The meaning of channel interrupts depends on the channel's current mode (input capture, output compare, edge-aligned PWM or center-aligned PWM).

20.6.2.2.1 Input capture events

When a channel is configured as an input-capture channel, the ELS*n*B:ELS*n*A bits select if the channel pin is not controlled by TPM, rising edges, falling edges or any edge as the edge that triggers an input-capture event. When the selected edge is detected, the interrupt flag is set.

The flag is cleared by the two-step sequence described in Interrupt operation.

20.6.2.2.2 Output compare events

When a channel is configured as an output-compare channel, the interrupt flag is set each time the main timer counter matches the 16-bit value in the channel-value register. The flag is cleared by the two-step sequence described in Interrupt operation.

20.6.2.2.3 PWM end-of-duty-cycle events

When the channel is configured for edge-aligned PWM, the channel flag is set when the timer counter matches the channel-value register that marks the end of the active, duty-cycle period.

When the channel is configured for center-aligned PWM, the timer count matches the channel-value register twice during each PWM cycle. In this CPWM case, the channel flag is set at the start and end of the active, duty-cycle period when the timer counter matches the channel value register.

The flag is cleared by the two-step sequence described in Interrupt operation.

Chapter 21
ColdFire v1 Core (CF1_CORE)

21.1 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*.

21.1.1 Overview

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

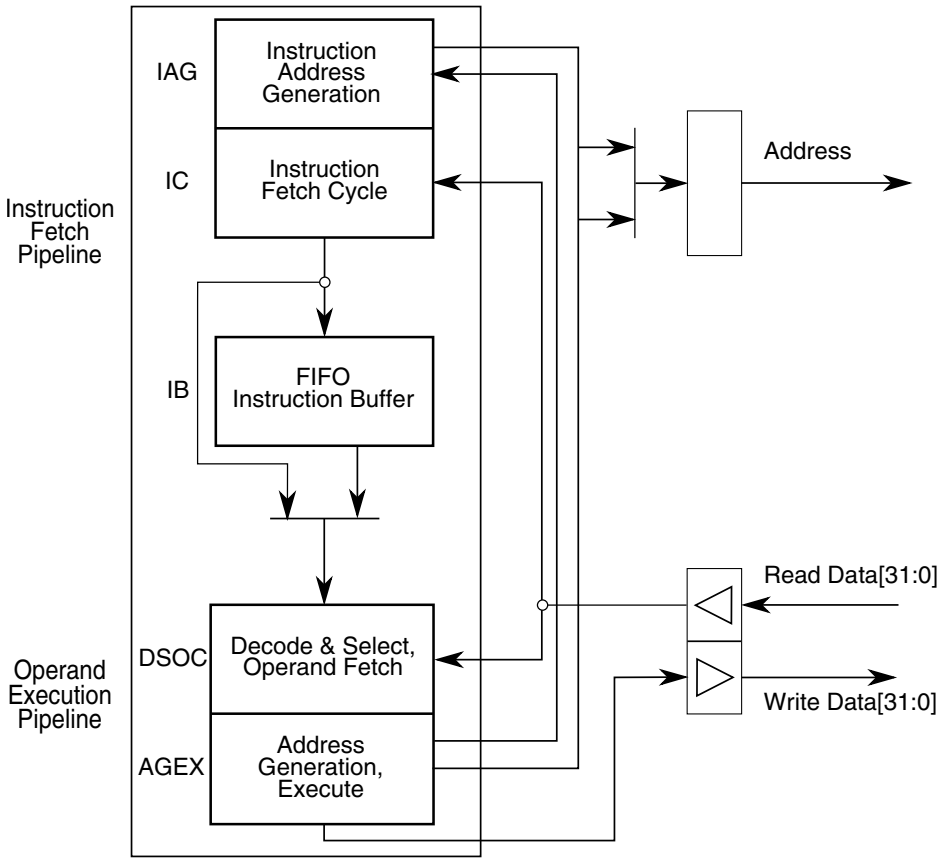


Figure 21-1. V1 ColdFire Core Pipelines

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.

21.2 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).

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)
- MAC registers (refer to EMAC description)
  - One 32-bit accumulator (ACC) register
  - One 16-bit mask register (MASK)
  - One 8-bit status register (MACSR)

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 21-1. ColdFire core programming model

BDM command[1]	Register[2]	Width (bits)	Access	Reset value	Written with MOVEC[3]
Supervisor/user access registers					
Load: 0x60 Store: 0x40	Data register 0 (D0) see Data registers (D0–D7)	32	R/W	See Reset Exception	No
Load: 0x61 Store: 0x41	Data register 1 (D1) see Data registers (D0–D7)	32	R/W	See Reset Exception	No
Load: 0x62-7 Store: 0x42-7	Data registers 2-7 (D2-7) see Data registers (D0–D7)	32	R/W	POR: Undefined Else: Unaffected	No
Load: 0x68-E Store: 0x48-E	Address registers (A0–A6)	32	R/W	POR: Undefined Else: Unaffected	No
Load: 0x6F Store: 0x4F	User stack pointer see Supervisor/user stack pointers (A7 and OTHER_A7)	32	R/W	POR: Undefined Else: Unaffected	No
Load: 0xE4 Store: 0xC4	MAC status register (MACSR)	8	R/W	0x00	No
Load: 0xE5 Store: 0xC5	MAC address mask register (MASK)	16	R/W	0xFFFF	No
Load: 0xE6 Store: 0xC6	MAC accumulator (ACC)	32	R/W	POR: Undefined Else: Unaffected	No
Load: 0xEE Store: 0xCE	Condition code register (CCR) (LSB of Status register (SR))	8	R/W	POR: Undefined Else: Unaffected	No
Load: 0xEF Store: 0xCF	Program counter (PC)	32	R/W	Contents of location 0x(00)00_0004	No
Supervisor access only registers					

Table continues on the next page...

Table 21-1. ColdFire core programming model (continued)

BDM command[1]	Register[2]	Width (bits)	Access	Reset value	Written with MOVEC[3]
Load: 0xE0 Store: 0xC0	Supervisor stack pointer see Supervisor/user stack pointers (A7 and OTHER_A7)	32	R/W	Contents of location 0x(00)00_0000	No
Load: 0xE1 Store: 0xC1	Vector base register (VBR)	32	R/W	See register's description	Yes Rc = 0x801
Load: 0xE2 Store: 0xC2	CPU configuration register (CPUCR)	32	W	See register's description	Yes Rc = 0x802
Load: 0xEE Store: 0xCE	Status register (SR)	16	R/W	0x27--	No

1. 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 details, see information about ColdFire debug operation. (These BDM commands are not similar to those of non-V1 ColdFire processors.)
2. The MAC registers are available only if the MAC module is present on the device.
3. If the given register is written using the MOVEC instruction, the 12-bit control register address (Rc) is also specified.

21.2.1 Data registers (D0–D7)

These 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

The D0 and D1 registers contain hardware configuration details after reset. See Reset Exception for more details.

Table 21-2. Data registers (D0–D7)

BDM:	Load: 0x60 + *n*; *n* = 0–7 (D*n*) Store: 0x40 + *n*; *n* = 0–7 (D*n*)												Access: User read/write BDM read/write			
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Data															
Reset (D2–D7)	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
Reset (D0, D1)	See Reset Exception															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Data															
Reset (D2–D7)	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Table continues on the next page...

Table 21-2. Data registers (D0–D7) (continued)

Reset (D0, D1)	See Reset Exception

21.2.2 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.

Table 21-3. Address registers (A0–A6)

BDM:	Load: 0x68 + *n*; *n* = 0–6 (A*n*) Store: 0x48 + *n*; *n* = 0–6 (A*n*)												Access: User read/write BDM read/write			
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Address															
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Address															
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

21.2.3 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
```

The *ColdFire Family Programmer's Reference Manual* describes these instructions. 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.

Table 21-4. Stack pointer registers (A7 and OTHER_A7)

BDM:	Load: 0x6F (A7) Store: 0x4F (A7) Load: 0xE0 (OTHER_A7) Store: 0xC0 (OTHER_A7)												Access: A7: User or BDM read/write OTHER_A7: Supervisor or BDM read/write			
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Address															
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Address															
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

21.2.4 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 CMP, Bcc, or Scc instructions are executed.

Table 21-5. Condition code register (CCR)

BDM:	LSB of status register (SR) Load: 0xEE (SR) Store: 0xCE (SR)							Access: User read/write BDM read/write
	7	6	5	4	3	2	1	0
R	0	0	0	X	N	Z	V	C
W								
Reset	0	0	0	–	–	–	–	–

Table 21-6. CCR field descriptions

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

21.2.5 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.

Table 21-7. Program counter (PC) register

BDM:	Load: 0xEF (PC) Store: 0xCF (PC)												Access: User read/write BDM read/write			
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	Address							
W																
Reset	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	Address															
W																
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

21.2.6 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.

Table 21-8. Vector base register (VBR)

BDM:	0x801 (VBR) Load: 0xE1 (VBR) Store: 0xC1 (VBR)												Access: Supervisor read/write BDM read/write			
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	Base Address				–	–	–	–
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	–	–	–	–
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
W																
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

21.2.7 CPU configuration register (CPUCR)

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

Table 21-9. CPU configuration register (CPUCR)

BDM:	0x802 (CPUCR) Load: 0xE2 (CPUCR) Store: 0xC2 (CPUCR)												Access: Supervisor read/write BDM read/write			
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	ARD	IRD	IAE	IME	BWD	–	FSD	–	–	–	–	–	0	–	–	–
W													–			
Reset	0	0	0	0	0	0	0	0	–	0	0	0	0	–	–	–
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
W																
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Table 21-10. 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.

Table continues on the next page...

Table 21-10. CPUCR field descriptions (continued)

Field	Description
	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.
27 BWD	Buffered write disable The ColdFire core is capable of marking processor memory writes as bufferable or non-bufferable. **NOTE:** If buffered writes are enabled (BWD is 0), any error status is lost as the immediate termination of the data transfer assumes an error-free completion. 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.
25 FSD	Flash speculation disable This bit controls whether prefetching by the speculation buffer is enabled. When enabled, prefetching occurs only for program flash accesses. Disabling prefetching also clears the current prefetch buffer. 0 Prefetching is enabled. 1 Prefetching is disabled.
26,24–0	Reserved; must be cleared.

21.2.8 Status register (SR)

This register 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 (the 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.

Table 21-11. Status register (SR)

BDM: Load: 0xEE (SR) Store: 0xCE (SR) Access: Supervisor read/write BDM read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	T	0	S	M	0	I			0	0	0	X	N	Z	V	C
W																
Reset	0	0	1	0	0	1	1	1	0	0	0	–	–	–	–	–

Table 21-12. SR field descriptions

Field	Description
15 T	Trace enable When this bit is set, the processor performs a trace exception after every instruction.
14	Reserved; must be cleared.
13 S	Supervisor/user state 0 User mode 1 Supervisor mode
12 M	Master/interrupt state This 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	Refer to Condition code register (CCR)

21.3 Functional Description

21.3.1 Instruction Set Architecture

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

The following table 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 21-13. Instruction Enhancements over Revision ISA_A

Instruction	Description
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.

21.3.2 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 21-2 , 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 × 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 21-14). 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 the interrupt chapter 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 21-14. Exception Vector Assignments

Vector Number(s)	Vector Offset (Hex)	Stacked Program Counter[1]	Assignment
0	0x000	—	Initial supervisor stack pointer
1	0x004	—	Initial program counter
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

1. 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*.

21.3.2.1 Exception Stack Frame Definition

The following figures shows the 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.

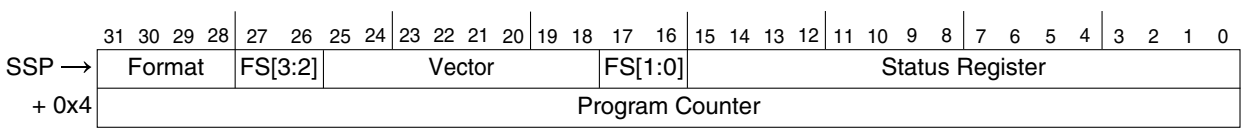


Figure 21-2. Exception Stack Frame Form

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 21-15 .

Table 21-15. 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 21-16 .

Table 21-16. 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 21-14 .

21.3.2.2 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 21-17. 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

Table continues on the next page...

Table 21-17. Exception Processing Comparison (continued)

Attribute	S08	V1 ColdFire
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.

21.3.3 Processor Exceptions

21.3.3.1 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.

21.3.3.2 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.

21.3.3.3 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. The opword line definition is shown below.

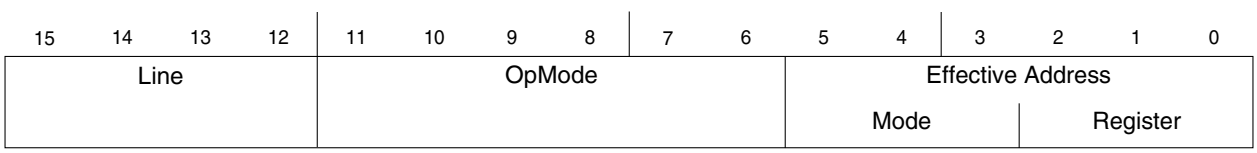


Figure 21-3. ColdFire Instruction Operation Word (Opword) Format

Table 21-18. 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	MAC, 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 non-MAC 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.

21.3.3.4 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.

21.3.3.5 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.

21.3.3.6 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.

21.3.3.7 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.

21.3.3.8 Debug Interrupt

See the debug chapter 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.

21.3.3.9 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.

21.3.3.10 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.

21.3.3.11 Unsupported Instruction Exception

If execution of a valid instruction is attempted but the required hardware is not present in the processor (e.g., if the MAC is not present), 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 Reset Exception," for details.

21.3.3.12 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 the interrupt chapter for details on the interrupt controller.

21.3.3.13 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 to exit this state.

21.3.3.14 Reset Exception

Resetting 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 (such as 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 Table 21-19.

Table 21-19. D0 Hardware Configuration Information

BDM:	Load: 0x60 (D0) Store: 0x40 (D0)												Access: User read-only BDM read-only			
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	PF								VER				REV			
W																
Reset	1	1	0	0	1	1	1	1	0	0	0	1	Device-specific			
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	MAC	DIV	0	0	0	CAU	0	0	ISA				DEBUG			
W																
Reset	1	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1

Table 21-20. D0 Hardware Configuration Information Field Descriptions

Field	Description
31–24 PF	Processor family. This field is fixed to a hex value of 0xCF indicating a ColdFire core is present.
23–20 VER	ColdFire core version number. Defines the hardware microarchitecture version of ColdFire core. 0001 V1 ColdFire core
19–16 REV	Processor revision number
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 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 1 Divide execute engine is present in core
13	Reserved
12	Reserved
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
9–8	Reserved
7–4 ISA	ISA revision. Defines the instruction-set architecture (ISA) revision level implemented in ColdFire processor core. 0010 ISA_C

Table continues on the next page...

Table 21-20. D0 Hardware Configuration Information Field Descriptions (continued)

Field	Description
3–0 DEBUG	Debug module revision number. Defines revision level of the debug module used in the ColdFire processor core. 1001 DEBUG_B+

Information loaded into D1 defines the local memory hardware configuration as shown in the following tables.

Table 21-21. D1 Hardware Configuration Information

BDM: Load: 0x61 (D1) Access: User read-only
Store: 0x41 (D1) BDM read-only

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

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

Table 21-22. D1 Hardware Configuration Information Field Descriptions

Field	Description
31–24	Reserved
23–19 FLASHSZ	Flash memory bank size 00000-01110: No flash 10000: 64 KB flash 10010: 128 KB flash 10011: 96 KB flash 10100: 256 KB flash 10110: 512 KB flash Else: Reserved for future use
18–8	Reserved
7–3 SRAMSZ	SRAM bank size 00000: No SRAM 00010: 512 bytes 00100: 1 KB 00110: 2 KB 01000: 4 KB

Table continues on the next page...

Table 21-22. D1 Hardware Configuration Information Field Descriptions (continued)

Field	Description
	01010: 8 KB 01100: 16 KB 01101: 24 KB 01110: 32 KB 10000: 64 KB 10010: 128 KB Else: Reserved for future use
2–0	Reserved

21.3.4 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.

21.3.4.1 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 21-23.

Table 21-23. 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

21.3.4.2 MOVE Instruction Execution Times

Table 21-25 lists execution times for MOVE.{B,W} instructions. Table 21-26 lists execution times for MOVE.L.

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

Table 21-24. Effective addressing modes with equal execution time

PC-relative effective addressing mode	An-relative effective addressing mode
ET with {<ea> = (d16,PC)}	ET with {<ea> = (d16,An)}
ET with {<ea> = (d8,PC,Xi*SF)}	ET with {<ea> = (d8,An,Xi*SF)}

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

Table 21-25. MOVE Byte and Word Execution Times

Source	Destination						
	RX	(Ax)	(Ax)+	-(Ax)	(d16,Ax)	(d8,Ax,Xi*SF)	xxx.wl
Dy	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)
Ay	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)
(Ay)	2 (1/0)	3 (1/1)	3 (1/1)	3 (1/1)	3 (1/1)	4 (1/1))	3 (1/1)
(Ay)+	2 (1/0)	3 (1/1)	3 (1/1)	3 (1/1)	3 (1/1)	4 (1/1))	3 (1/1)
-(Ay)	2 (1/0)	3 (1/1)	3 (1/1)	3 (1/1)	3 (1/1)	4 (1/1))	3 (1/1)
(d16,Ay)	2 (1/0)	3 (1/1)	3 (1/1)	3 (1/1)	3 (1/1)	—	—
(d8,Ay,Xi*SF)	3 (1/0)	4 (1/1)	4 (1/1)	4 (1/1)	—	—	—
xxx.w	2 (1/0)	3 (1/1)	3 (1/1)	3 (1/1)	—	—	—
xxx.l	2 (1/0)	3 (1/1)	3 (1/1)	3 (1/1)	—	—	—
(d16,PC)	2 (1/0)	3 (1/1)	3 (1/1)	3 (1/1)	3 (1/1)	—	—
(d8,PC,Xi*SF)	3 (1/0)	4 (1/1)	4 (1/1)	4 (1/1))	—	—	—
#xxx	1(0/0)	3 (0/1)	3 (0/1)	3 (0/1)	1(0/1)	—	—

Table 21-26. 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)	—	—	—

21.3.4.3 Standard One Operand Instruction Execution Times

Table 21-27. 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)	—	—	—	—	—	—	—

Table continues on the next page...

Table 21-27. One Operand Instruction Execution Times (continued)

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An)	(d8,An,Xn*SF)	xxx.wl	#xxx
BYTEREV	Dx	1(0/0)	—	—	—	—	—	—	—
CLR.B	<ea>	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)	—
CLR.W	<ea>	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)	—
CLR.L	<ea>	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)	—
EXT.W	Dx	1(0/0)	—	—	—	—	—	—	—
EXT.L	Dx	1(0/0)	—	—	—	—	—	—	—
EXTB.L	Dx	1(0/0)	—	—	—	—	—	—	—
FF1	Dx	1(0/0)	—	—	—	—	—	—	—
NEG.L	Dx	1(0/0)	—	—	—	—	—	—	—
NEGX.L	Dx	1(0/0)	—	—	—	—	—	—	—
NOT.L	Dx	1(0/0)	—	—	—	—	—	—	—
SATS.L	Dx	1(0/0)	—	—	—	—	—	—	—
SCC	Dx	1(0/0)	—	—	—	—	—	—	—
SWAP	Dx	1(0/0)	—	—	—	—	—	—	—
TAS.B	<ea>	—	3 (1/1)	3 (1/1)	3 (1/1)	3 (1/1)	4 (1/1)	3 (1/1)	—
TST.B	<ea>	1(0/0)	2 (1/0)	2 (1/0)	2 (1/0)	2 (1/0)	3 (1/0)	2 (1/0)	1(0/0)
TST.W	<ea>	1(0/0)	2 (1/0)	2 (1/0)	2 (1/0)	2 (1/0)	3 (1/0)	2 (1/0)	1(0/0)
TST.L	<ea>	1(0/0)	2 (1/0)	2 (1/0)	2 (1/0)	2 (1/0)	3 (1/0)	2 (1/0)	1(0/0)

21.3.4.4 Standard Two Operand Instruction Execution Times

Table 21-28. 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)	—

Table continues on the next page...

Table 21-28. 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
BCHG	#imm,<ea>	2(0/0)	4 (1/1)	4 (1/1)	4 (1/1)	4 (1/1)	—	—	—
BCLR	Dy,<ea>	2(0/0)	4 (1/1)	4 (1/1)	4 (1/1)	4 (1/1)	5 (1/1)	4 (1/1)	—
BCLR	#imm,<ea>	2(0/0)	4 (1/1)	4 (1/1)	4 (1/1)	4 (1/1)	—	—	—
BSET	Dy,<ea>	2(0/0)	4 (1/1)	4 (1/1)	4 (1/1)	4 (1/1)	5 (1/1)	4 (1/1)	—
BSET	#imm,<ea>	2(0/0)	4 (1/1)	4 (1/1)	4 (1/1)	4 (1/1)	—	—	—
BTST	Dy,<ea>	2(0/0)	3 (1/0)	3 (1/0)	3 (1/0)	3 (1/0)	4 (1/0)	3 (1/0)	—
BTST	#imm,<ea>	1(0/0)	3 (1/0)	3 (1/0)	3 (1/0)	3 (1/0)	—	—	—
CMP.B	<ea>,Rx	1(0/0)	3 (1/0)	3 (1/0)	3 (1/0)	3 (1/0)	4 (1/0)	3 (1/0)	1(0/0)
CMP.W	<ea>,Rx	1(0/0)	3 (1/0)	3 (1/0)	3 (1/0)	3 (1/0)	4 (1/0)	3 (1/0)	1(0/0)
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)	—	—	—	—	—	—	—

21.3.4.5 Miscellaneous Instruction Execution Times

Table 21-29. 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)	—	—	—	—	—	—	—

Table continues on the next page...

Table 21-29. Miscellaneous Instruction Execution Times (continued)

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An)	(d8,An,Xn*SF)	xxx.wl	#xxx
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)	—	—	—	—	—	—	—
MOVE.W	<ea>,SR	7 (0/0)	—	—	—	—	—	—	7 (0/0) [1]
MOVEC	Ry,Rc	9 (0/1)	—	—	—	—	—	—	—
MOVEM.L	<ea>,and list	—	1+ n(n/0) [2]	—	—	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) [3]	3 (0/1) [4]	2 (0/1)	—
PULSE		1(0/0)	—	—	—	—	—	—	—
STLDSR	#imm	—	—	—	—	—	—	—	5(0/1)
STOP	#imm	—	—	—	—	—	—	—	3 (0/0) [5]
TRAP	#imm	—	—	—	—	—	—	—	13 (1/2)
TPF		1(0/0)	—	—	—	—	—	—	—
TPF.W		1(0/0)	—	—	—	—	—	—	—
TPF.L		1(0/0)	—	—	—	—	—	—	—
UNLK	Ax	2 (1/0)	—	—	—	—	—	—	—
WDDATA	<ea>	—	3 (1/0)	3 (1/0)	3 (1/0)	3 (1/0)	4 (1/0)	3 (1/0)	—
WDEBUG	<ea>	—	5 (2/0)	—	—	5 (2/0)	—	—	—

1. If a MOVE.W #imm,SR instruction is executed and imm[13] equals 1, the execution time is 1(0/0).
2. The n is the number of registers moved by the MOVEM opcode.
3. PEA execution times are the same for (d16,PC).
4. PEA execution times are the same for (d8,PC,Xn*SF).
5. The execution time for STOP is the time required until the processor begins sampling continuously for interrupts.

21.3.4.6 MAC Instruction Execution Times

Table 21-30. MAC Instruction Execution Times

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An)	(d8,An, Xn*SF)	xxx.wl	#xxx
MAC.L	Ry, Rx	3 (0/0)	—	—	—	—	—	—	—
MAC.L	Ry, Rx, <ea>, Rw	—	5 (1/0)	5 (1/0)	5 (1/0)	5 (1/0)[1]	—	—	—
MAC.W	Ry, Rx	1(0/0)	—	—	—	—	—	—	—
MAC.W	Ry, Rx, <ea>, Rw	—	2 (1/0)	2 (1/0)	2 (1/0)	2 (1/0)	—	—	—
MOVE.L	<ea>y, Racc	1(0/0)	—	—	—	—	—	—	1(0/0)
MOVE.L	<ea>y, MACSR	2 (0/0)	—	—	—	—	—	—	2 (0/0)
MOVE.L	<ea>y, Rmask	1 (0/0)	—	—	—	—	—	—	1 (0/0)
MOVE.L	Racc,<ea>x	1(0/0)[2]	—	—	—	—	—	—	—
MOVE.L	MACSR,<ea>x	1(0/0)	—	—	—	—	—	—	—
MOVE.L	Rmask, <ea>x	1(0/0)	—	—	—	—	—	—	—
MSAC.L	Ry, Rx	3 (0/0)	—	—	—	—	—	—	—
MSAC.W	Ry, Rx	1(0/0)	—	—	—	—	—	—	—
MSAC.L	Ry, Rx, <ea>, Rw	—	5 (1/0)	5 (1/0)	5 (1/0)	5 (1/0) [1]	—	—	—
MSAC.W	Ry, Rx, <ea>, Rw	—	2 (1/0)	2 (1/0)	2 (1/0)	2 (1/0) [1]	—	—	—
MULS.L	<ea>y, Dx	18 (0/0)	20 (1/0)	20 (1/0)	20 (1/0)	20 (1/0)	—	—	—
MULS.W	<ea>y, Dx	9 (0/0)	11 (1/0)	11 (1/0)	11 (1/0)	11 (1/0)	12 (1/0)	11 (1/0)	9 (0/0)
MULU.L	<ea>y, Dx	18 (0/0)	20 (1/0)	20 (1/0)	20 (1/0)	20 (1/0)	—	—	—
MULU.W	<ea>y, Dx	9 (0/0)	11 (1/0)	11 (1/0)	11 (1/0)	11 (1/0)	12 (1/0)	11 (1/0)	9 (0/0)

1. Effective address of (d16,PC) not supported
2. Storing the accumulator requires one additional processor clock cycle when rounding is performed

21.3.4.7 Branch Instruction Execution Times

Table 21-31. General Branch Instruction Execution Times

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d_{16},An) (d_{16},PC)	(d8,An,Xi*SF) (d8,PC,Xi*SF)	xxx.wl	#xxx
BRA		—	—	—	—	2 (0/1)	—	—	—
BSR		—	—	—	—	3 (0/1)	—	—	—
JMP	<ea>	—	3 (0/0)	—	—	3 (0/0)	4 (0/0)	3 (0/0)	—
JSR	<ea>	—	3 (0/1)	—	—	3 (0/1)	4 (0/1)	3 (0/1)	—
RTE		—	—	7 (2/0)	—	—	—	—	—
RTS		—	—	5 (1/0)	—	—	—	—	—

Table 21-32. 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 22
ColdFire v1 Debug (CF1_DEBUG)

22.1 Chip-specific information about CF1_DEBUG

The built-in PST trace buffer (PSTB) feature is not implemented on FXLC95000CL. Ignore all references to this feature.

22.2 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 the following figure.

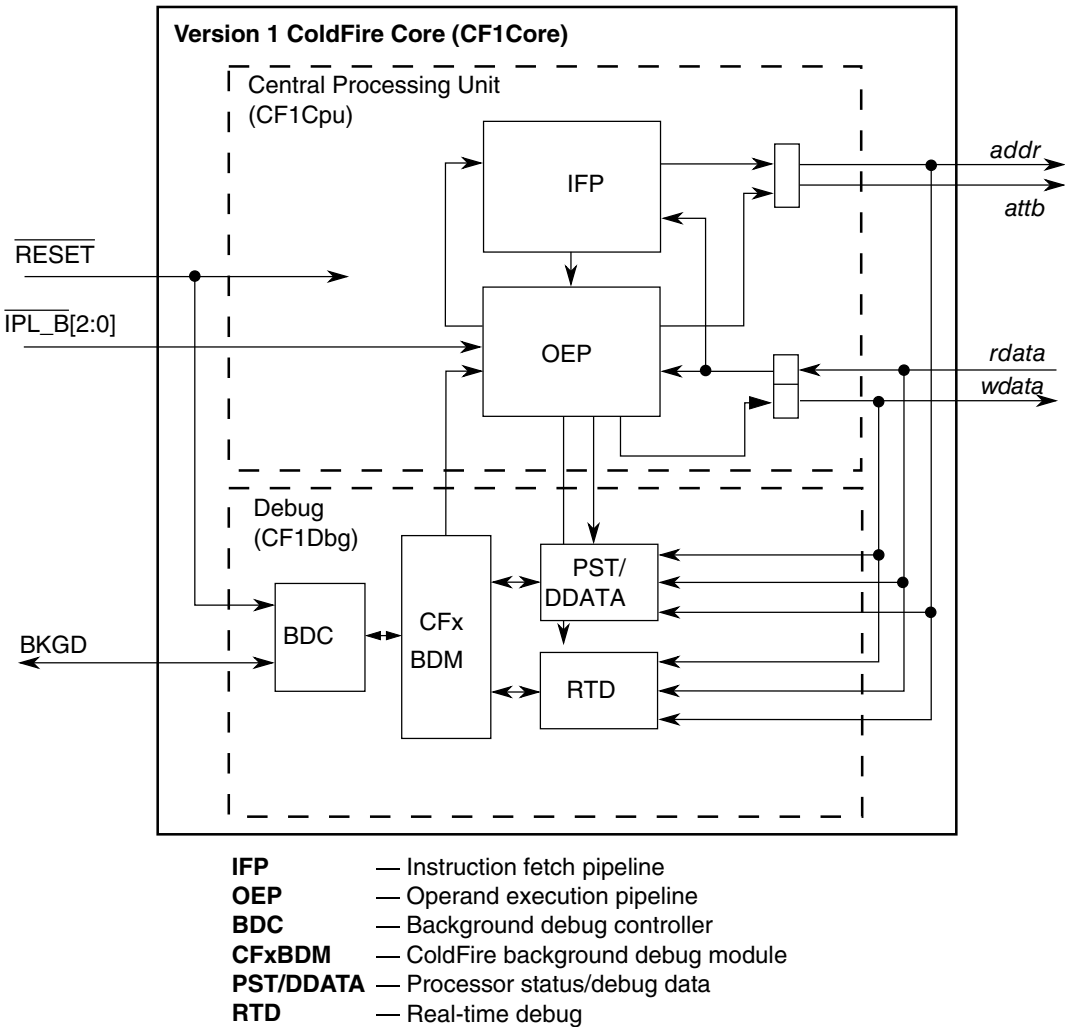


Figure 22-1. Simplified Version 1 ColdFire Core Block Diagram

22.2.1 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 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 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 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. The following table summarizes the various debug revisions.

Table 22-1. 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) $\overline{\text{BKPT}}$ 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
CF1_B +_no_PSTB	1001	0010	Standard CF1 Debug_B+ without the PST trace buffer

22.2.2 Features

The Version 1 ColdFire debug definition supports the following features:

- Classic ColdFire DEBUG_B+ functionality mapped into the single-pin BDM interface
- Real time debug support, with 6 hardware breakpoints (4 PC, 1 address pair and 1 data) that can be configured into a 1- or 2-level trigger with a programmable response (processor halt or interrupt)

- Capture of compressed processor status and debug data into on-chip trace buffer provides program (and optional slave bus data) trace capabilities
- On-chip trace buffer provides programmable start/stop recording conditions plus support for obtrusive or PC-profiling modes
- Debug resources are accessible via single-pin BDM interface or the privileged WDEBUG instruction from the core

22.2.3 Modes of Operation

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 that determine its operating state.

When operating in secure mode, as defined by a 2-bit field in the flash memory examined at reset, BDM access to debug resources is extremely restricted. It is possible to tell that the device has been secured and to clear security, which involves mass erasing the on-chip flash memory. No other debug access is allowed. Secure mode can be used in conjunction with each of the wait and stop low-power modes.

If the BDM interface is not enabled, access to the debug resources is limited in the same manner as a secure device.

If the device is not secure and the BDM interface is enabled (XCSR[ENBDM] is set), the device is operating in debug mode and additional resources are available via the BDM interface. In this mode, the status of the processor (running, stopped, or halted) determines which BDM commands may be used.

Debug mode functions are managed through the background debug controller (BDC) in the Version 1 ColdFire core. The BDC provides the means for analyzing MCU operation during software development.

BDM commands can be classified into three types as shown in the following table.

Table 22-2. BDM Command Types

Command Type	Flash Secure?	BDM?	Core Status	Command Set
Always-available	Secure or Unsecure	Enabled or Disabled	—	• Read/write access to XCSR[31–24], CSR2[31–24], CSR3[31–24]
Non-intrusive	Unsecure	Enabled	Run, Halt	• Memory access • Memory access with status

Table continues on the next page...

Table 22-2. BDM Command Types (continued)

Command Type	Flash Secure?	BDM?	Core Status	Command Set
				• Debug register access • BACKGROUND
Active background	Unsecure	Enabled	Halt	• Read or write CPU registers (also available in stop mode) • Single-step the application • Exit halt mode to return to the application program (GO)

For more information on these three BDM command classifications, see BDM Command Set Summary.

The core's halt mode is entered in a number of ways:

- The BKGD pin is low during POR
- The BKGD pin is low immediately after a BDM-initiated force reset (see CSR2[BDFR] in Configuration/Status Register 2 (CSR2), for details)
- A background debug force reset occurs (CSR2[BDFR] is set) and CSR2[BFHBR] is set
- An illegal operand reset occurs and CSR2[IOPHR] is set
- An illegal address reset occurs and CSR2[IADHR] is set
- A reset occurs and CSR2[HR] is set
- A BACKGROUND command is received through the BKGD pin. If necessary, this wakes the device from STOP/WAIT modes.
- A properly-enabled (XCSR[ENBDM] is set) HALT instruction is executed
- Encountering a BDM breakpoint and the trigger response is programmed to generate a halt
- Reaching a PSTB trace buffer full condition when operating in an obtrusive recording mode (CSR2[PSTBRM] is set to 01 or 11)

While in halt mode, the core waits for serial background commands rather than executing instructions from the application program.

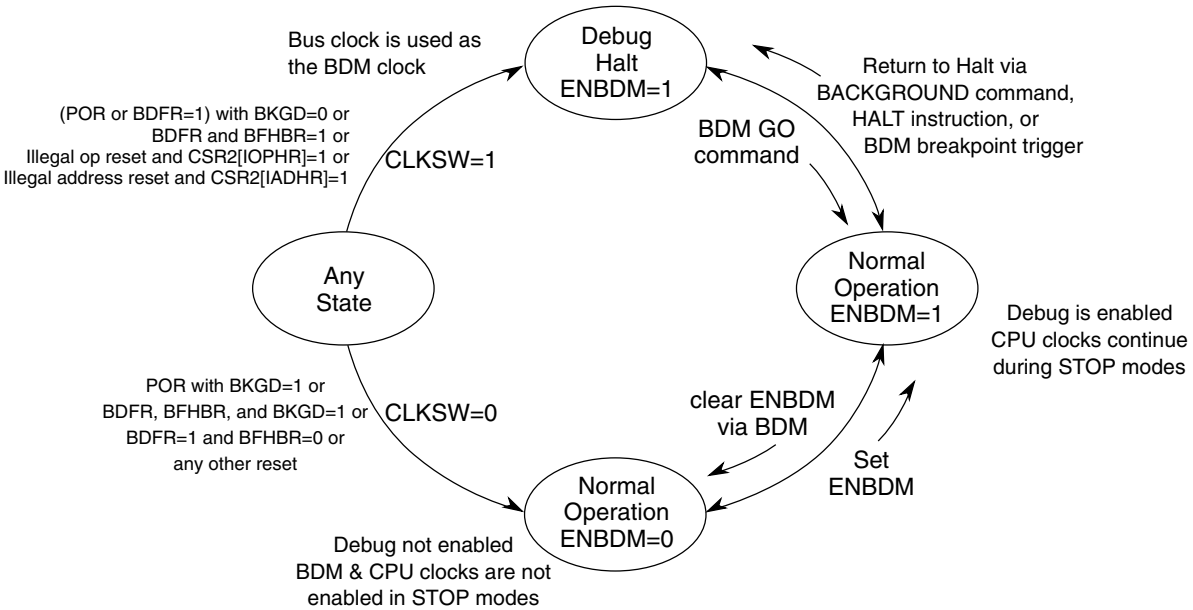


Figure 22-2. Debug Modes State Transition Diagram

The preceding figure 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 the same as the synchronous bus 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.

22.3 External Signal Descriptions

The following table describes the debug module's 1-pin external signal (BKGD). A standard 6-pin debug connector is shown in Freescale-Recommended BDM Pinout.

Table 22-3. 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.

22.4 Memory Map and Register Descriptions

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 are treated as 32-bit quantities regardless of the number of implemented bits. 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 BDM Command Set Summary. These commands contain a 5-bit field, DRc, that specifies the register, as shown in the following table.

Note

Most debug control registers can be written either by the external development system or by 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.

Table 22-4. Debug Module Memory Map

DRc[4:0]	Register	Width (bits)	Access	Reset Value
0x00	Configuration/Status Register (CSR)	32	R/W (BDM), W (CPU)	0x0090_0000
0x01	Extended Configuration/Status Register (XCSR)	32	R/W (BDM), W (CPU)	0x0000_0000
0x02	Configuration/Status Register 2 (CSR2)	32	R/W[1] (BDM), W (CPU)	See section
0x03	Configuration/Status Register 3 (CSR3)	32	R/W[1] (BDM), W (CPU)	0x0000_0000
0x05	BDM Address Attribute Register (BAAR)	32[2]	W	0x0000_0005
0x06	Address Attribute Trigger Register (AATR)	32[2]	W	0x0000_0005
0x07	Trigger Definition Register (TDR)	32	W	0x0000_0000
0x08	Program Counter Breakpoint Register 0 (PBR0)	32	W	Undefined, unaffected
0x09	Program Counter Mask Register (PBMR)	32	W	Undefined, unaffected[3]
0x0C	Address Breakpoint High Register (ABHR)	32	W	Undefined, unaffected[3]
0x0D	Address Breakpoint Low Register (ABLR)	32	W	0x0000_0000
0x0E	Data Breakpoint Register (DBR)	32	W	0x0000_0000
0x0F	Data Breakpoint Mask Register (DBMR)	32	W	0x0000_0000
0x18	Program Counter Breakpoint Register 1 (PBR1)	32	W	PBR1[0] = 0
0x1A	Program Counter Breakpoint Register 2 (PBR2)	32	W	PBR2[0] = 0
0x1B	Program Counter Breakpoint Register 3 (PBR3)	32	W	PBR3[0] = 0
—	PST Trace Buffer*n* (PSTB*n*); *n* = 0–11 (0xB)	32	R (BDM)[4]	Undefined, unaffected[3]

1. 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.
2. Each debug register is accessed as a 32-bit value; undefined fields are reserved and must be cleared.
3. The register's reset value is undefined after POR and unaffected by any other reset type. After POR, any value written to the register is retained.
4. The contents of the PST trace buffer is only read from BDM (32 bits per access) using READ_PSTB commands.

22.4.1 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: 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	VBD	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	TRO	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

Table 22-6. CSR Field Descriptions

Field	Description
31–28 BSTAT	Breakpoint status Provides read-only status (from the BDM port only) information concerning hardware breakpoints. BSTAT is cleared by a TDR write, by a CSR read when a level-2 breakpoint is triggered, or a level-1 breakpoint is triggered and the level-2 breakpoint is disabled. The PSTB value that follows the PSTB entry of 0x1B is 0x20 + (2 x BSTAT). 0000 No breakpoints enabled 0001 Waiting for level-1 breakpoint 0010 Level-1 breakpoint triggered 0101 Waiting for level-2 breakpointB 0110 Level-2 breakpoint triggered
27 FOF	Fault-on-fault Indicates a catastrophic halt occurred and forced entry into BDM. FOF is cleared by reset or when CSR is read (from the BDM port only).
26 TRG	Hardware breakpoint trigger Indicates a hardware breakpoint halted the processor core and forced entry into BDM. Reset, the debug GO command, or reading CSR (from the BDM port only) clears TRG.
25 HALT	Processor halt Indicates the processor executed a HALT and forced entry into BDM. Reset, the debug go command, or reading CSR (from the BDM port only) clears HALT.
24 BKPT	Breakpoint assert Indicates when either:

Table continues on the next page...

Table 22-6. CSR Field Descriptions (continued)

Field	Description
	• The $\overline{\text{BKPT}}$ input was asserted, • BDM BACKGROUND command received, or • The PSTB halt on full condition, CSR2[PSTBH], sets. This forces the processor into a BDM halt. Reset, the debug go command, or reading CSR (from the BDM port only) clears BKPT.
23–20 HRL	Hardware revision level Indicates, from the BDM port only, the level of debug module functionality. An emulator can use this information to identify the level of functionality supported. 0000 Revision A 0001 Revision B 0010 Revision C 0011 Revision D 1001 Revision B+ (The value used for this device) 0110 Revision D+
19	Reserved; must be cleared.
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 (default behavior). 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 Table 22-39. 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

Table continues on the next page...

Table 22-6. CSR Field Descriptions (continued)

Field	Description
	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. 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.
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 ipg_debug The core generates this output to the device, signaling it is in debug mode. 0 Do not force the assertion of ipg_debug. 1 Force the assertion of ipg_debug.
0 DDH	Disable ipg_debug 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 ipg_debug if the core is halted. 1 Negate ipg_debug due to the core being halted.

22.4.2 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).

This table summarizes the methods for accessing the XCSR.

Table 22-7. XCSR Access Summary

Reference method	Details
READ_XCSR_BYTE	Reads bits 31–24 from the BDM interface. Available in all modes.
WRITE_XCSR_BYTE	Writes bits 31–24 from the BDM interface. Available in all modes.
READ_DREG	Reads bits 31–0 from the BDM interface. Classified as a non-intrusive BDM command. Available in all modes.
WRITE_DREG	Writes bits 23–0 from the BDM interface. Classified as a non-intrusive BDM command. Available in all modes.
WDEBUG instruction	Writes bits 23–0 during execution of the core 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	CPUHALT	CPUSTOP	CSTAT			CLKSW	SEC	ENBDM	0	0	0	0	0	0	0	0
W			ESEQO				ERASE									
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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

Table 22-9. XCSR Field Descriptions

Field	Description
31 CPUHALT	CPU Halt Indicates that the CPU is in the halt state. The CPU state may be running, stopped, or halted, as indicated by the CPUHALT and CPUSTOP bits. If both CPUHALT and CPUSTOP are 0, then the CPU is running.
30 CPUSTOP	CPU Stop Indicates that the CPU is in the stop state. The CPU state may be running, stopped, or halted, as indicated by the CPUHALT and CPUSTOP bits. If both CPUHALT and CPUSTOP are 0, then the CPU is running.
29–27 (Read) CSTAT	BDM Command Status Indicates the BDM command status. 000 Command done, no errors 001 Command done, data invalid 01x Command done, illegal 1xx Command busy, overrun If an overrun is detected (CSTAT = 1xx), the following sequence is suggested to clear the source of the error: 1. Issue a SYNC command to reset the BDC channel. 2. The host issues a BDM NOP command. 3. The host checks the channel status using a READ_XCSR_BYTE command. 4. If CSTAT = 000, then you can proceed; else: a. Halt the CPU with a BDM BACKGROUND command. b. Repeat steps 1 through 3. c. If CSTAT != 000, then reset device.
29–27 (Write) ESEQC	Erase Sequence Control Use the ESEQC field for erase sequence control during flash programming. ERASE must also be 1b for this bit to have an effect. 000 User mass erase Other Reserved **NOTE:** See the detailed description of chip security for information about the algorithm for clearing security.
26 CLKSW	Clock Select Select source for serial BDC communication clock. 0 Alternate, asynchronous BDC clock 1 Synchronous bus clock The initial state of the CLKSW bit is loaded by the hardware in response to certain reset events and the state of the BKGD pin, as described in Figure 22-2.
25 (Read) SEC	Flash Security Status This bit's read value typically indicates the status of the flash security field. 0 Flash security disabled 1 Flash security enabled

Table continues on the next page...

Table 22-9. XCSR Field Descriptions (continued)

Field	Description
	If the user programs a mass erase via BDM, then SEC acts as a flash busy flag and its values have the following meanings. You can poll SEC to track the progress of the mass erase sequence. This bit is cleared immediately after the erase operation is complete, and then it returns to indicating the status of flash security. 0 Flash is not busy performing a BDM mass erase sequence 1 Flash is busy performing a BDM mass erase sequence
25 (Write) ERASE	Erase Enable Enables mass erase using the ESEQC bit. 0 Mass erase through ESEQC is disabled 1 Mass erase through ESEQC is enabled
24 ENBDM	Enable BDM 0 Disable BDM 1 Enable BDM (assuming the flash is not secure, as indicated in SEC)
23–3 Reserved	Reserved for future use by the debug module. These bits must all equal 0b.
2–1 APCSC	Automatic PC Synchronization Control If APCENB is 1b, determines the periodic interval of PC address captures. When the selected interval is reached, a SYNC_PC command is sent to the CPU. For more information on the SYNC_PC operation, see the APCENB description. The chosen frequency depends on the setting of CSR2[APCDIV16], as shown in Table 22-10.
0 APCENB	Automatic PC Synchronization Enable Enables the periodic output of the PC, which can be used for PST/DDATA trace synchronization and code profiling. As described in APCSC, when the enabled periodic timer expires, a SYNC_PC command is sent to the CPU that 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 0b, three bytes if CSR[9] is 1b). This produces a PST sequence of the PST marker indicating a 2- or 3-byte address, followed by the captured instruction address. 0 Disable automatic PC synchronization 1 Enable automatic PC synchronization

This table shows the selected PC address capture period as determined by the XCSR[APCENB], CSR2[APCDIV16], and XCSR[APCSC] fields.

Table 22-10. PC address capture period (SYNC_PC interval)

XCSR[APCENB]	CSR2[APCDIV16]	XCSR[APCSC]	SYNC_PC interval (cycles)
1	1	00	128
1	1	01	256
1	1	10	512
1	1	11	1024
1	0	00	2048

Table continues on the next page...

Table 22-10. PC address capture period (SYNC_PC interval) (continued)

XCSR[APCENB]	CSR2[APCDIV16]	XCSR[APCSC]	SYNC_PC interval (cycles)
1	0	01	4096
1	0	10	8092
1	0	11	16384

22.4.3 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).

This table summarizes the methods for accessing CSR2.

Table 22-11. CSR2 Access Summary

Reference method	Details
READ_CSR2_BYTE	Reads bits 31–24 from the BDM interface. Available in all modes.
WRITE_CSR2_BYTE	Writes bits 31–24 from the BDM interface. Available in all modes.
READ_DREG	Reads bits 31–0 from the BDM interface. Classified as a non-intrusive BDM command.
WRITE_DREG	Writes bits 23–0 from the BDM interface. Classified as a non-intrusive BDM command.
WDEBUG instruction	Writes bits 23–0 during execution of the core WDEBUG instruction. This instruction is a privileged supervisor-mode instruction.

DRc: 0x02 (CSR2)

Access: Supervisor read-only

BDM read/write

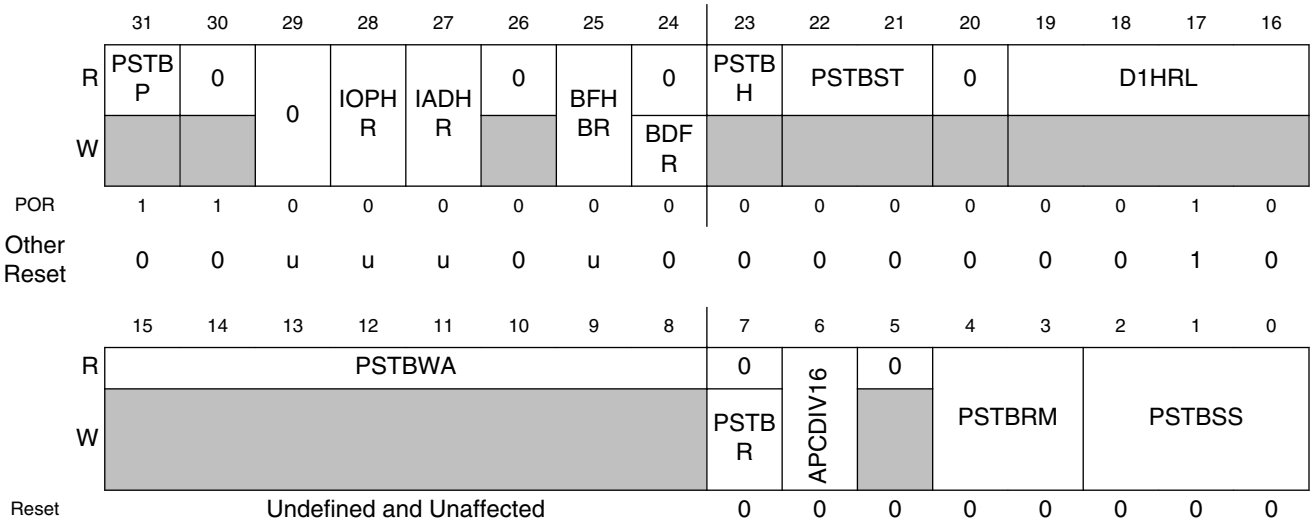


Table 22-12. CSR2 Field Descriptions

Field	Description
31 PSTBP	PST Buffer Stop Signals if a PST buffer stop condition has been reached. 0 Stop condition has not been reached 1 Stop condition has been reached
30–29	Reserved. Must write 0.
28 IOPHR	Illegal Operation Halt After Reset Specifies operation of the device after an illegal operation reset. This bit is 0b after a power-on reset and is unaffected by any other reset. **NOTE:** This bit can be changed only if XCSR[ENBDM] is 1b and the flash is not secure. 0 The device immediately enters normal operation mode. 1 The device halts (as if the BKGD pin was held low after a power-on reset).
27 IADHR	Illegal Address Halt After Reset Specifies operation of the device after an illegal address reset. This bit is 0b after a power-on reset and is unaffected by any other reset. **NOTE:** This bit can be changed only if XCSR[ENBDM] is 1b and the system is not secure. 0 The device immediately enters normal operation mode. 1 The device halts (as if the BKGD pin was held low after a power-on reset).
26	Reserved. Must write 0.
25 BFHBR	BDM Force Halt on BDM Reset Specifies operation of the device after a BDM reset (on any reset based on external BKGD logic implementation). This bit is cleared after a power-on reset and is unaffected by any other reset. **NOTE:** This bit can be changed only if XCSR[ENBDM] is 1b and the system is not secure. 0 The device immediately enters normal operation mode.

Table continues on the next page...

Table 22-12. CSR2 Field Descriptions (continued)

Field	Description
	1 The device halts (as if the BKGD pin was held low after a power-on reset).
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 1 Force a BDM reset
23 PSTBH	PST Trace Buffer Halt Indicates if the processor is halted due to the PST trace buffer being full when recording in obtrusive mode. 0 Not halted 1 Halted
22–21 PSTBST	PST Trace Buffer State Indicates the current state of PST trace buffer recording. 00 Disabled 01 Enabled and waiting for the start condition 10 Enabled, recording, and waiting for the stop condition 11 Enabled, completed recording after the stop condition was reached
20	Reserved. Must write 0.
19–16 D1HRL	Debug 1-pin Hardware Revision Level Indicates the hardware revision level of the 1-pin debug module implemented in the core. For this device, this field is 2h.
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 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 most-significant bit of this field can be used to determine if the entire PST trace buffer has been loaded with valid data. 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. **NOTE:** Because this device contains a 64-entry trace buffer, PSTBWA[6] is always zero. Bit 7 PSTB valid data locations (oldest to newest) 0 0, 1, ..., PSTBWA – 1 1 PSTBWA, PSTBWA + 1, ..., 0, 1, ..., PSTBWA – 1
7 PSTBR	PST Trace Buffer Reset Generates a reset of the PST trace buffer logic, which clears PSTBWA and PSTBST. The same resources are reset when a disabled trace buffer becomes enabled and upon the receipt of a BDM GO command when operating in obtrusive trace mode. These reset events also clear any accumulation of PSTs. This bit always reads as 0b. 0 No reset 1 Force a PST trace buffer reset

Table continues on the next page...

Table 22-12. CSR2 Field Descriptions (continued)

Field	Description
6 APCDIV16	Automatic PC Synchronization Divide Cycle Counts by 16 Divides the cycle counts for automatic SYNC_PC command insertion by 16. See the XCSR[APCSC] and XCSR[APCENB] fields.
5	Reserved. Must write 0.
4–3 PSTBRM	PST Trace Buffer Recording Mode Specifies the trace buffer recording mode. The start and stop recording conditions are defined by the PSTBSS field. The terms obtrusive and non-obtrusive are defined as: • Non-obtrusive—The core is not halted. The PST trace buffer is overwritten unless a PSTB start/stop combination results in less than or equal to 64 PSTB captures. • Obtrusive—The core is halted when the PSTB trace buffer reaches its full level (full before overwriting). The PSTB trace buffer contents are available by the BDM PSTB_READ commands. The PSTB trace buffer write address resets and the CPU resumes upon a BDM GO command. 00 Non-obtrusive, normal recording mode 01 Obtrusive, normal recording 10 Non-obtrusive, PC profile recording. Automatic PC synchronization must be enabled (see XCSR[APCSC, APCENB], CSR2[APCDIV16], and CSR[BTB]). 11 Obtrusive, PC profile recording. Automatic PC synchronization must be enabled (see XCSR[APCSC, APCENB], CSR2[APCDIV16], and CSR[BTB]).
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. See Table 22-13.

This table shows the start and stop recording conditions as specified by the PSTBSS field.

Table 22-13. PST trace buffer start and stop recording conditions (CSR2[PSTBSS])

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

22.4.4 Configuration/Status Register 3 (CSR3)

The CSR3 contains the BDM flash clock divider (BFCDIV) value in a format similar to HCS08 devices.

This table summarizes the methods for accessing CSR3.

Table 22-14. CSR3 Reference Summary

Method	Reference Details
READ_CSR3_BYTE	Reads bits CSR3[31–24] from the BDM interface. Available in all modes.
WRITE_CSR3_BYTE	Writes bits CSR3[31–24] from the BDM interface. Available in all modes.
READ_DREG	Reads bits CSR3[31–0] from the BDM interface. Classified as a non-intrusive BDM command.
WRITE_DREG	Writes bits CSR3[23–0] from the BDM interface. Classified as a non-intrusive BDM command.
WDEBUG instruction	No operation while the core executes 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	BFCDIV8	BFCDIV						0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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

Table 22-15. CSR3 Field Descriptions

Field	Description
31	Reserved; must be cleared.
30 BFCDIV8	BDM flash clock divide 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. Specifies the frequency of the internal flash clock during 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.

Table continues on the next page...

Table 22-15. CSR3 Field Descriptions (continued)

Field	Description
	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 ms. 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.

22.4.5 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 has no effect.

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
R																
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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

Table 22-17. BAAR Field Descriptions

Field	Description	
31–8	Reserved for future use by the debug module; must be cleared.	
7 R	Read/Write 0 1	 Write. Read.
6–5 SZ	Size 00	 Longword

Table continues on the next page...

Table 22-17. BAAR Field Descriptions (continued)

Field	Description
	01 Byte 10 Word 11 Reserved
4–3 TT	Transfer type See the TT definition in the AATR description.
2–0 TM	Transfer modifier See the TM definition in the AATR description.

22.4.6 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
R																
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

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W	RM	SZM		TTM		TMM			R	SZ		TT		TM		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1

Table 22-19. 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.

Table continues on the next page...

Table 22-19. AATR Field Descriptions (continued)

Field	Description
10–8 TMM	Transfer modifier mask Masks the corresponding TM bit in address comparisons.
7 R	Read/Write R is compared with the R/$\overline{W}$ 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

22.4.7 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 (write 0 to L2EBL and 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

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
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

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

Table 22-21. 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 Breakpoints Global enable for the level 2 breakpoint triggers. 0 Disable all level 2 breakpoints 1 Enable all level 2 breakpoints
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. Bit Description 28 Data longword. Entire processor's local data bus 27 Lower data word 26 Upper data word

Table continues on the next page...

Table 22-21. TDR Field Descriptions (continued)

Field	Description
	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
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 Do not invert 1 Invert
20–18 L2EA	Enable Level 2 Address Breakpoint Setting an L2EA bit enables the corresponding address breakpoint. Clearing all three bits disables the breakpoint. Bit Description 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 1 Enable
16 L2PCI	Level 2 PC Breakpoint Invert 0 Do not invert 1 Invert
15 L2T	Level 2 Trigger Determines the logic operation for the trigger between the PC_condition and the (Address_range and Data_condition) condition where the inclusion of a Data_condition is optional. The debug architecture supports the creation of single- or double-level triggers. 0 Trigger = PC_condition && (Address_range && Data_condition) 1 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) condition where the inclusion of a Data_condition is optional. The debug architecture supports the creation of single- or double-level triggers. 0 Trigger = PC_condition && (Address_range && Data_condition) 1 Trigger = PC_condition || (Address_range && Data_condition)
13 L1EBL	Enable Level 1 Breakpoints Global enable for the level 1 breakpoint triggers. 0 Disable all level 1 breakpoints 1 Enable all level 1 breakpoints
12–6	Enable Level 1 Data Breakpoint

Table continues on the next page...

Table 22-21. TDR Field Descriptions (continued)

Field	Description
L1ED	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 ED bits disables data breakpoints. 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
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 — Do not invert 1 — Invert
4–2 L1EA	Enable Level 1 Address Breakpoint Setting an L1EA bit enables the corresponding address breakpoint. Clearing all three bits disables the breakpoint. Bit — Description 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.
1 L1EPC	Enable Level 1 PC Breakpoint 0 — Disable 1 — Enable
0 L1PCI	Level 1 PC Breakpoint Invert 0 — Do not invert 1 — Invert

22.4.8 Program Counter Breakpoint/Mask Registers (PBR0–3, PBMR)

The PBRn 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 BDM Command Set Descriptions.

NOTE

Version 1 ColdFire core devices implement a 24-bit, 16 MB address map. When programming these registers with a 32-bit address, the upper byte must be zero-filled.

DRc: 0x08 (PBR0) Access: Supervisor write-only

BDM write-only

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W	Address															
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

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

Table 22-23. 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.

DRc: 0x18 (PBR1)
0x1A (PBR2)
0x1C (PBR3) Access: Supervisor write-only

BDM write-only

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W	Address															
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

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

Table 22-25. 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.

The following display 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.

DRc: 0x09 Access: Supervisor write-only, BDM write-only

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W	Mask															
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

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

Table 22-27. PBMR Field Descriptions

Field	Description
31–1 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.

22.4.9 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 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.

DRc: 0x0C (ABHR) Access: Supervisor write-only

0x0D (ABLR) BDM write-only

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W	Address															
ABHR Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
ABLR Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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

Table 22-29. ABLR Field Descriptions

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 22-30. ABHR Field Descriptions

Field	Description
31–0 Address	High address Holds the 32-bit address marking the upper bound of the address breakpoint range.

22.4.10 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.

DRc: 0x0E (DBR) Access: Supervisor write-only

0x0F (DBMR) BDM write-only

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W	Data (DBR); Mask (DBMR)															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W	Data (DBR); Mask (DBMR)															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 22-32. 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 22-33. 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. This table shows the relationships between processor address, access size, and location within the 32-bit data bus.

Table 22-34. Access Size and Operand Data Location

Address[1–0]	Access Size	Operand Location
00	Byte	D[31–24]

Table continues on the next page...

Table 22-34. Access Size and Operand Data Location (continued)

Address[1–0]	Access Size	Operand Location
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]

22.4.11 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.

22.4.12 PST Buffer (PSTB)

The PST trace buffer contains 64 six-bit entries, packed consecutively into 12 longword locations. The following table illustrates 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.

Table 22-35. PST Trace Buffer Entries and Locations

Core register number (CRN)	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x10	TB #00						TB #01						TB #02						TB #03						TB #04						05[5:4]	
0x11	TB #05[3:0]				TB #06				TB #07				TB #08						TB #09						TB #10[5:2]							
0x12	10[1:0]		TB #11						TB #12						TB #13						TB #14						TB #15					
0x13	TB #16						TB #17						TB #18						TB #19						TB #20						21[5:4]	
0x14	TB #21[3:0]				TB #22				TB #23				TB #24						TB #25						TB #26[5:2]							
0x15	26[1:0]		TB #27						TB #28						TB #29						TB #30						TB #31					
0x16	TB #32						TB #33						TB #34						TB #35						TB #36						37[5:4]	
0x17	TB #37[3:0]				TB #38				TB #39				TB #40						TB #41						TB #42[5:2]							
0x18	42[1:0]		TB #43						TB #44						TB #45						TB #46						TB #47					
0x19	TB #48						TB #49						TB #50						TB #51						TB #52						53[5:4]	

Table continues on the next page...

Table 22-35. PST Trace Buffer Entries and Locations (continued)

Core register number (CRN)	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x1A	TB #53[3:0]				TB #54						TB #55						TB #56						TB #57						TB #58[5:2]			
0x1B	58[1:0]		TB #59						TB #60						TB #61						TB #62						TB #63					

22.5 Functional Description

The following sections describe functional details of the debug module.

22.5.1 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 22-1, 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 othernon-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.

22.5.1.1 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 22-36. 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.

Table continues on the next page...

Table 22-36. CPU Halt Sources (continued)

Halt Source	Halt Timing	Description		
		CPUCR[ARD] = 1	Immediately enters halt.	
		CPUCR[ARD] = 0	Reset event is initiated.	
Hardware breakpoint trigger	Pending	Halt is made pending in the processor. The processor samples for pending halt and interrupt conditions once per instruction. When a pending condition is asserted, the processor halts execution at the next sample point.		
HALT instruction	Immediate	BDM disabled	CPUCR[IRD] = 0	A reset is initiated since attempted execution of an illegal instruction
			CPUCR[IRD] = 1	An illegal instruction exception is generated.
		BDM enabled, supervisor mode	Processor immediately halts execution at the next instruction sample point. The processor can be restarted by a BDM GO command. Execution continues at the instruction after HALT.	
		BDM enabled, user mode	CSR[UHE] = 0 CPUCR[IRD] = 0	A reset event is initiated, because a privileged instruction was attempted in user mode.
			CSR[UHE] = 0 CPUCR[IRD] = 1	A privilege violation exception is generated.
			CSR[UHE] = 1	Processor immediately halts execution at the next instruction sample point. The processor can be restarted by a BDM GO command. Execution continues at the instruction after HALT.
BACKGROUND command	Pending	BDM disabled or flash secure	Illegal command response and BACKGROUND command is ignored.	
		BDM enabled and flash unsecure	Processor is running	Halt is made pending in the processor. The processor samples for pending halt and interrupt conditions once per instruction. When a pending condition is asserted, the processor halts execution at the next sample point.
			Processor is stopped	Processing of the BACKGROUND command is treated in a special manner. The processor exits the stopped mode and enters the halted state, at which point all BDM commands may be exercised. When restarted, the processor continues by executing the next sequential instruction (the instruction following STOP).
PSTB full condition	Pending	PSTB	PSTB obtrusive recording mode pends halt in the processor if the trace buffer reaches its full threshold (full is defined as before the buffer is overwritten). When a pending condition is asserted, the processor halts at the next sample point.	
BKGD held low for ≥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.	

Table continues on the next page...

Table 22-36. CPU Halt Sources (continued)

Halt Source	Halt Timing	Description	
		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: • Issue a BDM reset setting CSR2[BDFR] with CSR2[BDHBR] cleared and the BKGD pin held high to reset into normal operating mode • Erase the flash to unsecure the memory and then proceed with debug • Power cycle the device with the BKGD pin held high to reset into the normal operating mode

The processor's run/stop/halt status is always accessible in XCSR[CPUHALT,CPUSTOP]. Additionally, CSR[27–24] indicate the halt source, showing the highest priority source for multiple halt conditions. This field is cleared by a read of the CSR. A processor halt due to the PSTB full condition as indicated by CSR2[PSTH] is also reflected in CSR[BKPT]. The debug GO command clears CSR[26–24] and CSR2[PSTBH].

22.5.1.2 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 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 BDM Communication Details, for more details.

When no debugger pod is connected to the standard 6-pin BDM interface connector (Freescale-Recommended BDM Pinout), the internal pullup on BKGD chooses normal operating mode. When a development system is connected, it can pull BKGD and $\overline{\text{RESET}}$ low, release $\overline{\text{RESET}}$ to select active background (halt) mode rather than normal operating mode, and then release BKGD. It is not necessary to reset the target MCU to communicate with it through the background debug interface. There is also a mechanism to generate a reset event in response to setting CSR2[BDFR].

22.5.1.3 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].

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_BYTE 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.

The following figure 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.

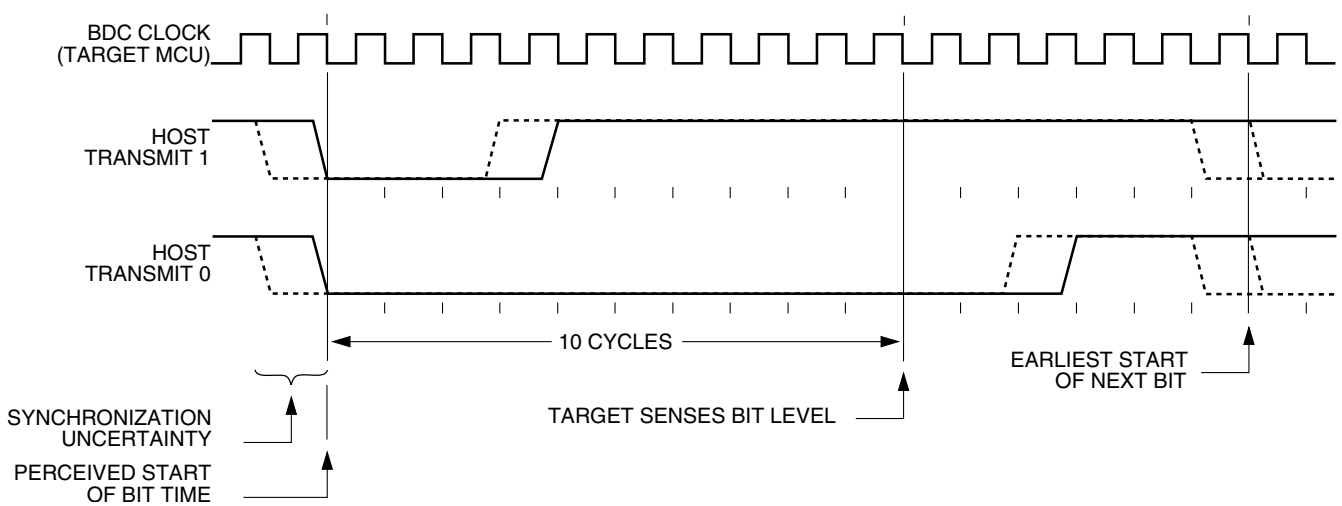


Figure 22-3. BDC Host-to-Target Serial Bit Timing

The following figure 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.

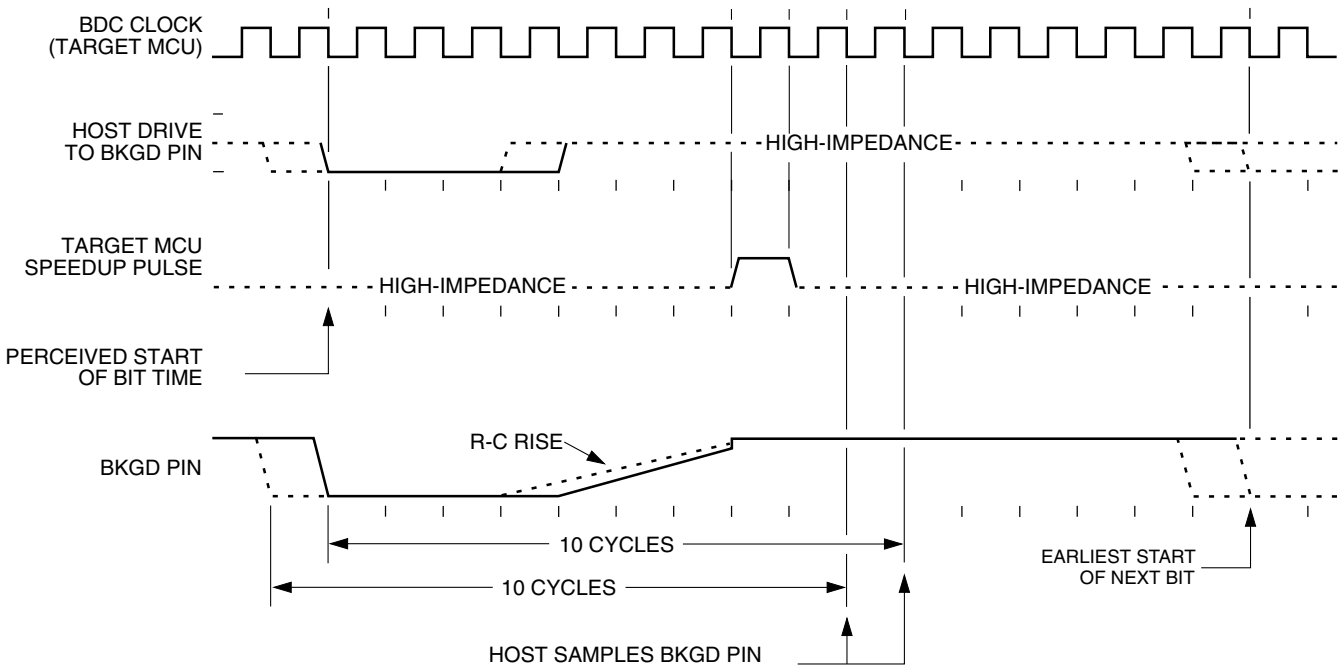


Figure 22-4. BDC Target-to-Host Serial Bit Timing (Logic 1)

The following figure 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.

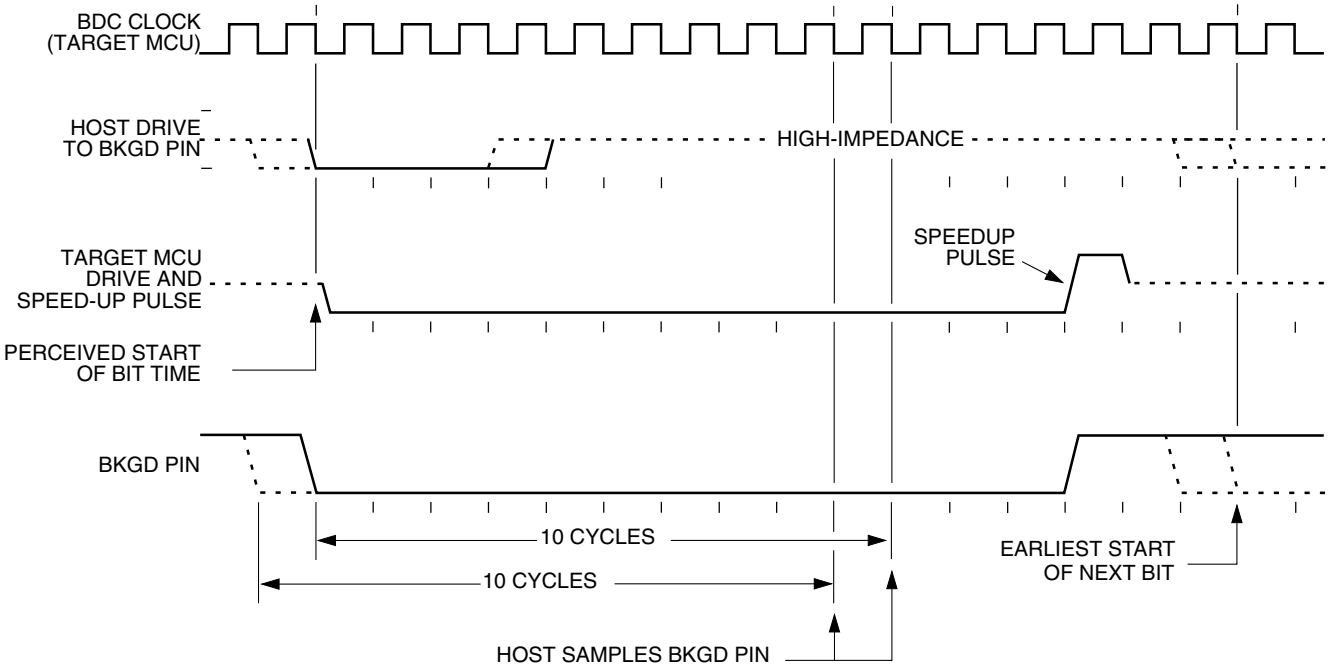


Figure 22-5. BDM Target-to-Host Serial Bit Timing (Logic 0)

22.5.1.4 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 the following figure.

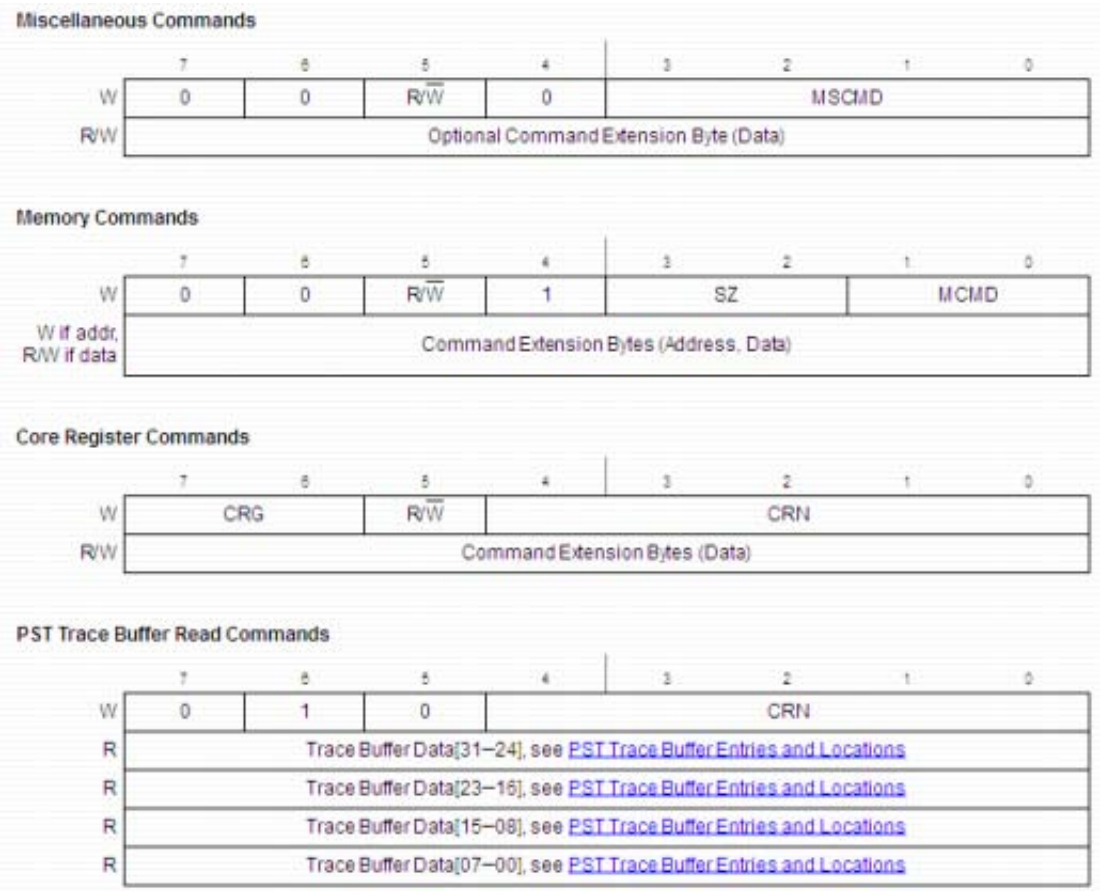


Figure 22-6. BDM Command Encodings

Table 22-37. BDM Command Field Descriptions

Field	Description
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

Table continues on the next page...

Table 22-37. BDM Command Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td></td><td>01 16-bit word
10 32-bit long</td></tr>
<tr><td>1–0
MCMD</td><td>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</td></tr>
<tr><td>7–6
CRG</td><td>Core register group. Defines the core register group to be referenced.
01 CPU's general-purpose registers (An, Dn) or PST trace buffer
10 Debug's control registers
11 CPU's control registers (PC, SR, VBR, CPUCR,...)</td></tr>
<tr><td>4–0
CRN</td><td>Core register number. Defines the specific core register (its number) to be referenced. All other CRN values are reserved.
<table>
<tr><th>CRG</th><th>CRN</th><th>Register</th></tr>
<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 colspan="2">DRc[4:0] as described in CSR</td></tr>
<tr><td rowspan="8">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>0x04</td><td>MACSR</td></tr>
<tr><td>0x05</td><td>MASK</td></tr>
<tr><td>0x06</td><td>ACC</td></tr>
<tr><td>0x0E</td><td>SR</td></tr>
<tr><td>0x0F</td><td>PC</td></tr>
</table></td></tr>
</table>

22.5.1.5 BDM Command Set Summary

The following table summarizes the BDM command set. Subsequent paragraphs contain detailed descriptions of each command. The nomenclature below is used in the following table 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 22-38. BDM Command Summary

Command Mnemonic	Command Classification	ACK if Enb? [1]	Command Structure	Description
SYNC	Always Available	N/A	N/A[2]	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.
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.

Table continues on the next page...

Table 22-38. BDM Command Summary (continued)

Command Mnemonic	Command Classification	ACK if Enb? [1]	Command Structure	Description
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[3]
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
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

Table continues on the next page...

Table 22-38. BDM Command Summary (continued)

Command Mnemonic	Command Classification	ACK if Enb? [1]	Command Structure	Description
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	0x0D/wd8	Write the most significant byte of the debug module's XCSR
WRITE_CSR2_BYTE	Always Available	No	0x0E/wd8	Write the most significant byte of the debug module's CSR2
WRITE_CSR3_BYTE	Always Available	No	0x0F/wd8	Write the most significant byte of the debug module's CSR3

1. This column identifies if the command generates an ACK pulse if operating with acknowledge mode enabled. See ACK_ENABLE , for addition information.
2. The SYNC command is a special operation which does not have a command code.
3. If a GO command is received while the processor is not halted, it performs no operation.

22.5.1.5.1 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.

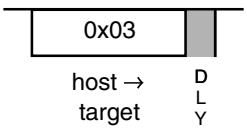
Upon detecting the sync request from the host (which is a much longer low time than would ever occur during normal BDC communications), the target:

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.

22.5.1.5.2 ACK_DISABLE

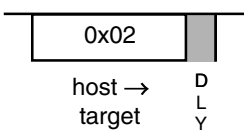
Disable host/target handshake protocol **Always Available**



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.

22.5.1.5.3 ACK_ENABLE

Enable host/target handshake protocol **Always Available**



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 Serial Interface Hardware Handshake Protocol and Hardware Handshake Abort Procedure.

22.5.1.5.4 BACKGROUND

Enter active background mode (if enabled)	Non-intrusive

Provided XCSR[ENBDM] is set (BDM enabled), the BACKGROUND command causes the target MCU to enter active background (halt) mode as soon as the current CPU instruction finishes. If ENBDM is cleared (its default value), the BACKGROUND command is ignored.

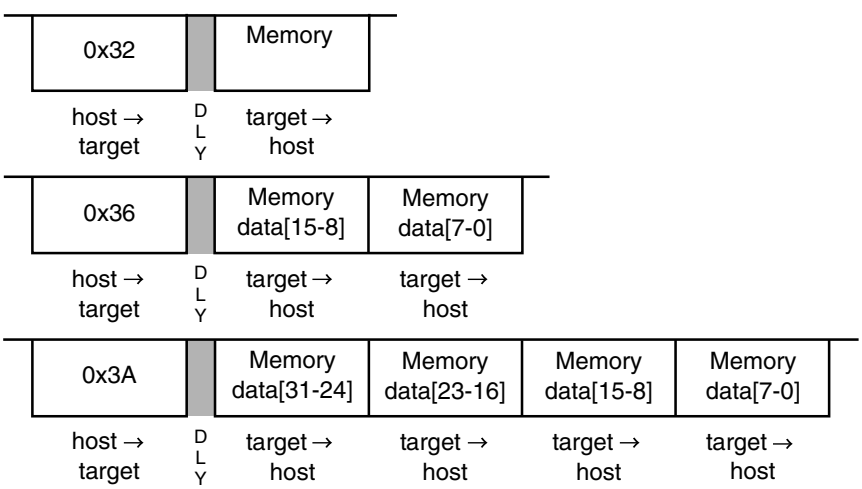
A delay of 32 BDC clock cycles is required after the BACKGROUND command to allow the target MCU to finish its current CPU instruction and enter active background mode before a new BDC command can be accepted.

After the target MCU is reset into a normal operating mode, the host debugger must send a WRITE_XCSR_BYTE command to set ENBDM before attempting to send the BACKGROUND command the first time. Normally, the development host sets ENBDM once at the beginning of a debug session or after a target system reset, and then it leaves the ENBDM bit set during debugging operations. During debugging, the host uses GO commands to move from active background mode to normal user program execution and uses BACKGROUND commands or breakpoints to return to active background mode.

22.5.1.5.5 DUMP_MEM.sz, DUMP_MEM.sz_WS

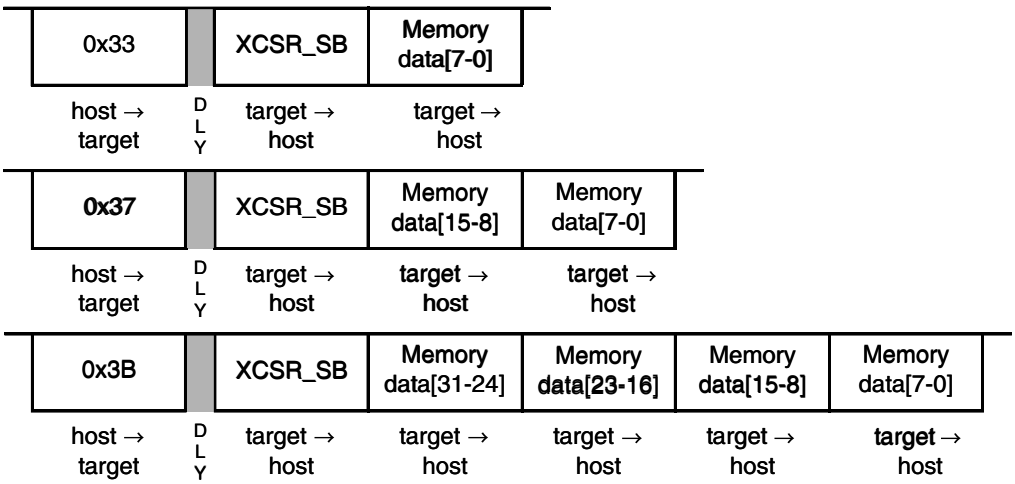
DUMP_MEM.sz

Read memory specified by debug address register, then increment address	Non-intrusive



DUMP_MEM.sz_WS

Read memory specified by debug address register with status, then increment address **Non-intrusive**



DUMP_MEM{_WS} is used with the READ_MEM{_WS} command to access large blocks of memory. An initial READ_MEM{_WS} is executed to set-up the starting address of the block and to retrieve the first result. If an initial READ_MEM{_WS} is not executed before the first DUMP_MEM{_WS}, an illegal command response is returned. The DUMP_MEM{_WS} command retrieves subsequent operands. The initial address is incremented by the operand size (1, 2, or 4) and saved in a temporary register. Subsequent DUMP_MEM{_WS} commands use this address, perform the memory read, increment it by the current operand size, and store the updated address in the temporary register. If the with-status option is specified, the core status byte (XCSR_SB) contained in XCSR[31–24] is returned before the read data. XCSR_SB reflects the state after the memory read was performed.

Note

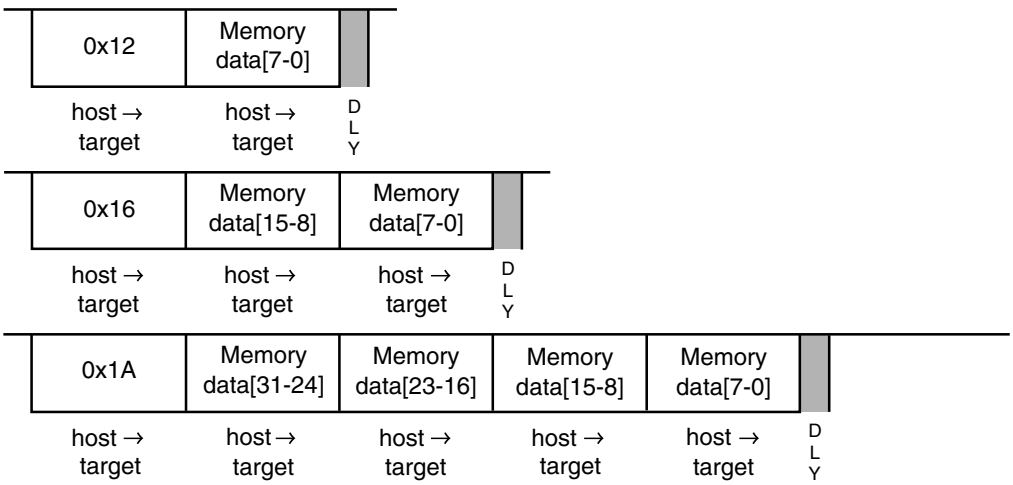
DUMP_MEM{_WS} does not check for a valid address; it is a valid command only when preceded by NOP, READ_MEM{_WS}, or another DUMP_MEM{_WS} command. Otherwise, an illegal command response is returned. NOP can be used for inter-command padding without corrupting the address pointer.

The size field (sz) is examined each time a DUMP_MEM{_WS} command is processed, allowing the operand size to be dynamically altered. The examples show the DUMP_MEM.B{_WS}, DUMP_MEM.W{_WS} and DUMP_MEM.L{_WS} commands.

22.5.1.5.6 FILL_MEM.sz, FILL_MEM.sz_WS

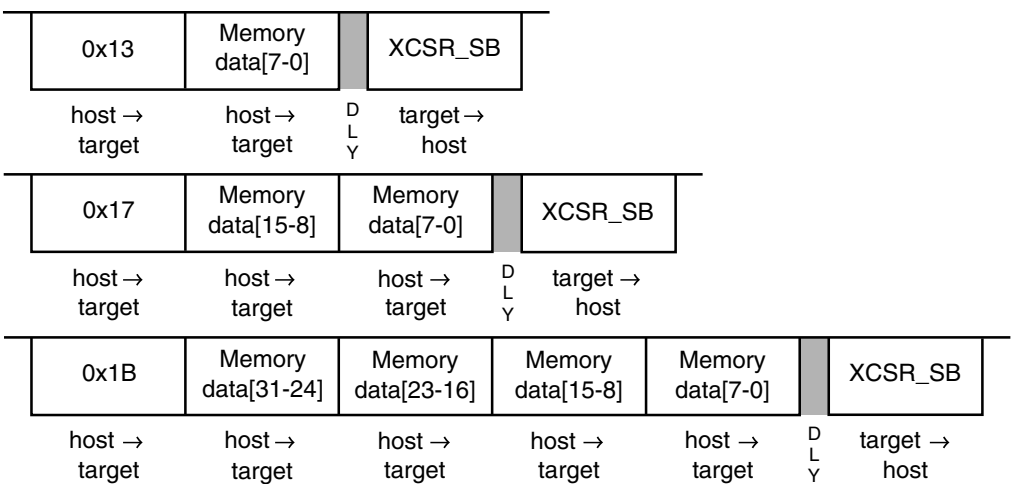
FILL_MEM.sz

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 FILL_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.

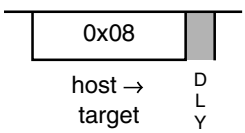
Note

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.

22.5.1.5.7 GO

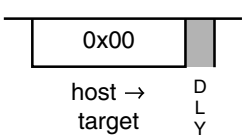
Go **Non-intrusive**



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.

22.5.1.5.8 NOP

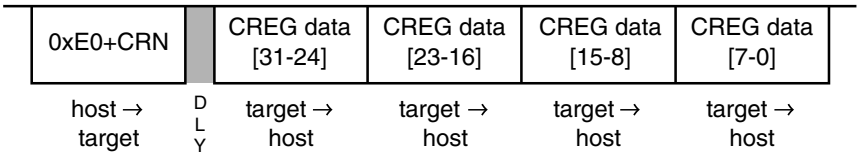
No operation **Non-intrusive**



NOP performs no operation and may be used as a null command where required.

22.5.1.5.9 READ_CREG

Read CPU control register **Active Background**

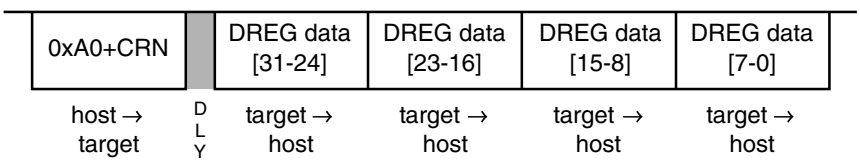


If the processor is halted, this command reads the selected control register and returns the 32-bit result. This register grouping includes the PC, SR, CPUCR, MACSR, MASK, ACC, VBR, and OTHER_A7. Accesses to processor control registers are always 32-bits wide, regardless of implemented register width. The register is addressed through the core register number (CRN). See Table 22-37 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.

22.5.1.5.10 READ_DREG

Read debug control register **Non-intrusive**

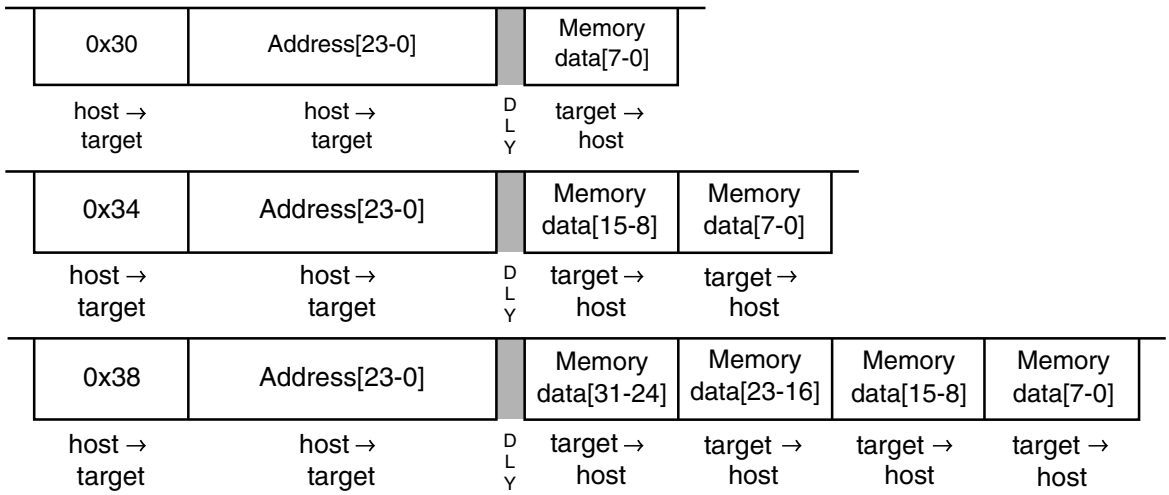


This command reads the selected debug control register and returns the 32-bit result. This register grouping includes the CSR, XCSR, CSR2, and CSR3. Accesses to debug control registers are always 32-bits wide, regardless of implemented register width. The register is addressed through the core register number (CRN). See Table 22-4 for CRN details.

22.5.1.5.11 READ_MEM.sz, READ_MEM.sz_WS

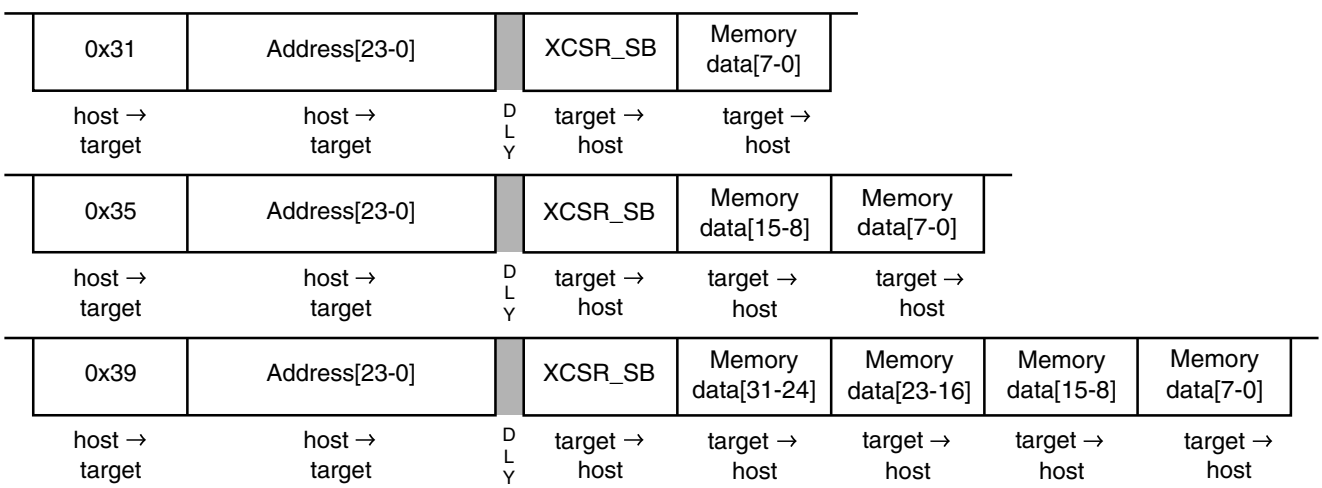
READ_MEM.sz

Read memory at the specified address **Non-intrusive**



READ_MEM.sz_WS

Read memory at the specified address with status **Non-intrusive**

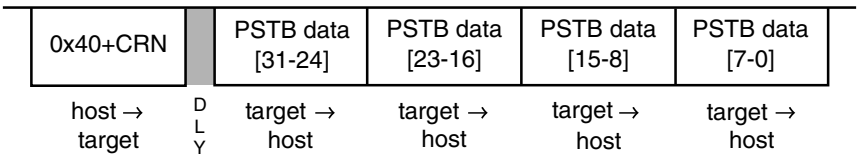


Read data at the specified memory address. The reference address is transmitted as three 8-bit packets (msb to lsb) immediately after the command packet. The access attributes are defined by BAAR[TT,TM]. The hardware forces low-order address bits to zeros for word and longword accesses to ensure these 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.

22.5.1.5.12 READ_PSTB

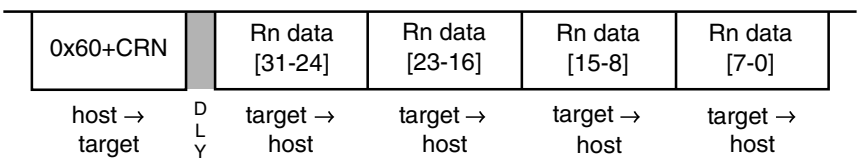
Read PST trace buffer at the specified address **Non-intrusive**



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 Table 22-35 for an illustration of how the buffer entries are packed.

22.5.1.5.13 READ_Rn

Read general-purpose CPU register **Active Background**

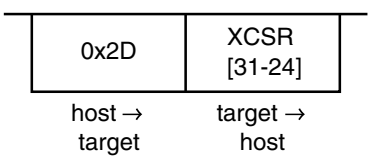


If the processor is halted, this command reads the selected CPU general-purpose register (An, Dn) and returns the 32-bit result. See Table 22-37 for the CRN details when CRG is 01.

If the processor is not halted, this command is rejected as an illegal operation and no operation is performed.

22.5.1.5.14 READ_XCSR_BYTE

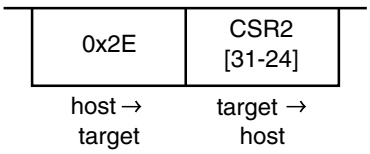
Read XCSR Status Byte **Always Available**



Read the special status byte of XCSR (XCSR[31–24]). This command can be executed in any mode.

22.5.1.5.15 READ_CSR2_BYTE

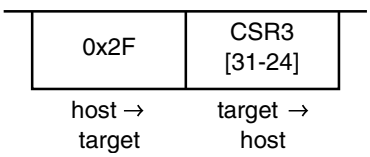
Read CSR2 Status Byte **Always Available**



Read the most significant byte of CSR2 (CSR2[31–24]). This command can be executed in any mode.

22.5.1.5.16 READ_CSR3_BYTE

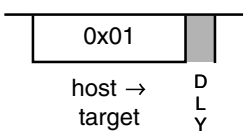
Read CSR2 Status Byte **Always Available**



Read the most significant byte of the CSR3 (CSR3[31–24]). This command can be executed in any mode.

22.5.1.5.17 SYNC_PC

Synchronize PC to PST/DDATA Signals **Non-intrusive**



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:

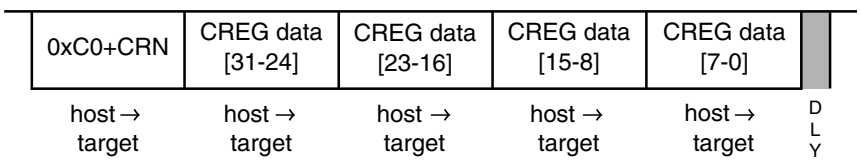
1. Debug signals a SYNC_PC command is pending.
2. CPU completes the current instruction.

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 intrusive to the real-time operation of an application than a BACKGROUND/read-PC/GO command sequence.

22.5.1.5.18 WRITE_CREG

Write CPU control register **Active Background**

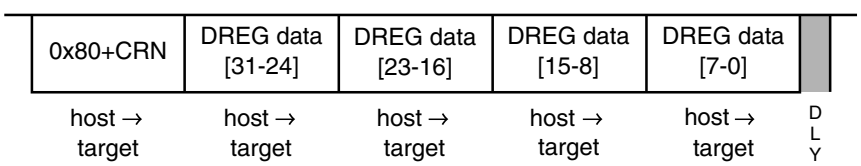


If the processor is halted, this command writes the 32-bit operand to the selected control register. This register grouping includes the PC, SR, CPUCR, MACSR, MASK, ACC, VBR, and OTHER_A7. Accesses to processor control registers are always 32-bits wide, regardless of implemented register width. The register is addressed through the core register number (CRN). See Table 22-37 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.

22.5.1.5.19 WRITE_DREG

Write debug control register **Non-intrusive**



This command writes the 32-bit operand to the selected debug control register. This grouping includes all the debug control registers ({X}CSR*n*, BAAR, AATR, TDR, PBR*n*, PBMR, AB*x*R, 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).

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.

22.5.1.5.20 WRITE_MEM.sz, WRITE_MEM.sz_WS

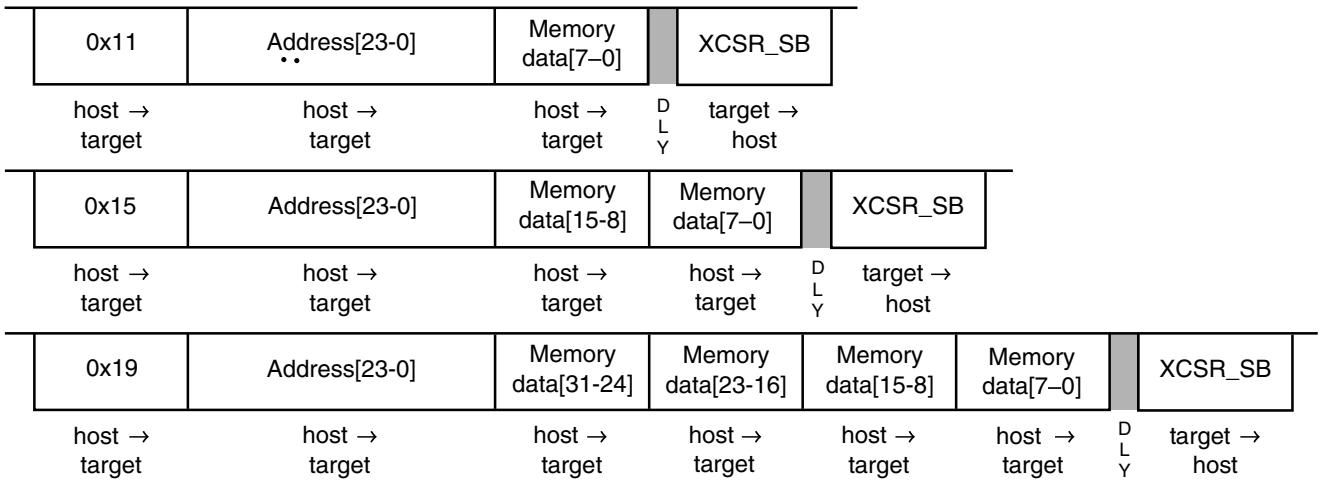
WRITE_MEM.sz

Write memory at the specified address **Non-intrusive**

0x10	Address[23-0]	Memory data[7–0]				
host → target	host → target	host → target	DLY			
0x14	Address[23-0]	Memory data[15-8]	Memory data[7–0]			
host → target	host → target	host → target	host → target	DLY		
0x18	Address[23-0]	Memory data[31-24]	Memory data[23-16]	Memory data[15-8]	Memory data[7–0]	
host → target	host → target	host → target	host → target	host → target	host → target	DLY

WRITE_MEM.sz_WS

Write memory at the specified address with status **Non-intrusive**

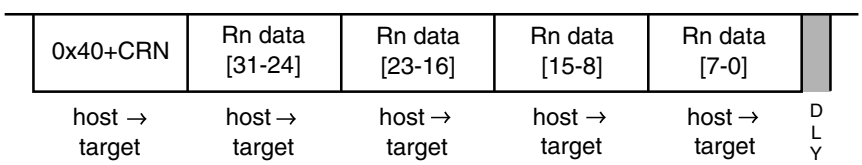


Write data at the specified memory address. The reference address is transmitted as three 8-bit packets (msb to lsb) immediately after the command packet. The access attributes are defined by BAAR[TT,TM]. The hardware forces low-order address bits to zeros for word and longword accesses to ensure these accesses are on 0-modulo-size alignments. If the with-status option is specified, the core status byte (XCSR_SB) contained in XCSR[31–24] is returned after the read data. XCSR_SB reflects the state after the memory write was performed.

The examples show the WRITE_MEM.B{_WS}, WRITE_MEM.W{_WS}, and WRITE_MEM.L{_WS} commands.

22.5.1.5.21 WRITE_Rn

Write general-purpose CPU register **Active Background**

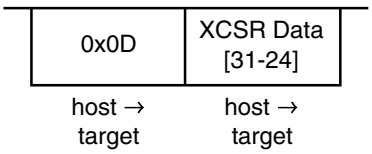


If the processor is halted, this command writes the 32-bit operand to the selected CPU general-purpose register (An, Dn). See Table 22-37 for the CRN details when CRG is 01.

If the processor is not halted, this command is rejected as an illegal operation and no operation is performed.

22.5.1.5.22 WRITE_XCSR_BYTE

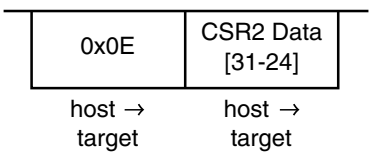
Write XCSR Status Byte **Always Available**



Write the special status byte of XCSR (XCSR[31–24]). This command can be executed in any mode.

22.5.1.5.23 WRITE_CSR2_BYTE

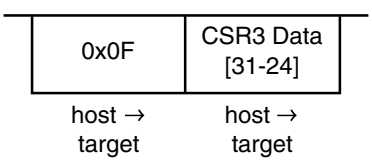
Write CSR2 Status Byte **Always Available**



Write the most significant byte of CSR2 (CSR2[31–24]). This command can be executed in any mode.

22.5.1.5.24 WRITE_CSR3_BYTE

Write CSR3 Status Byte **Always Available**



Write the most significant byte of CSR3 (CSR3[31–24]). This command can be executed in any mode.

22.5.1.5.25 BDM Accesses of the MAC Registers

The presence of rounding logic in the output datapath of the MAC requires special care for BDM-initiated reads and writes of its programming model. In particular, any result rounding modes must be disabled during the read/write process so the exact bit-wise MAC register contents are accessed.

For example, a BDM read of the accumulator (ACC) must be preceded by two commands accessing the MAC status register, as shown in the following sequence:

```
BdmReadACC (
        rcreg    macsr;             // read current macsr contents and save
        wcreg    #0,macsr;          // disable all rounding modes
        rcreg    ACC;               // read the desired accumulator
        wcreg    #saved_data,macsr; // restore the original macsr
)
```

Likewise, to write an accumulator register, the following BDM sequence is needed:

```
BdmWriteACC (
        rcreg     macsr;             // read current macsr contents and save
        wcreg     #0,macsr;          // disable all rounding modes
        wcreg     #data,ACC;         // write the desired accumulator
        wcreg     #saved_data,macsr; // restore the original macsr
```

For more information on saving and restoring the complete MAC programming model, see <<create reference to "Saving and Restoring the EMAC Programming Model" section>>.

22.5.1.6 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 22-7. 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.

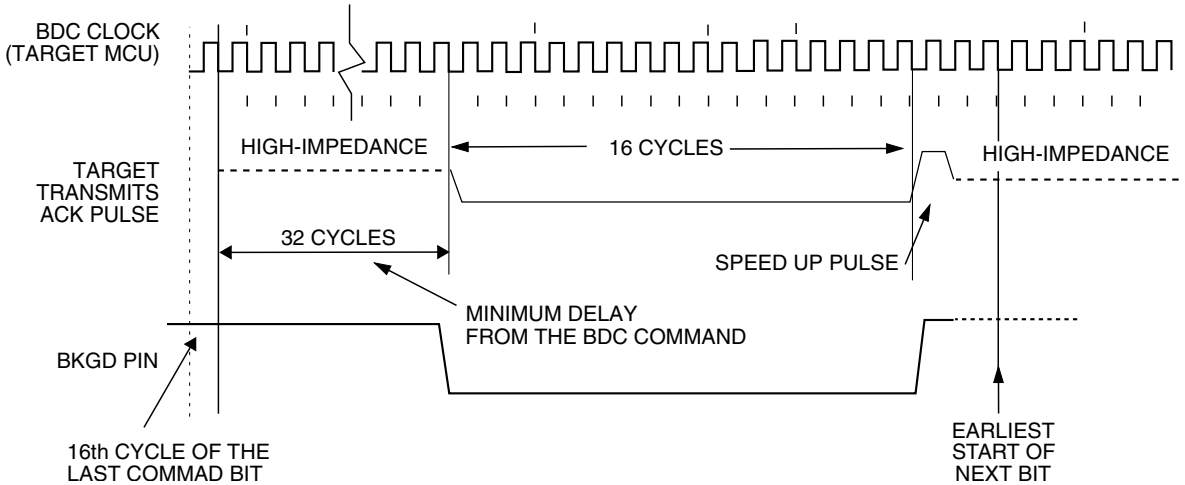


Figure 22-7. Target Acknowledge Pulse (ACK)

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 22-8 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.

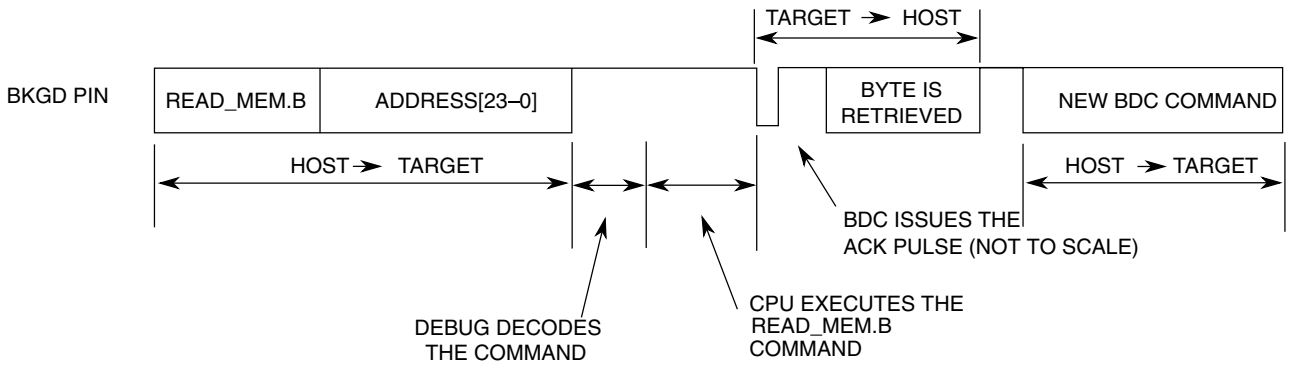


Figure 22-8. Handshake Protocol at Command Level

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 22-8 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 Hardware Handshake Abort Procedure.

22.5.1.7 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 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 1*xx*, 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

The following figure shows a SYNC command aborting a READ_MEM.B. After the command is aborted, a new command could be issued by the host.

Note

In the following figure, signal timing is not drawn to scale.

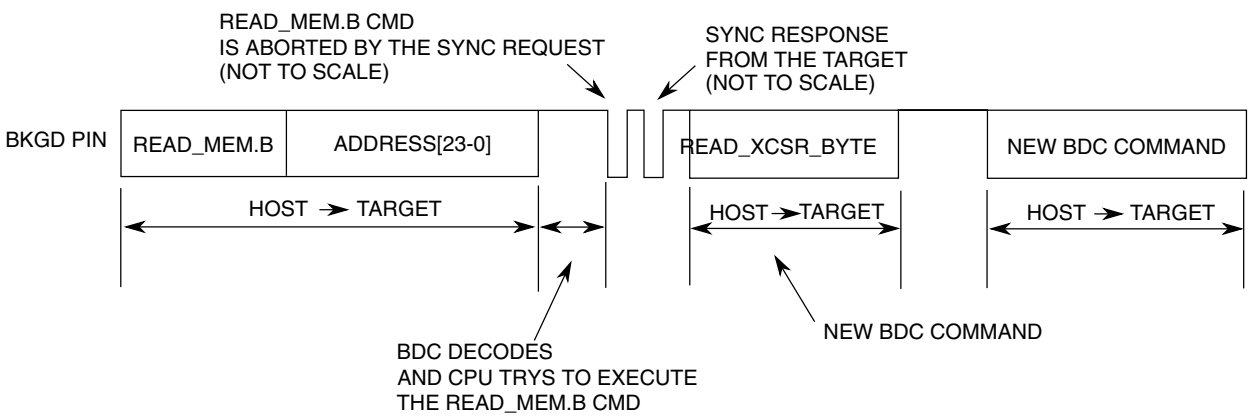


Figure 22-9. ACK Abort Procedure at the Command Level

The following figure 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.

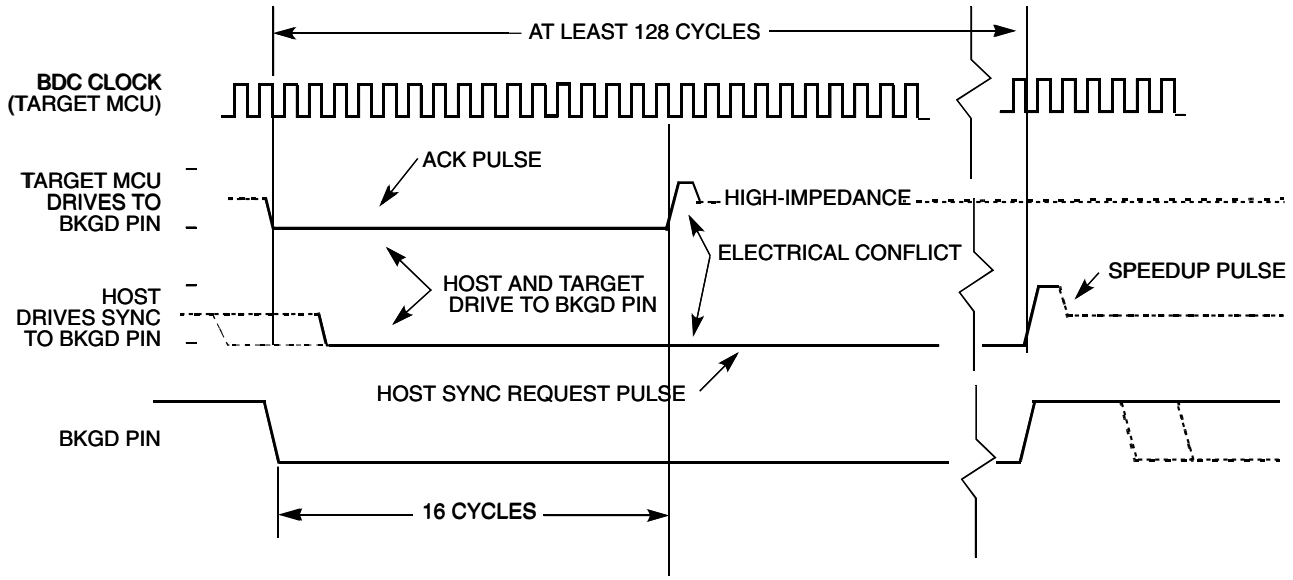


Figure 22-10. ACK Pulse and SYNC Request Conflict

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 22-38 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.

22.5.2 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.

22.5.3 Trace Support

For the baseline V1 ColdFire core and its single debug signal, support for trace functionality is completely redefined. The V1 solution provides an on-chip PST/DDATA trace buffer (known as the PSTB) to record the stream of PST and DDATA values.

As a review, the classic ColdFire debug architecture supports real-time trace via the PST/DDATA output signals. For this functionality, the following apply:

- One (or more) PST value is generated for each executed instruction
- Branch target instruction address information is displayed on all non-PC-relative change-of-flow instructions, where the user selects a programmable number of bytes of target address
  - Displayed information includes PST marker plus target instruction address as DDATA
  - Captured address creates the appropriate number of DDATA entries, each with 4 bits of address
- Optional data trace capabilities are provided for accesses mapped to the slave peripheral bus

- Displayed information includes PST marker plus captured operand value as DDATA
- Captured operand creates the appropriate number of DDATA entries, each with 4 bits of data

The resulting PST/DDATA output stream, with the application program memory image, provides an instruction-by-instruction dynamic trace of the execution path.

Even with the application of a PST trace buffer, problems associated with the PST bandwidth and associated fill rate of the buffer remain. Given that there is one (or more) PST entry per instruction, the PSTB would fill rapidly without some type of data compression.

Consider the following example to illustrate the PST compression algorithm. Most sequential instructions generate a single PST = 1 value. Without compression, the execution of ten sequential instructions generates a stream of ten PST = 1 values. With PST compression, the reporting of any PST = 1 value is delayed so that consecutive PST = 1 values can be accumulated. When a PST ≠ 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 the following table.

Table 22-39. 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.

Table continues on the next page...

Table 22-39. CF1 Debug Processor Status Encodings (continued)

PST[4:0]	Definition
0x05	Begin execution of taken branch or SYNC_PC BDM command. For some opcodes, a branch target address may be displayed on DDATA depending on the CSR settings. CSR also controls the number of address bytes displayed, indicated by the PST marker value preceding the DDATA nibble that begins the data output. This encoding also indicates that the SYNC_PC command has been processed.
0x06	Reserved
0x07	Begin execution of return from exception (RTE) instruction.
0x08–0x0B	Indicates the number of data bytes to be loaded into the PST trace buffer. The capturing of peripheral bus data references is controlled by CSR[DDC]. 0x08 Begin 1-byte data transfer on DDATA 0x09 Begin 2-byte data transfer on DDATA 0x0A Reserved 0x0B Begin 4-byte data transfer on DDATA
0x0C–0x0F	Indicates the number of address bytes to be loaded into the PST trace buffer. The capturing of branch target addresses is controlled by CSR[BTB]. 0x0C Reserved 0x0D Begin 2-byte address transfer on DDATA (Displayed address is shifted right 1: ADDR[16:1]) 0x0E Begin 3-byte address transfer on DDATA (Displayed address is shifted right 1: ADDR[23:1]) 0x0F Reserved
0x10–0x11	Reserved
0x12	Completed execution of 2 sequential instructions
0x13	Completed execution of 3 sequential instructions
0x14	Completed execution of 4 sequential instructions
0x15	Completed execution of 5 sequential instructions
0x16	Completed execution of 6 sequential instructions
0x17	Completed execution of 7 sequential instructions
0x18	Completed execution of 8 sequential instructions
0x19	Completed execution of 9 sequential instructions
0x1A	Completed execution of 10 sequential instructions
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 × 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.

Table continues on the next page...

Table 22-39. CF1 Debug Processor Status Encodings (continued)

PST[4:0]	Definition
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.

22.5.3.1 Begin Execution of Taken Branch (PST = 0x05)

The PST is 0x05 when a taken branch is executed. For some opcodes, a branch target address may be loaded into the trace buffer (PSTB) depending on the CSR settings. CSR also controls the number of address bytes loaded that is indicated by the PST marker value immediately preceding the DDATA entry in the PSTB that begins the address entries.

Multiple byte DDATA values are displayed in least-to-most-significant order. The processor captures only those target addresses associated with taken branches that use a variant addressing mode (RTE and RTS instructions, JMP and JSR instructions using address register indirect or indexed addressing modes, and all exception vectors).

The simplest example of a branch instruction using a variant address is the compiled code for a C language case statement. Typically, the evaluation of this statement uses the variable of an expression as an index into a table of offsets, where each offset points to a unique case within the structure. For such change-of-flow operations, the ColdFire processor loads the PSTB as follows:

1. Load PST=0x05 to identify that a taken branch is executed.
2. Optionally load the marker for the target address capture. Encodings 0x0D or 0x0E identify the number of bytes loaded into the PSTB.
3. The new target address is optionally available in the PSTB. The number of bytes of the target address loaded is configurable (2 or 3 bytes, where the encoding is 0x0D and 0x0E, respectively).

Another example of a variant branch instruction is a JMP (A0) instruction. The following figure shows the PSTB entries that indicate a JMP (A0) execution, assuming CSR[BTB] was programmed to display the lower two bytes of an address.

PST/DDATA Values	Description
0x05	Taken Branch
0x0D	2-byte Address Marker
{10, Address[4:1]}	Address >> 1
{10, Address[8:5]}	
{10, Address[12:9]}	
{10, Address[16:13]}	

Figure 22-11. Example JMP Instruction Output in PSTB

The PST of 0x05 indicates a taken branch and the marker value 0x0D indicates a 2-byte address. Therefore, the following entries display the lower two bytes of address register A0, right-shifted by 1, in least-to-most-significant nibble order. The next PST entry after the JMP instruction completes depends on the target instruction. See PST Trace Buffer (PSTB) Entry Format, for entry descriptions explaining the 2-bit prefix before each address nibble.

22.5.3.2 PST Trace Buffer (PSTB) Entry Format

As PST and DDATA values are captured and loaded in the trace buffer, each entry is six bits in size therefore, the type of the entry can easily be determined when post-processing the PSTB.

	5	4	3	2	1	0
PSTB[PST]	0	PST[4:0]				
Data PSTB[DDATA]	1	$\overline{R}$/W	Data[3:0]			
Address PSTB[DDATA]	1	0	Address[3:0][1]			
Reset:	—	—	—	—	—	—

1 Depending on which nibble is displayed (as determined by SDR[9:8]), Address [3:0] s (least-to-most-significant nibble order) displays four bits of the real CPU address [16:1] or [24:1].

Figure 22-12. V1 PST/DDATA Trace Buffer Entry Format

22.5.3.3 PST/DDATA Example

In this section, an example showing the behavior of the PST/DDATA functionality is detailed. Consider the following interrupt service routine that counts the interrupt, negates the IRQ, performs a software IACK, and then exits. This example is presented here because it exercises a considerable set of the PST/DDATA capabilities.

```
                    _isr:
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.l*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 only the display of 2-byte branch addresses. Operand data captures are not enabled. The sequence begins with an interrupt exception:

```
interrupt exception occurs @ pc = 5432 while in user mode
                                                  # pst   = 1c, 1c, 05, 0d
                                                  # ddata = 2a, 23, 28, 20
                                                  #         trg_addr = 083a << 1
                                                  #         trg_addr = 1074
                    _isr:
01074: 46fc 2700      mov.w   &0x2700,%sr         # pst   = 01
01078: 2f08           mov.l   %a0,-(%sp)          # pst   = 01
0107a: 2f00           mov.l   %d0,-(%sp)          # pst   = 01
0107c: 302f 0008      mov.w   (8,%sp),%d0         # pst   = 01
01080: e488           lsr.l   &2,%d0              # pst   = 01
01082: 0280 0000 00ff andi.l  &0xff,%d0           # pst   = 01
01088: 207c 0080 1400 mov.l   &int_count,%a0      # pst   = 01
0108e: 52b0 0c00      addq.l  &1,(0,%a0,%d0.l*4) # pst   = 01
01092: 11c0 a021      mov.b   %d0,IGCR0+1.w       # pst   = 01, 08c
                                                  # 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.

22.5.3.4 Processor Status, Debug Data Definition

This section specifies the ColdFire processor and debug module's generation of the processor status (PST) and debug data (DDATA) output on an instruction basis. In general, the PST/DDATA output for an instruction is defined as follows:

PST = 0x01, {PST = 0x0[89B], DDATA = operand}

where the {...} definition is optional operand information defined by the setting of the CSR, and [...] indicates the presence of one value from the list.

The CSR provides capabilities to display operands based on reference type (read, write, or both). A PST value {0x08, 0x09, or 0x0B} identifies the size and presence of valid data to follow in the PST trace buffer (PSTB) {1, 2, or 4 bytes, respectively}. Additionally, CSR[DDC] specifies whether operand data capture is enabled and what size. Also, for certain change-of-flow instructions, CSR[BTB] provides the capability to display the target instruction address in the PSTB (2 or 3 bytes) using a PST value of 0x0D or 0x0E, respectively.

22.5.3.4.1 User Instruction Set

The following table 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 22-40. 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}
addx.l	Dy,Dx	PST = 0x01
and.l	<ea>y,Dx	PST = 0x01, {PST = 0x0B, DD = source operand}
and.l	Dy,<ea>x	PST = 0x01, {PST = 0x0B, DD = source}, {PST = 0x0B, DD = destination}
andi.l	#<data>,Dx	PST = 0x01
asl.l	{Dy,#<data>},Dx	PST = 0x01
asr.l	{Dy,#<data>},Dx	PST = 0x01
bcc.{b,w,l}		if taken, then PST = 0x05, else PST = 0x01
bchg.{b,l}	#<data>,<ea>x	PST = 0x01, {PST = 0x08, DD = source}, {PST = 0x08, DD = destination}
bchg.{b,l}	Dy,<ea>x	PST = 0x01, {PST = 0x08, DD = source}, {PST = 0x08, DD = destination}
bclr.{b,l}	#<data>,<ea>x	PST = 0x01, {PST = 0x08, DD = source}, {PST = 0x08, DD = destination}
bclr.{b,l}	Dy,<ea>x	PST = 0x01, {PST = 0x08, DD = source}, {PST = 0x08, DD = destination}
bitrev.l	Dx	PST = 0x01
bra.{b,w,l}		PST = 0x05
bset.{b,l}	#<data>,<ea>x	PST = 0x01, {PST = 0x08, DD = source}, {PST = 0x08, DD = destination}
bset.{b,l}	Dy,<ea>x	PST = 0x01, {PST = 0x08, DD = source}, {PST = 0x08, DD = destination}
bsr.{b,w,l}		PST = 0x05, {PST = 0x0B, DD = destination operand}
btst.{b,l}	#<data>,<ea>x	PST = 0x01, {PST = 0x08, DD = source operand}
btst.{b,l}	Dy,<ea>x	PST = 0x01, {PST = 0x08, DD = source operand}
byterev.l	Dx	PST = 0x01
clr.b	<ea>x	PST = 0x01, {PST = 0x08, DD = destination operand}
clr.l	<ea>x	PST = 0x01, {PST = 0x0B, DD = destination operand}
clr.w	<ea>x	PST = 0x01, {PST = 0x09, DD = destination operand}
cmp.b	<ea>y,Dx	PST = 0x01, {0x08, source operand}
cmp.l	<ea>y,Dx	PST = 0x01, {PST = 0x0B, DD = source operand}
cmp.w	<ea>y,Dx	PST = 0x01, {0x09, source operand}
cmpa.l	<ea>y,Ax	PST = 0x01, {PST = 0x0B, DD = source operand}
cmpa.w	<ea>y,Ax	PST = 0x01, {0x09, source operand}
cmpi.b	#<data>,Dx	PST = 0x01
cmpi.l	#<data>,Dx	PST = 0x01
cmpi.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

Table continues on the next page...

Table 22-40. PST/DDATA Specification for User-Mode Instructions (continued)

Instruction	Operand Syntax	PST/DDATA
extb.l	Dx	PST = 0x01
illegal		PST = 0x01
jmp	<ea>y	PST = 0x05, {PST = 0x0[DE], DD = target address}[2]
jsr	<ea>y	PST = 0x05, {PST = 0x0[DE], DD = target address}, {PST = 0x0B, DD = destination operand}
lea.l	<ea>y,Ax	PST = 0x01
link.w	Ay,#<displacement>	PST = 0x01, {PST = 0x0B, DD = destination operand}
lsl.l	{Dy,#<data>},Dx	PST = 0x01
lsr.l	{Dy,#<data>},Dx	PST = 0x01
mov3q.l	#<data>,<ea>x	PST = 0x01, {PST = 0x0B,DD = destination operand}
move.b	<ea>y,<ea>x	PST = 0x01, {PST = 0x08, DD = source}, {PST = 0x08, DD = destination}
move.l	<ea>y,<ea>x	PST = 0x01, {PST = 0x0B, DD = source}, {PST = 0x0B, DD = destination}
move.w	<ea>y,<ea>x	PST = 0x01, {PST = 0x09, DD = source}, {PST = 0x09, DD = destination}
move.w	CCR,Dx	PST = 0x01
move.w	{Dy,#<data>},CCR	PST = 0x01
movea.l	<ea>y,Ax	PST = 0x01, {PST = 0x0B, DD = source}
movea.w	<ea>y,Ax	PST = 0x01, {PST = 0x09, DD = source}
movem.l	#list,<ea>x	PST = 0x01, {PST = 0x0B, DD = destination},...
movem.l	<ea>y,#list	PST = 0x01, {PST = 0x0B, DD = source},...
moveq.l	#<data>,Dx	PST = 0x01
muls.l	<ea>y,Dx	PST = 0x01, {PST = 0x0B, DD = source operand}
muls.w	<ea>y,Dx	PST = 0x01, {PST = 0x09, DD = source operand}
mulu.l	<ea>y,Dx	PST = 0x01, {PST = 0x0B, DD = source operand}
mulu.w	<ea>y,Dx	PST = 0x01, {PST = 0x09, DD = source operand}
mvs.b	<ea>y,Dx	PST = 0x01, {0x08, source operand}
mvs.w	<ea>y,Dx	PST = 0x01, {0x09, source operand}
mvz.b	<ea>y,Dx	PST = 0x01, {0x08, source operand}
mvz.w	<ea>y,Dx	PST = 0x01, {0x09, source operand}
neg.l	Dx	PST = 0x01
negx.l	Dx	PST = 0x01
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}

Table continues on the next page...

Table 22-40. PST/DDATA Specification for User-Mode Instructions (continued)

Instruction	Operand Syntax	PST/DDATA
sats.l	Dx	PST = 0x01
scc.b	Dx	PST = 0x01
sub.l	<ea>y,Dx	PST = 0x01, {PST = 0x0B, DD = source operand}
sub.l	Dy,<ea>x	PST = 0x01, {PST = 0x0B, DD = source}, {PST = 0x0B, DD = destination}
suba.l	<ea>y,Ax	PST = 0x01, {PST = 0x0B, DD = source operand}
subi.l	#<data>,Dx	PST = 0x01
subq.l	#<data>,<ea>x	PST = 0x01, {PST = 0x0B, DD = source}, {PST = 0x0B, DD = destination}
subx.l	Dy,Dx	PST = 0x01
swap.w	Dx	PST = 0x01
tas.b	<ea>x	PST = 0x01, {0x08, source}, {0x08, destination}
tpf		PST = 0x01
tpf.l	#<data>	PST = 0x01
tpf.w	#<data>	PST = 0x01
trap	#<data>	PST = 0x01[1]
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}

1. 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 excetion. 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.

2. 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).

Table 22-41 shows the PST/DDATA specification for the MAC instructions if the optional MAC unit is present.

Table 22-41. PST/DDATA Values for Multiply-Accumulate Instructions

Instruction	Operand Syntax	PST/DDATA
mac.l	Ry,Rx	PST = 0x1
mac.l	Ry,Rx,<ea>y,Rw	PST = 0x1, {PST = 0xB, DD = source operand}
mac.w	Ry,Rx	PST = 0x1
mac.w	Ry,Rx,ea,Rw	PST = 0x1, {PST = 0xB, DD = source operand}
move.l	{Ry,#<data>},ACC	PST = 0x1
move.l	{Ry,#<data>},MACSR	PST = 0x1
move.l	{Ry,#<data>},MASK	PST = 0x1
move.l	ACC,Rx	PST = 0x1
move.l	MACSR,CCR	PST = 0x1
move.l	MACSR,Rx	PST = 0x1
move.l	MASK,Rx	PST = 0x1
msac.l	Ry,Rx	PST = 0x1
msac.l	Ry,Rx,<ea>y,Rw	PST = 0x1, {PST = 0xB, DD = source}, {PST = 0xB, DD = destination}
msac.w	Ry,Rx	PST = 0x1
msac.w	Ry,Rx,<ea>y,Rw	PST = 0x1, {PST = 0xB, DD = source}, {PST = 0xB, DD = destination}

22.5.3.4.2 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 the following table.

Table 22-42. 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},

Table continues on the next page...

Table 22-42. PST/DDATA Specification for Supervisor-Mode Instructions (continued)

Instruction	Operand Syntax	PST/DDATA
		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.

22.5.4 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_{DD}. An open-drain connection to reset allows the host to force a target system reset, useful to regain control of a lost target system or to control startup of a target system before the on-chip nonvolatile memory has been programmed. Sometimes V_{DD} can be used to allow the pod to use power from the target system to avoid the need for a separate power supply. However, if the pod is powered separately, it can be connected to a running target system without forcing a target system reset or otherwise disturbing the running application program.

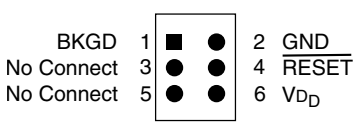


Figure 22-13. Recommended BDM Connector

Appendix A
Revision History

A.1 Revision History

Revision number	Revision date	Description
0.6	May 2013	**Initial Public Release (Preliminary)**

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

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

Freescale reserves the right to make changes without further notice to any products herein. Freescale makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale 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 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 does not convey any license under its patent rights nor the rights of others. Freescale sells products pursuant to standard terms and conditions of sale, which can be found at the following address: freescale.com/SalesTermsandConditions.

Freescale, the Freescale logo, AltiVec, C-5, CodeTest, CodeWarrior, ColdFire, ColdFire+, C-Ware, Energy Efficient Solutions logo, Kinetis, mobileGT, PowerQUICC, Processor Expert, QorIQ, Qorivva, StarCore, Symphony, and VortiQa are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. Airfast, BeeKit, BeeStack, CoreNet, Flexis, Layerscape, MagniV, MXC, Platform in a Package, QorIQ Qonverge, QUICC Engine, Ready Play, SafeAssure, SafeAssure logo, SMARTMOS, Tower, TurboLink, Vybrid, and Xtrinsic are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners.

© 2013 Freescale Semiconductor, Inc.

Document Number FXLC95000CLHWRM
Rev 0.6, May 2013
14 May 2013

Preliminary