THE DATASHEET OF

KITMMA9550LEVM

MMA955xL Intelligent, Motion-Sensing Platform Hardware Reference Manual

Devices Supported:

MMA9550L

MMA9551L

MMA9553L

MMA9559L

Document Number: MMA955xLHWRM
Rev. 1.0, 8/2013

Energy Efficient Solutions by Freescale™

freescale Xtrinsic

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: MMA955xLHWRM
Rev. 1.0
8/2013

Contents

Chapter 1 About This Document

Chapter 2 Introduction

Chapter 3 Pins and Connections

Chapter 4 Operational Phases and Modes of Operation

Chapter 5 Memory Maps

Chapter 6 Flash Memory Controller

Chapter 7 ROM

Chapter 8 Slave Port Interface

Chapter 9 Inter-Integrated Circuit

Chapter 10 Analog Front End

Chapter 11 System Integration Module

Chapter 12 On-Chip Oscillator

Chapter 13 Programmable Delay Block

Chapter 14 Port Controls

Chapter 15 Rapid GPIO

Chapter 16 Pin Interrupt Function

Chapter 17 16-Bit Modulo Timer

Chapter 18 Timer/PWM Module

Chapter 19 Interrupt Controller

Chapter 20 ColdFire Core

Chapter 21 Version 1 ColdFire Debug

Chapter 1 About This Document

1.1 Overview

1.1.1 Purpose

This reference manual describes the features, architecture and programming model of the MMA955xL platform, an intelligent, three-axis accelerometer.

1.1.2 Audience

This document is primarily for system architects and software application developers who are using or considering use of the MMA955xL in a system.

1.2 Terms and acronyms

AFE	Analog Front End
APP_ID	Application identifier
API	Application Programming Interface
BDM	Background Debug Module
CC	Command Complete
CI	Command Interpreter
CMD	Command
COCO	Conversion Complete or Command Complete
CSR	ColdFire Configuration Status Register
DFC	Data Format Code
DTAP	Double tap (n.)
FIFO	First In First Out, a data structure
FOPT	Flash Options register
GPIO	General-Purpose Input/Output, a microcontroller pin that can be programmed by software
hash	A deterministic, cryptographic function that converts an arbitrary block of data into a fixed-size bit string—the (cryptographic) hash value—such that an accidental or intentional change to the data will change that hash value
HG	High g

JTAG	Joint Test Action Group (JTAG), the common name for the IEEE 1149.1 standard *Standard Test Access Port and Boundary-Scan Architecture*, for test-access ports
LG	Low g
LL	Landscape Left
LR	Landscape Right
MBOX	Mailbox
MCU	Microcontroller
MTIMOV	Module Timer Overflow Module
PC	Program Counter
PD	Portrait Down
PDB	Program Delay Block
PL	Portrait/Landscape
POR	Power-on Reset
PU	Portrait Up
SFD	Start Frame Digital
Shared secret	Encrypted data known only to the parties involved in a secure communication. Can include a password, a pass phrase, a big number, or an array of randomly chosen bytes.
SSP	Supervisor Stack Pointer
TPM	Timer Program Module
VBR	Vector Base Register, a register in the ColdFire memory map that controls the location of the exception vector table

1.3 Conventions

This document uses the following notational conventions:

cleared/set	When a bit takes the value 0, it is said to be cleared; when it takes a value of 1, it is said to be set.
MNEMONICS	In text, instruction mnemonics are shown in uppercase.
`mnemonics`	In code and tables, instruction mnemonics are shown in lowercase.
italics	Italics indicate variable command parameters. Book titles also are italicized.
0x0	Prefix to denote a hexadecimal number
0b0	Prefix to denote a binary number
REG[FIELD]	Abbreviations for registers are shown in uppercase. Specific bits, fields or ranges appear in brackets. For example, RAMBAR[BA] identifies the base address field in the RAM base-address register.
nibble	A 4-bit data unit
byte	An 8-bit data unit
word	A 16-bit data unit
longword	A 32-bit data unit
x	In some contexts, such as signal encodings, x indicates a “do not care.”
n	Used to express an undefined numerical value.
~	NOT logical operator
&	AND logical operator
|	OR logical operator
||	Field concatenation operator
$\overline{\text{OVERBAR}}$	Indicates that a signal is active-low.

1.4 Register figure conventions

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

—	The bit is undefined at reset.
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:

Read Write	0	Indicates a reserved bit field in a memory-mapped register. These bits are always read as 0.
Read Write	1	Indicates a reserved bit field in a memory-mapped register. These bits are always read as 1.
Read Write	FIELDNAME	Indicates a read/write bit.
Read Write	FIELDNAME	Indicates a read-only bit field in a memory-mapped register.
Read Write	FIELDNAME	Indicates a write-only bit field in a memory-mapped register.
Read Write	FIELDNAME w1c	Write 1 to clear: indicates that writing a 1 to this bit field clears it.
Read Write	0 FIELDNAME	Indicates a self-clearing bit.

1.5 References

1. MMA955xL intelligent, motion-sensing platform documentation: “MMA955xL: Product Documentation Page”
2. IEEE Standard Test Access Port and Boundary-Scan Architecture, IEEE Std. 1149.1™-2001 (R2008)
3. The I^2C-Bus Specification Version 2.1, January 2000, Philips Semiconductors
4. I^2C-Bus Specification and User Manual, NXP Semiconductors Document UM10204, Rev. 03 - 19 June 2007
5. ColdFire Family Programmer’s Reference Manual, Freescale Semiconductor, CFPRM Rev. 3, 02/2005
6. Wikipedia entry for “Semaphore”: http://en.wikipedia.org/wiki/Semaphore_(programming)
7. *ITU-T V.41 Recommendation: Code-Independent Error Control System*, available at http://www.itu.int/publications/index.html.
8. *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.
9. *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

The MMA955xL three-axis accelerometer is a member of Freescale's Xtrinsic family of intelligent sensor platforms. This device incorporates dedicated accelerometer MEMS transducers, signal conditioning, data conversion and a 32-bit, programmable microcontroller.

This unique blend transforms Freescale's MMA955xL device into an intelligent, high-precision motion-sensing platform able to manage multiple sensor inputs and make system-level decisions required for sophisticated applications such as gesture recognition, tilt computation, and pedometer functionality.

The MMA955xL platform is programmed and configured with CodeWarrior Development Studio software. This integrated-design environment enables customers to quickly and easily shape and implement custom algorithms and features to exactly match their application needs.

Using its master I^2C module, the MMA955xL platform can manage secondary sensors such as pressure sensors, magnetometers or gyroscopes. This allows sensor initialization, calibration, data compensation and computation functions to be off-loaded from the system application processor. Multiple sensor inputs can be easily consolidated by the MMA955xL device which acts as an intelligent sensing hub and highly configurable decision engine. Total system power consumption is significantly reduced as the application processor stays powered down until absolutely needed.

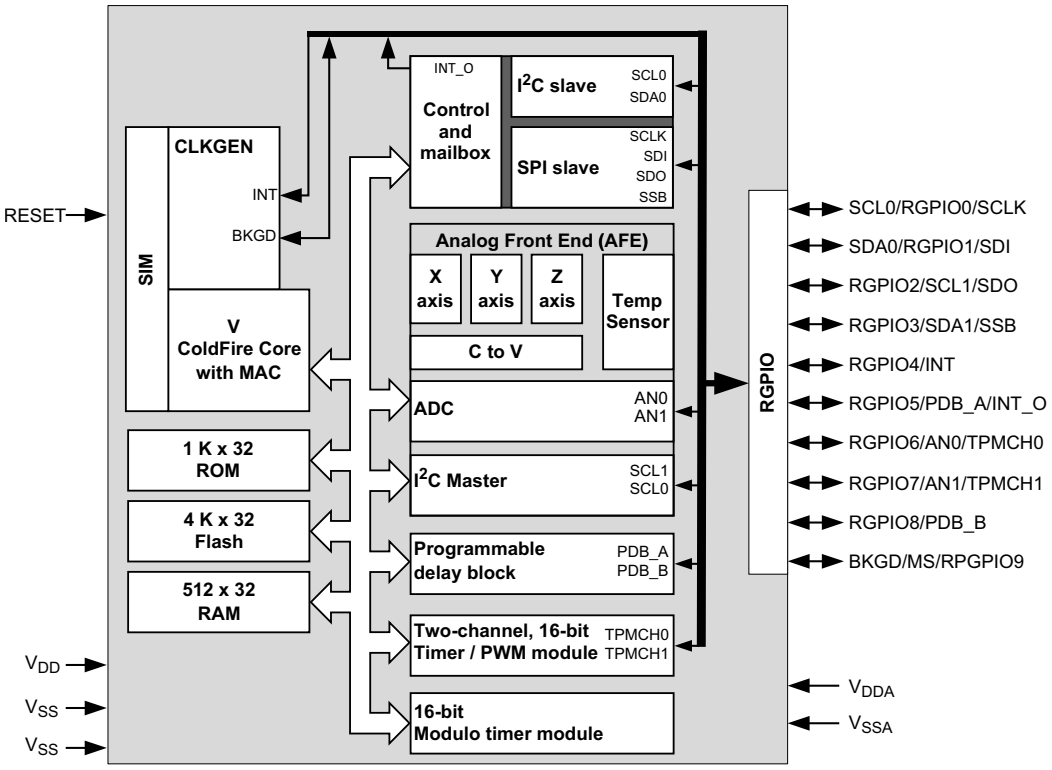


Figure 2-1. Platform block diagram

2.1 Hardware features

- Three accelerometer operating ranges:
  - ±2g: Suits most user-interaction (mouse) motions and free fall
  - ±4g: Covers most regular human dynamics (walking, jogging)
  - ±8g: Detects most abrupt activities (gaming)
- Integrated temperature sensor
- One slave SPI or I^2C interface operates up to 2 MHz dedicated to communication with host processor
- One master I^2C interface operates up to 400 kbps used to communicate with external sensors
- 10, 12, 14 and 16-bit ADC trimmed data formats available.
- 1.8V Supply voltage
- 32-bit ColdFire V1 CPU
- Extensive set of power management features and low power modes.
- Single Wire Background Debug Mode (BDM) pin interface
- 16 KB Flash memory
- 2 KB random access memory
- ROM-based flash controller and slave port command line interpreter
- Two channel timer with input capture, output capture or edge-aligned PWM
- Programmable delay block for scheduling events relative to start of frame
- Modulo timer for scheduling periodic events

2.2 Software features

This device may be programmed to provide any of the following:

- Orientation detection (portrait/landscape)
- High-g/low-g threshold detection
- Pulse detection (single, double and directional tap)
- Auto wake/sleep
- Linear and rotational free fall
- Flick detection
- Embedded smart FIFO
- Power management
- Pedometer
- Shock, vibration and sudden-motion detection
- Tilt-angle computation

The association of a high-performance accelerometer with a powerful, embedded ColdFire V1 MCU core gives the possibility to grow and customize this list in an unprecedented way.

2.3 Typical applications

This low-power intelligent sensor is optimized for use in portable and mobile consumer products such as:

- Mobile phones/PMPs/PDAs/digital cameras
  - Orientation detection (portrait/landscape)
  - Image stability
  - Tilt control enabled with higher resolution
  - Gesture recognition
  - Tap to control
  - Auto Wake/Sleep for low power consumption
- Smartbooks/eReaders/netbooks/laptops
  - Anti-theft
  - Freefall detection for Hard Disk Drives
  - Orientation Detection
  - Tap Detection
- Pedometers
- Gaming and toys
- Personal navigation devices (PNDs)
- Public transportation ticketing systems
- Activity monitoring in medical applications
- Security
  - Anti-theft
  - Shock detection
  - Tilt
- Fleet monitoring, tracking
  - Dead reckoning
  - System auto wake-up on movement
  - Detection
  - Shock recording
  - Anti-theft
- Power tools and small appliances
  - Tilt
  - Safety shutoff

Chapter 3 Pins and Connections

3.1 Package pinout

The package pinout for this device provides a superset of functions found on competitive devices, as well as other Freescale accelerator devices. All pins on the device are utilized and many pins have multiple possible uses.

Users may select from multiple pin functions via the SIM pin mux-control registers. (See “Pin Mux Control Register0” on page 212.)

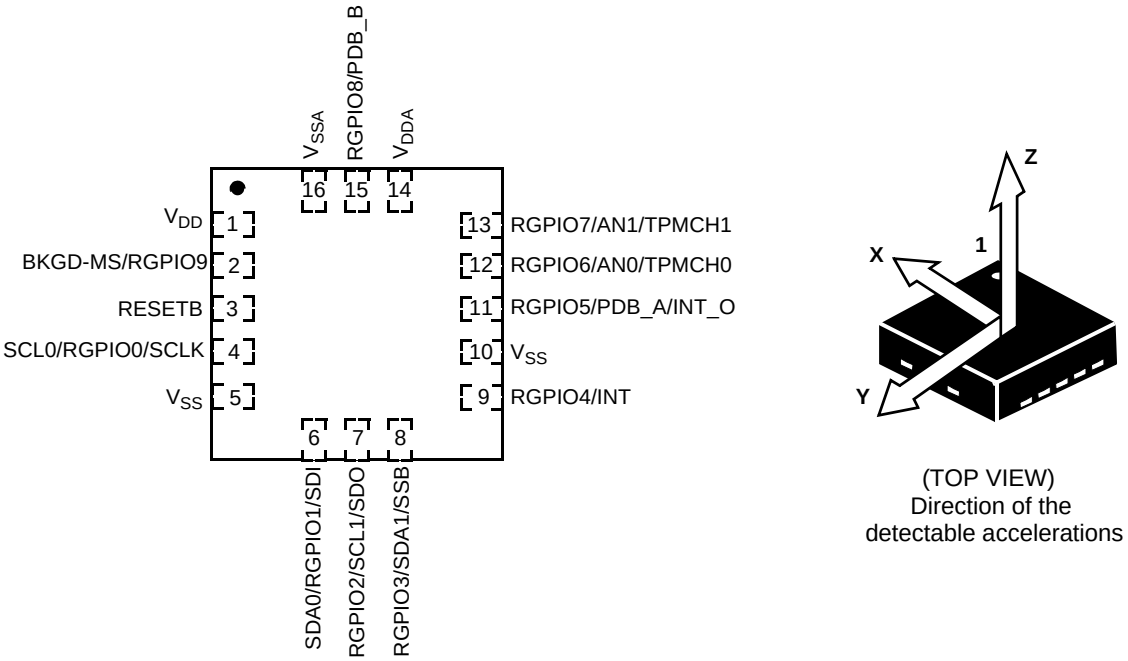


Figure 3-1. Device pinout and coordinate system

3.1.1 Pin functions

Table 3-1 summarizes functional options for each of the device's pins.

Table 3-1. Pin functions

Pin #	Pin function #1[1]	Pin function #2	Pin function #3	Description
1	V_{DD}			Digital power supply
2	BKGD/MS	RGPIO9		Background debug/mode select RGPIO9
3	RESETB[2]			Active low reset
4	SCL0	RGPIO0	SCLK	Serial clock for slave I^2C/RGPIO0/Serial clock for slave SPI
5	V_{SS}			Digital ground
6	SDA0	RGPIO1	SDI	Serial data for slave I^2C/RGPIO1/SPI serial data input
7	RGPIO2	SCL1	SDO	RGPIO2/Serial clock for master I^2C/SPI serial data output
8[3]	RGPIO3	SDA1	SSB	RGPIO3/Serial data for master I^2C/SPI slave select
9	RGPIO4	INT		RGPIO4/Interrupt input
10	RESERVED (Connect to V_{SS})			Must be connected to GND externally
11	RGPIO5	PDB_A	INT_O	RGPIO5/PDB_A/INT_O slave port interrupt output
12	RGPIO6	AN0	TPMCH0	RGPIO6/ADC Input 0 / TPM Channel 0
13	RGPIO7	AN1	TPMCH1	RGPIO7/ADC Input 1 / TPM Channel 1
14	V_{DDA}			Analog power
15	RGPIO8	PDB_B		RGPIO8/PDB_B
16	V_{SSA}			Analog ground

1 **Pin Function 1 represents the reset state of the device. Pin functions may be changed via the SIM pin mux-control registers (“Pin Mux Control Register0”).**

2 RESETB is an open-drain, bidirectional pin. By default, the output function is not on.

3 RGPIO3/SDA1/SSB = LOW at startup indicates that SPI should be used as slave instead of the I^2C module.

3.1.2 Sensing direction and output response

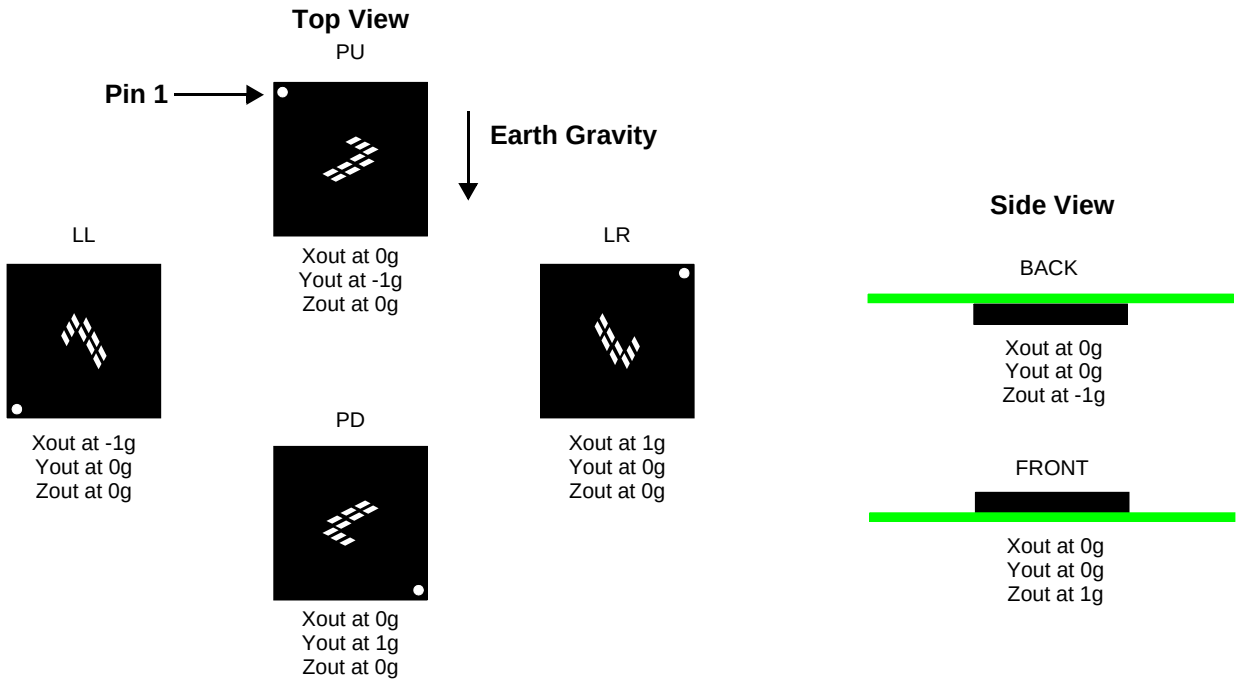


Figure 3-2. Sensing direction and output response

3.2 Pin descriptions

The following sections provide descriptions of the various pin functions available on the MMA955xL devices. Ten of the device pins are multiplexed with Rapid GPIO (RGPIO) functions. (See “Rapid GPIO” on page 239.) The “Primary Pin Function #1” column of Table 3-1 lists the functions that are active when the device exits the reset state. The pin mux control registers in the System Integration Module (or SIM) can be used to change pin assignments for these pins after reset. (See “System Integration Module” on page 189.)

3.2.1 V_{DD} and V_{SS}

These are the digital power and ground pins and must be connected to the same voltage. V_{DD} is nominally 1.8V for this device.

3.2.2 V_{DDA} and V_{SSA}

These are the analog-power and ground pins. V_{DDA} is nominally 1.8V for this device and must be filtered to remove any digital noise that may be present on the supply. V_{DDA} is usually connected to V_{DD} through an appropriate filtering network.

3.2.3 RESETB

The RESETB pin is an open-drain, bidirectional pin with an internal weak pull-up resistor. At power-up, it is configured strictly as an input pin. Setting RCSR[DR] (Reset Control and Status Register “Drive Reset” bit) to 1 will cause the RESET function to become bidirectional. (See “Reset Control and Status Register” on page 201.) Using this feature, the MMA955xL platform can reset external devices whenever it is reset for any purpose other than power-on-reset.

3.2.4 Slave I^2C: SDA0 and SCL0

These are the slave I^2C data and clock signals, respectively. The MMA955xL platform may be controlled via this serial port or through the slave SPI interface.

State at reset: Open-drain, bidirectional in input mode, pull-up resistor disabled.

3.2.5 Master I^2C: SDA1 and SCL1

These are the master I^2C clock and data signals, respectively. Because the MMA955xL device contains a 32-bit ColdFire V1 CPU, it is fully capable of mastering other devices in the system via this serial port. (For details, see “Inter-Integrated Circuit” on page 153.) This allows the MMA955xL platform to off-load certain tasks from the main CPU, allowing it to conserve power by entering sleep mode. The MMA955xL device can then issue a wake-up interrupt to the main CPU when motion is detected by the on-chip transducer or when a slave device (such as pressure sensor or magnetometer) flags that activity has occurred.

State at reset: Inactive. SDA1 and SCL1 are secondary functions on RGPIO[3:2], which owns the pins at reset.

3.2.6 Analog-to-digital conversion: AN0, AN1

The on-chip ADC can be used to perform a differential, analog-to-digital conversion based on the voltage present across pins AN0(-) and AN1(+). Conversions on these pins are subject to the same output data rate (ODR) rules as the MEMS transducer signals.

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

3.2.7 Rapid General-Purpose I/O: RGPIO[9:0]

The ColdFire V1 CPU has a feature called “Rapid GPIO” (RGPIO). This is a 16-bit input/output port with single-cycle write, set, clear and toggle functions available to the CPU. The MMA955xL platform brings out the lower 10 bits of that port as pins of the device.

State at reset:

- RGPIO[9]: Inactive. BKGD/MS owns the pin at reset.
- RGPIO[8:2]: Pin mux registers for these bits are configured as RGPIO. Pull-ups are disabled. RGPIO functionality can be enabled via RGPIO_ENB[8:2].
- RGPIO[1:0]: Inactive. SDA0 and SCL0 own the pin at reset.

3.2.8 Interrupts: INT

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.

State at reset: Inactive. RGPIO[4] owns the pin at reset.

3.2.9 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 can operate as a bidirectional, single-wire background debug port. It can be used by development tools for downloading code into on-chip RAM and flash and to debug code.

State at reset: Mode Select (MS).

- MS = 0 at exit from reset => Boot to debug halt mode
- MS = 1 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 additional details, see “Version 1 ColdFire Debug” on page 347.

3.2.10 Timer: PDB_A and PDB_B

These are the two outputs of the programmable delay block described in “Programmable Delay Block” on page 221. Normally, the PDB is used to schedule internal events at some fixed interval(s) with respect to the start of either an analog or digital phase. By bringing the PDB outputs to these pins, it becomes possible for the MMA955xL device to initiate some external event, also with respect to start of analog or digital phase.

3.2.11 Slave SPI interface: SCLK, SDI, SDO and SSB

These are the slave SPI clock, data in, data out and slave select signals, respectively. The MMA955xL platform may be controlled via this serial port or via the slave I^2C interface.

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

3.3 System connections

3.3.1 Platform as an intelligent slave

Figure 3-3 shows an example of the complete system connections when the MMA955xL platform is used as a smart-accelerometer, slave peripheral to a host processor.

All that are required to attach the MMA955xL device to a master CPU are I^2C termination resistors, a ferrite bead and a few bypass capacitors. Optionally, the RGPIO pins can be programmed to generate interrupts in order to wake the master CPU, as required by any changes in the inertial input. In the latter case, the interrupts should be routed to the external interrupt input pins of the master CPU.

Figure 3-3 includes the background debug header connections as well as a manual reset push button.

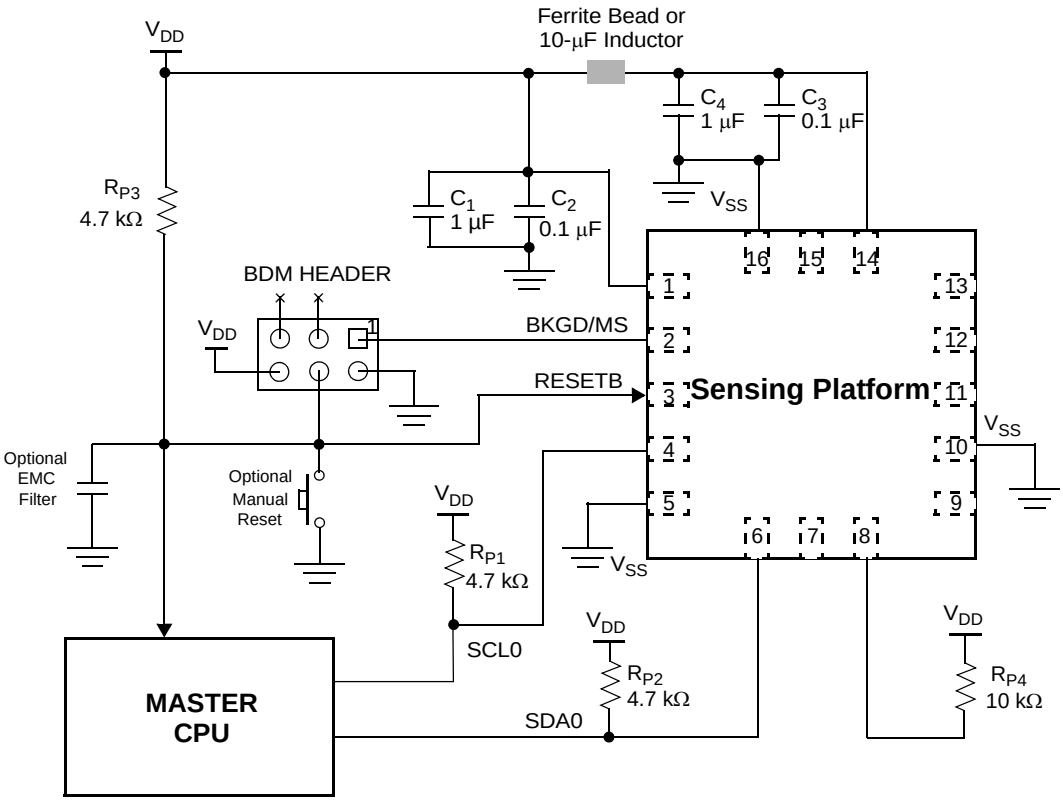


Figure 3-3. Platform as a slave with BDM header and reset button

3.3.2 Platform as a sensor hub

Figure 3-4 shows an example of the system connections when the MMA955xL platform is used as an autonomous sensor hub. This type of connection increases the overall system efficiency as the various sensors are handled directly by the MMA955xL device, through its master I²C bus and analog inputs.

In such a sensor-hub configuration, the MMA955xL platform processes and fuses the sensors' data before transferring it to the host platform, so that data is refined as higher-level information. The master CPU can go into Sleep mode as the MMA955xL device will issue a wake-up request should any external event require the CPU's attention. The RGPIO8 pin (Pin 15) is typically used for wake-up requests.

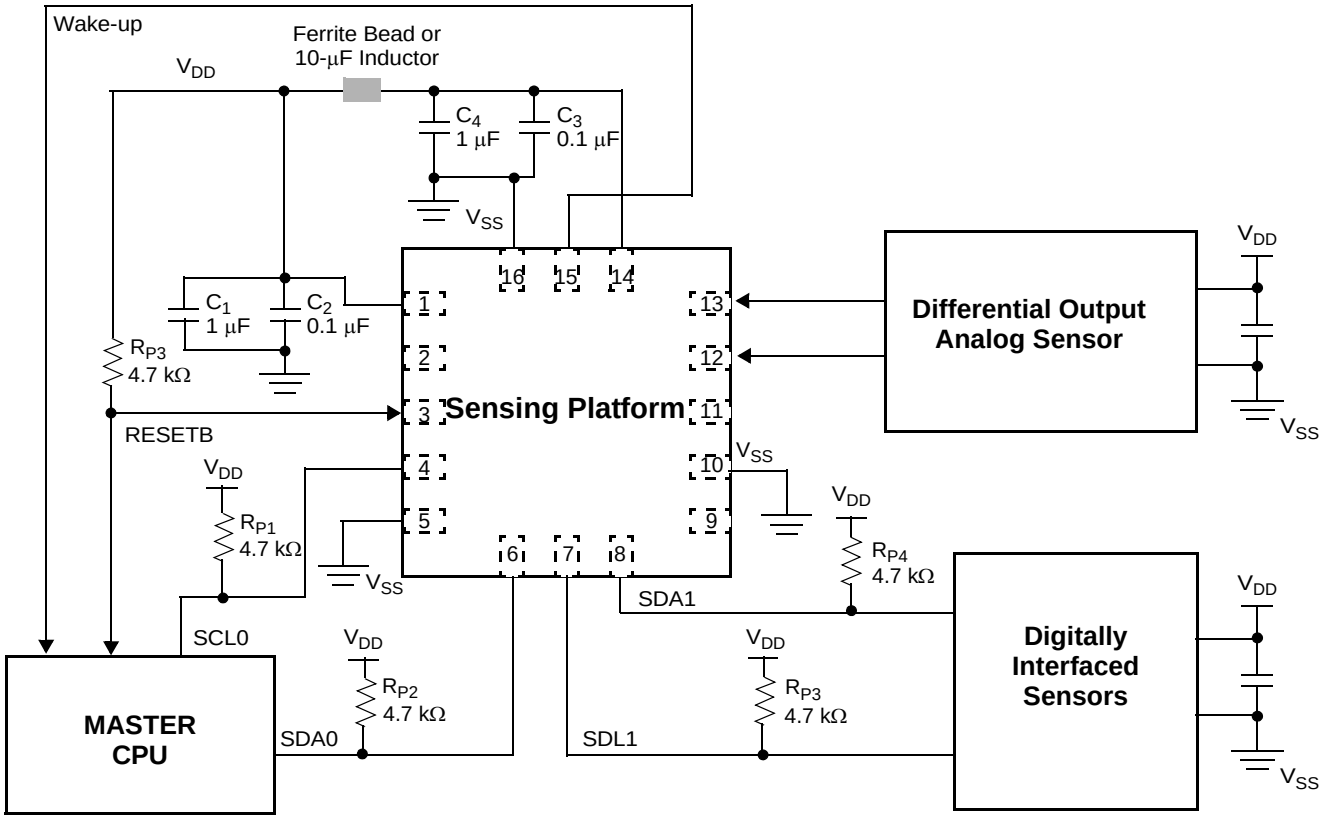


Figure 3-4. Platform as a sensor hub

3.3.3 Power

An internal circuit powered by V_{DDA} provides the MMA955xL platform with a power-on-reset signal. In order 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 0.1V. 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 VDDA and VSSA traces. Additionally, two

ceramic capacitors (of approximately 1 µF, ± 0.1 µF) can be used to efficiently bypass the power and ground of both digital and analog supply rails.

3.3.4 RESETB pin

Figure 3-3 on page 31 illustrates an exhaustive arrangement 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, pull-up 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 in order to reduce EMC and noise susceptibility. With the shunt capacitor, the maximum RC time constant has to be strictly bounded. (For details, see “System Integration Module” on page 189.)

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 MMA955xL platform 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 micro-seconds.

3.3.5 Background / mode select (BKGD/MS)

Figure 3-3 on page 31 depicts the connection to the BKGD/MS pin when 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 that may increase RESETB settling time. This situation must be considered in the user’s implementation.

Chapter 4 Operational Phases and Modes of Operation

4.1 Modes of operation

The V1 ColdFire core supports RUN, HALT, RESET and STOP modes natively. These are present on any ColdFire-based product. The MCU integration adds additional controls to STOP mode, effectively creating three modes where one existed previously.

The set of modes then becomes:

RUN	The CPU executes instructions in this mode that can be further subdivided into User and Supervisor modes.
HALT	Version 1 ColdFire Core HALT/DEBUG mode
RESET	Reset asserted. Circuitry in default state. RESET can be divided into several phases of operation. For details, see “System Integration Module” on page 189.
$STOP_{FC}$	STOP – Clock in Fast Mode – Nominally used for $\Phi_{\mathbf{A}}$ (See “Overview” on page 35.)
$STOP_{SC}$	STOP – Clock in Low Speed Mode – Nominally used for $\Phi_{\mathbf{I}}$ (See “Overview” on page 35.)
$STOP_{NC}$	STOP – No clocks – All clocks disabled. Nominally used for the SLEEP phase.

4.2 Frame structure

In addition to the modes above, the MMA955xL platform is designed to facilitate a “frame-based” software scheduler. Analog sensor conversions are best executed when the CPU is quiet and there may be times when both AFE and CPU are dormant. The MMA955xL device includes hardware mechanisms to make it easy to schedule these different functions.

4.3 Overview

The MMA955xL platform can be programmed to take a continuous sequence of evenly spaced samples over time. This section specifies the terms for timing and phases. Figure 4-1 on page 36, Figure 4-2 on page 36 and Figure 4-3 on page 36 illustrate a number of these terms that will be subsequently defined.

Timing is defined in terms of “frames.” There are 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), Frame Rate (FR) and Sample Data Rate (SDR) are three important terms 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 Over Sample Ratio (OSR) 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.

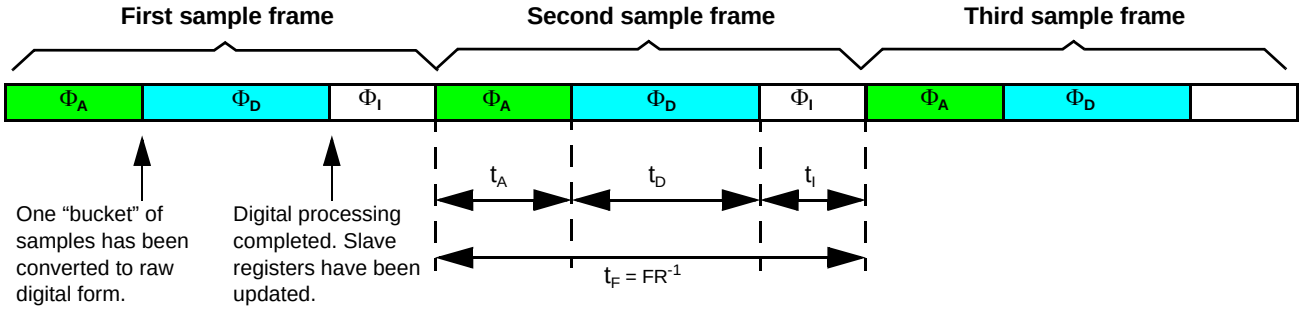


Figure 4-1. Sample-frame timing

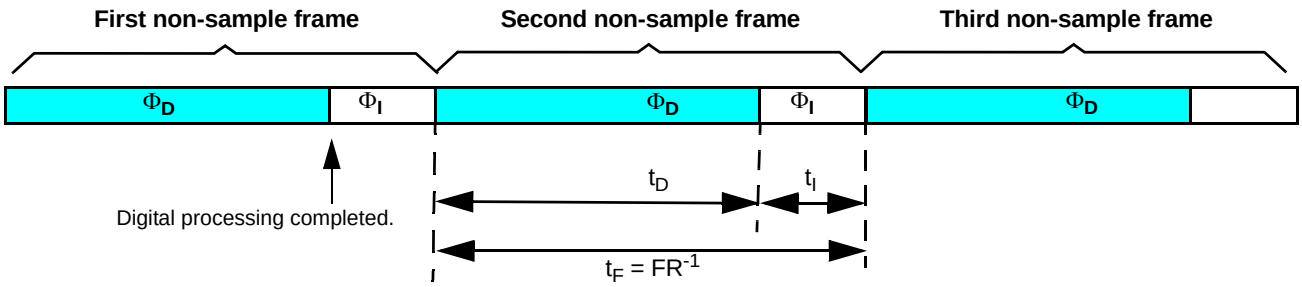


Figure 4-2. Non-sample frame timing

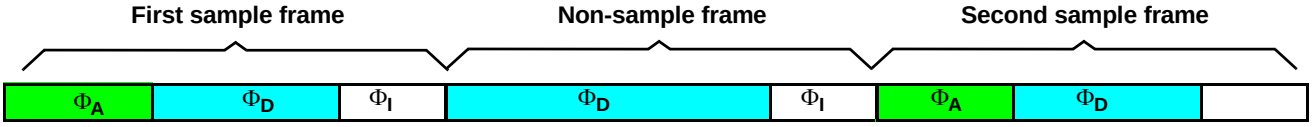


Figure 4-3. Mixed-frame timing

4.3.1 Definitions

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 MMA955xL platform provides conversion data to the user for a given quantity. This will be SDR/OSR.
Over-Sample Ratio (OSR)	The MMA955xL device 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 here. Depending on configuration data, 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 occurs here. 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.
Sample Data Rate (SDR)	The rate at which the MMA955xL device 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 equal to 1/FR.

4.3.2 Additional timing parameters

Additional terms that occasionally factor into the discussions include:

$F_{osc\text{-}high}$	The high-speed frequency of the on-chip oscillator. This is nominally 8 MHz.
$F_{osc\text{-}low}$	The low-speed frequency of the on-chip oscillator. This is nominally $F_{osc\text{-}high}/128$.
$P_{osc\text{-}high}$	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 ($= 1/F_{osc\text{-}low} = 128/F_{osc\text{-}high}$).

4.3.3 Phase triggers

Figure 11-1 on page 190 illustrates some of the major interactions between modules in this device:

- The “start-of-frame” signal is generated by the frame interval counter.
- 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 4-4 illustrates sequencing of the “Start Φ_A“and “Start Φ” hardware triggers. Figure 4-5 on page 39 shows that a STOP instruction (with STOPCR[SC] = 1) is required to transition into the idle phase. (See the SIM register map.)

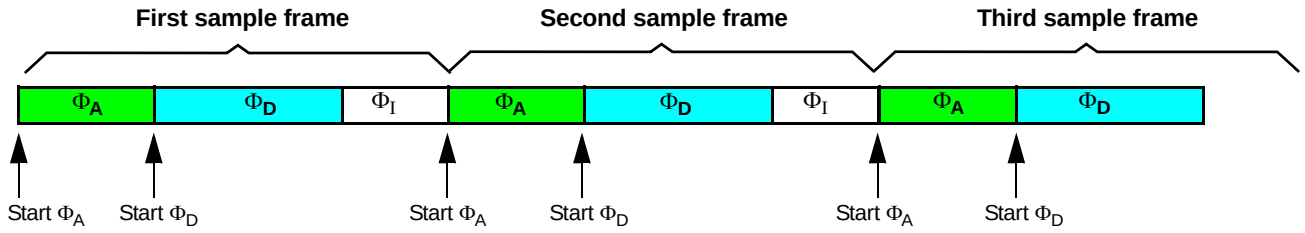


Figure 4-4. Phase triggers required in hardware

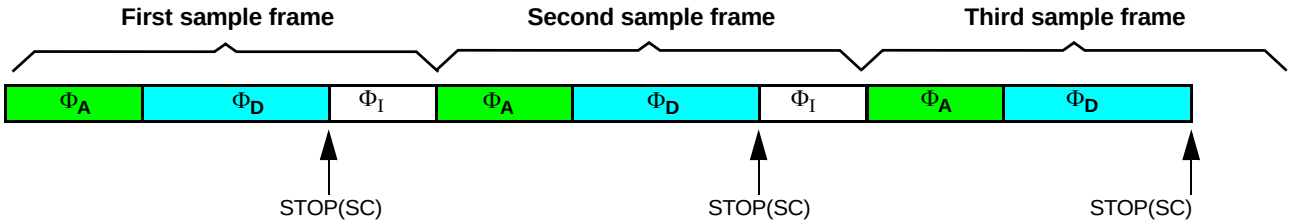


Figure 4-5. Phase triggers required in software

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.

4.4 Clock operation as a function of mode/phase

Figure 4-6 illustrates the nominal phases of operation for this device. The values of Φ_A, Φ_D and Φ_I were discussed briefly in "Frame structure" on page 35. The reset operation is described in "Reset generation" on page 191. The sleep phase is defined as the device oscillator being off and all circuitry in its lowest power state.

"Power control modes of operation" on page 42 maps these phases into modes of operation of the Version 1 Coldfire CPU.

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. Tasks scheduling will be handled by the Scheduler Application (ID 0x01) as described in the *MMA955xL Intelligent, Motion-Sensing Platform Software Reference Manual* (MMA955XLSWRM).

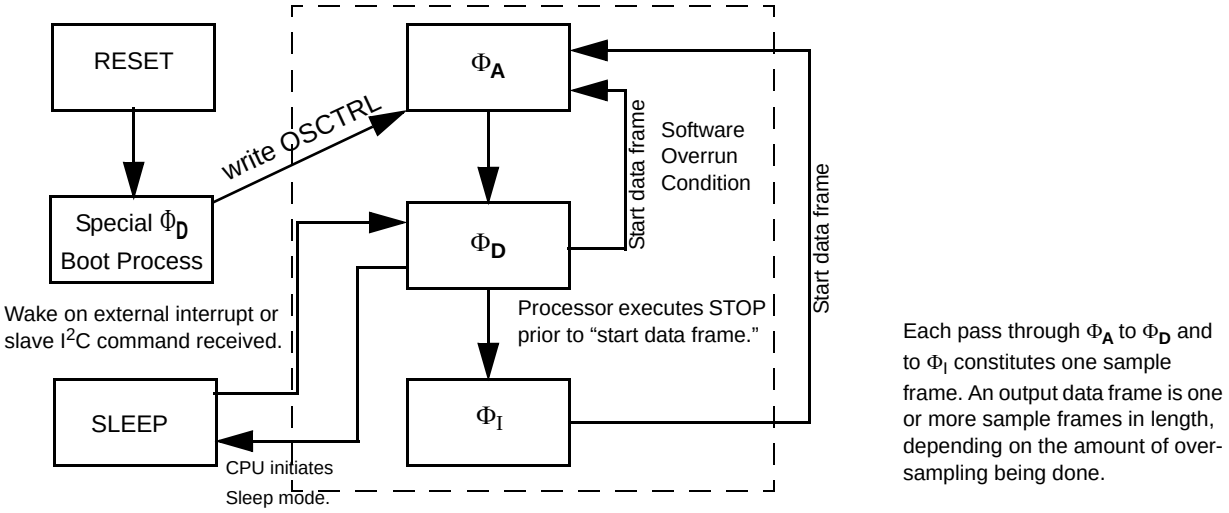


Figure 4-6. Operational phases

The phases shown above have distinct characteristics with regard to clock operation. These are outlined in Table 4-1. 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 4-1. Clock operation per phase

Phase	CPU and standard peripherals		Analog Front End		Slave I^2C
	Normal	**Debug[1]**	**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-peed 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.

4.5 Power control modes of operation

The Version 1 ColdFire architecture incorporates several modes of operation. These include Reset, Run, Stop and Halt (debug). Φ_A, Φ_I and Sleep phases in Figure 4-6 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 4-7 adds HALT mode to the set and remaps the collection as a full-state transition diagram, including debug modes. Table 4-2 summarizes the triggers that cause transitions from one mode to the next.

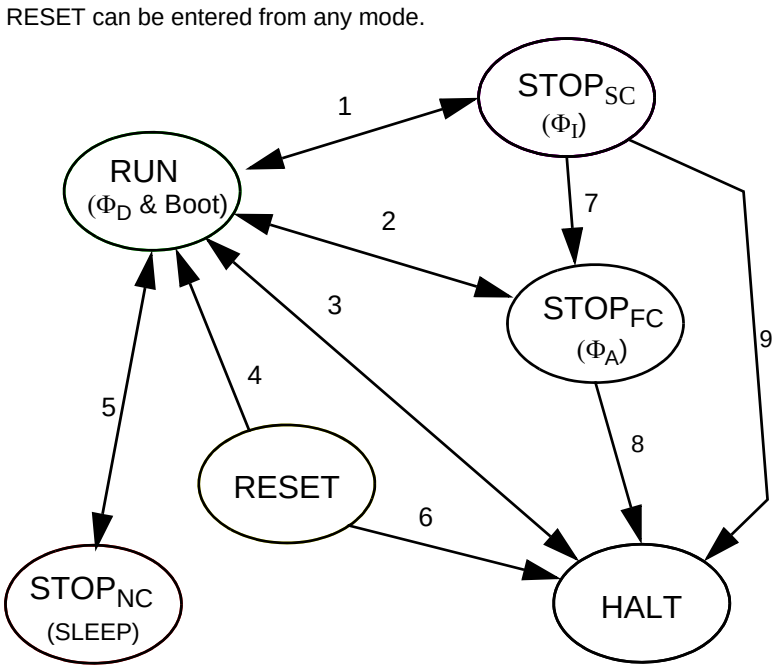


Figure 4-7. Allowable state transitions

Table 4-2. State transitions

Transition #	From	To	Trigger[1]
1	RUN	$STOP_{SC}$	XCSR[ENBDM] = 0, STOPCR[SC] = 1; followed by STOP instruction
	$STOP_{SC}$	RUN	Any interrupt

Table 4-2. State transitions

Transition #	From	To	Trigger[1]
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	De-assert all reset sources. Internal de-assert is subject to timing sequences outlined in “Reset generation” on page 191.
5	RUN	$STOP_{NC}$	XCSR[ENBDM] = 0, STOPCR[NC] = 1, followed by STOP instruction
	$STOP_{NC}$	RUN	Any 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 $STOP_{FC}$.

1 Interrupts are subject to the limitations discussed in “Clock gating” on page 196 and “Peripheral Clock Enable Register 0 for $STOP_{FC}$ Mode” on page 203.

Chapter 5 Memory Maps

5.1 High-level memory map

Table 5-1. V1 ColdFire memory maps

Address range	Generic V1 ColdFire memory usage	Address range	Platform's memory usage
0x(00)00_0000	Allocated to on-chip flash memory	0x(00)00_0000	16-KB flash memory
		0x(00)00_3FFF	
		0x(00)00_4000	Unimplemented
0x(00)2F_FFFF		0x(00)2F_FFFF	
0x(00)30_0000	Allocated to on-chip ROM	0x(00)30_0000	4-KB ROM
0x(00)3F_FFFF		0x(00)30_FFFF	
0x(00)40_0000	Optional off-chip expansion	0x(00)30_0800	Unimplemented
0x(00)7F_FFFF		0x(00)7F_FFFF	
0x(00)80_0000	Allocated to on-chip RAM	0x(00)80_0000	2-KB RAM
0x(00)9F_FFFF		0x(00)80_07FF	
0x(00)A0_0000	Optional off-chip expansion	0x(00)80_0800	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	
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 5-1 is the generic, high-level, memory map applicable to the V1 ColdFire family. Memory map areas shown for RAM, ROM and flash are a superset for the family. Lesser amounts of all three will usually be included on specific devices. The memory map for the MMA955xL platform is shown on the right.

The slave peripherals section of the memory map is further broken down as shown in Table 5-2. MMA955xL 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 5-1 result in an illegal address reset if CPUCR[ARD] = 0 or an address error exception if CPUCR[ARD] = 1.

Table 5-2. 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	SI^2C	8	0x(FF)FF_8000
I^2C	Inter-Integrated IC	MI^2C	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	PT0	8	0x(FF)FF_80E0
Port Control module	Port I/O Control Module 1	PT1	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	N/A[1]
AFE	Analog Front End	AFE	16	N/A[2]
INTC	V1 ColdFire Interrupt Controller	INTC	8	0x(FF)FF_FFC0[3]

1 The FC registers are only available under Superviser mode.

2 The AFE registers are only available under Superviser mode.

3 The INTC_FRC register is the first in the INC memory map, and is located at the base address + $10, or (FF)FF_FFD0.

The lower 32 KB of flash memory (16 KB in the MMA955xL platform) 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).

5.2 Alignment issues

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 the following figure. 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.

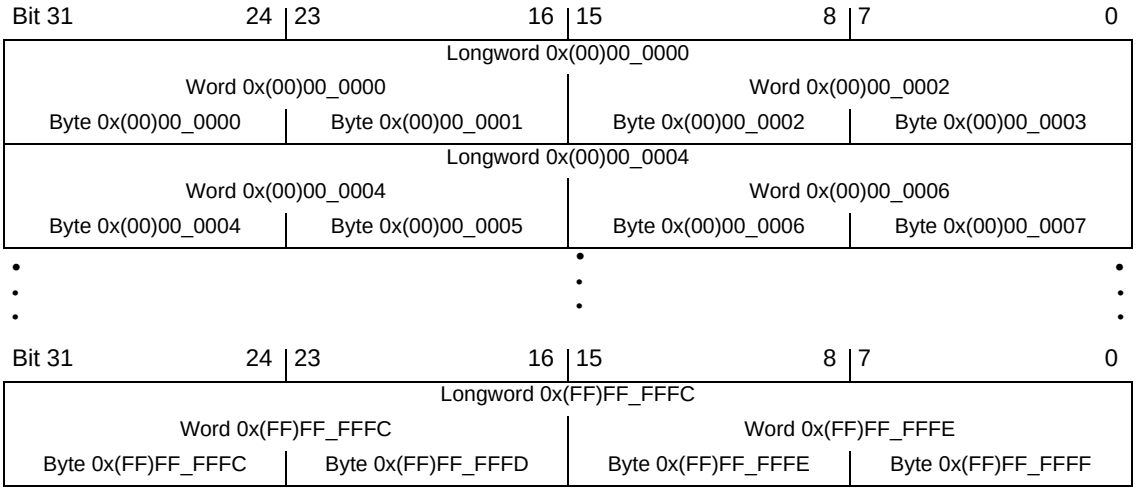


Figure 5-1. ColdFire memory organization

Regions within the memory map are subject to restrictions with regard to the types of CPU accesses allowed. These are outlined in Table 5-3. 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 5-3. V1 ColdFire memory maps

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[1]	x	x	x	x	x	x
0x(FF)FF_EC00	16-bit peripherals[2]	—	x	x	—	x	x

1 Allowed access types are peripheral-specific. The peripheral bus bridge will serialize 16- and 32-bit accesses into multiple 8-bit accesses. When using 8-bit peripherals, care must be taken to ensure that all accesses are properly aligned and only desired 8-bit locations are accessed.

2 Allowed access types are peripheral-specific. The peripheral bus bridge will serialize 32-bit accesses into multiple 16-bit accesses. When using 16-bit peripherals, care must be taken to ensure that all accesses are properly aligned and only desired 16-bit locations are accessed.

5.3 Memory-mapped components

5.3.1 Interrupt controller

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 and 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.

Table 5-15 is a summary of CF1_INTC user-accessible peripheral registers and control bits. Cells that are not associated with named bits are shaded. A shaded cell with a 0 indicates this unused bit is always read as a 0. Shaded cells with dashes indicate unused or reserved bit locations that could be read as 1s or 0s. When writing to these bits, write a 0 unless otherwise specified.

5.3.2 Nonvolatile register area

There is a nonvolatile register area consisting of a block of 4 bytes in flash memory at 0x(00)00_3FFB–0x(00)00_3FFF. The byte at 0x(00)00_3FFF is allocated to flash protection and security functions. Additionally, the byte at 0x(00)00_3FFE is used to initialize boot options for the device. See “Security” on page 69 for further details on both topics.

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

5.3.3 RGPIO

The section of memory at 0x(00)C0_0000 is assigned for use by the ColdFire Rapid GPIO module. See Table 5-4 for the Rapid GPIO memory map and Chapter 15, “Rapid GPIO” for further details on the module.

5.4 Detailed register set

The following tables summarize register-bit fields for on-chip peripherals. For further details, see the chapter on applicable peripheral.

Table 5-4. Rapid GPIO (RGPIO) detailed memory map

Address	Register	Bit 15/7	14/6	13/5	12/4	11/3	10/2	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[15:0] (Write only)							
(00)C0_000E	RGPIO_DATA	DATA[15:8] (Read only)							
		DATA[7:0] (Read only)							

Table 5-5. Slave port detailed memory map

Address	Register	Bit 7	6	5	4	3	2	1	Bit 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							

Table 5-5. Slave port detailed memory map (continued)

Address	Register	Bit 7	6	5	4	3	2	1	Bit 0
(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
(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

Table 5-6. Master I²C (MI²C) detailed memory map

Address	Register	Bit 7	6	5	4	3	2	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

Table 5-7. System Integration Module (SIM) detailed memory map

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

Table 5-8. CLKGEN detailed memory map

Address	Register	Bit 7	6	5	4	3	2	1	Bit 0
(FF)FF_8080	CK_OSCTRL	FCEN	FFCEN	FFSEN	FLE				

Table 5-9. 16-bit Modulo Timer (MTIM) detailed memory map

Address	Register	Bit 7	6	5	4	3	2	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							

Table 5-10. Interrupt (IRQ) pin detailed memory map

Address	Register	Bit 7	6	5	4	3	2	1	Bit 0
(FF)FF_80C0	IRQSC	0	IRQPDD	IRQEDG	IRQPE	IRQF	IRQACK	IRQIE	IRQMOD

Table 5-11. Port Control (PC0) detailed memory map

Address	Register	Bit 7	6	5	4	3	2	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

Table 5-12. Port Control (PC1) detailed memory map

Address	Register	Bit 7	6	5	4	3	2	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

Table 5-13. Two-Channel TPM detailed memory map

Address	Register	Bit 7	6	5	4	3	2	1	Bit 0
(FF)FF_8120	TPM_SC	TOF	TOIE	CPWMS	CLKS		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

Table 5-14. Programmable Delay Block (PDB) detailed memory map

Address	Register	Bit 7	6	5	4	3	2	1	Bit 0
(FF)FF_EC00	PDB_SCR	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]							
(FF)FF_EC06	PDB_MOD	MOD[15:8]							
		MOD[7:0]							
(FF)FF_EC08	PDB_COUNT	COUNT[15:8]							
		COUNT[7:0]							

NOTE

The Flash Controller registers may only be accessed when the CPU is in Supervisor mode.

NOTE

The AFE registers may only be accessed when the CPU is in Supervisor mode.

Table 5-15. Interrupt Controller (INTC) detailed memory map

Address	Register	Bit 7	6	5	4	3	2	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						

5.5 Interrupt vector table

For details of the Interrupt-Controller operation, see Chapter 19, “Interrupt Controller”. Table 5-16 summarizes the default vector map for this device.

Table 5-16. Interrupt vector table

Vector name	Vector numbers	Vector address offset	Interrupt 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 22-6)		
			7	7-5		Reserved on this chip		
irq	64	0x100	7	mid	Next	IRQ	IRQ_IRQSC[IRQIE]	IRQ_IRQSC[IRQF]
frame_err	65	0x104	7	3	Next	SIM Frame Error	SIM_FCSR[SFEIE]	SIM_FCSR[FE]
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_TPM1SC[TOIE]	TPM_TPM1SC[TOF]
tpm1ch0	71	0x11C	6	2	Next	TPM[CH0]	TPM_TPM1C0SC[CH0IE]	TPM_TPM1C0SC[CH0F]
tpm1ch1	72	0x120	6	1	Next	TPM[CH1]	TPM_TPM1C1SC[CH1IE]	TPM_TPM1C1SC[CH1F]
N/A	73	0x124	5	7	Next	Expansion		

Table 5-16. Interrupt vector table (continued)

Vector name	Vector numbers	Vector address offset	Interrupt level	Priority within level	Stacked program counter	Assignment	Interrupt enable	Interrupt source
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		
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]

Table 5-16. Interrupt vector table (continued)

Vector name	Vector numbers	Vector address offset	Interrupt level	Priority within level	Stacked program counter	Assignment	Interrupt enable	Interrupt source
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 I^2C	• Complete 1-byte transfer (TCF) Interrupt • Match of received calling address (IAAS) Interrupt • Arbitration Lost (ARBL) Interrupt • SMBus Timeout (SLTF) Interrupt	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		

Table 5-16. Interrupt vector table (continued)

Vector name	Vector numbers	Vector address offset	Interrupt level	Priority within level	Stacked program counter	Assignment	Interrupt enable	Interrupt source
N/A	115	0x1CC	N/A	N/A	Next	Reserved on this chip		
N/A	...		...	...	Next	Reserved on this chip		
N/A	255	0x3FC	N/A	N/A	Next	Reserved on this chip		

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.

5.6 RAM

This microcontroller includes 2 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 power-on, the contents of RAM are uninitialized. RAM data is unaffected by any reset provided that the supply voltage does not drop below the minimum value for RAM retention (V_{RAM}).

Chapter 6 Flash Memory Controller

6.1 Introduction

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.

6.1.1 Overview

The main flash memory array is intended primarily for program storage. In-circuit programming allows the operating program to be loaded into the flash memory after final assembly of the application product. It is possible to program the entire array through the single-wire, background-debug interface.

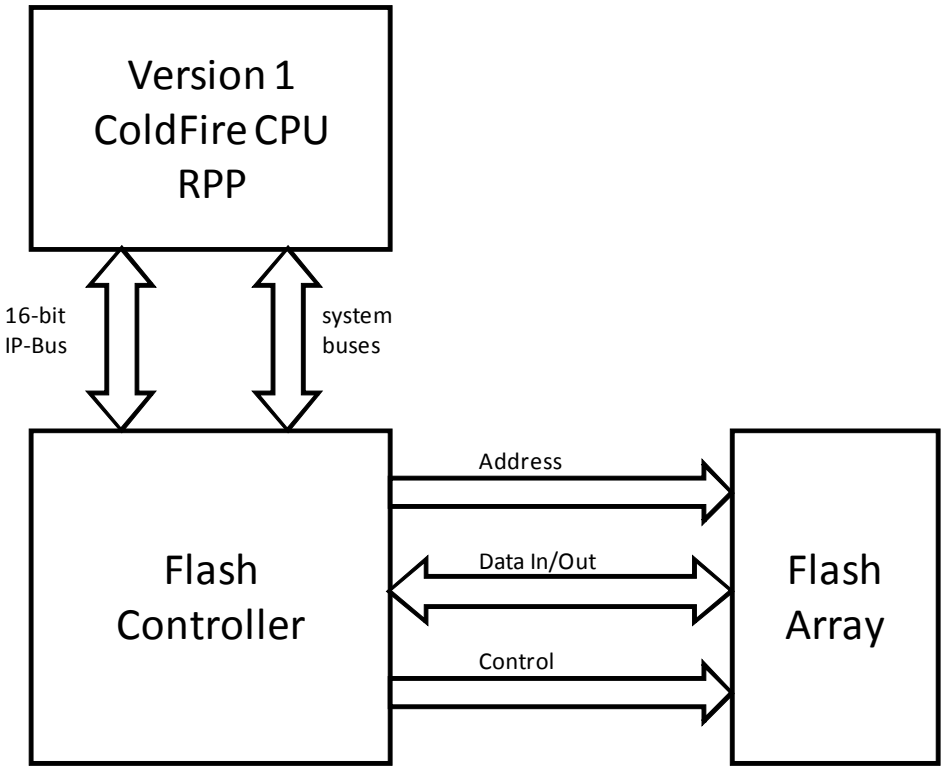


Figure 6-1. Flash-memory block diagram

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.

6.1.2 Features

Features of the on-chip flash memory include:

- 4K-deep by 32-bit main array (16 KB total)
- Page erase size = 512 bytes
- Security lockout
- Protection against accidental programming/erase operations
- Program, erase and mass-erase procedures can be performed using pre-programmed ROM routines.

6.2 Theory of operation

Flash memory is nonvolatile and is ideal for single-supply applications allowing for field reprogramming with no need for 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. Nevertheless 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 in order 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", "RMF_FLASH_PROGRAM" and "RMF_FLASH_ERASE".

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

6.3 Modes of operation

There are four user modes of operation for the flash controller: IDLE, READ, PROGRAM and ERASE. PROGRAM and ERASE modes will only be reached while CPU is in Supervisor state.

6.3.1 Flash IDLE

Whenever the flash is not accessed by the CPU, including during WAIT and STOP modes, it will be in this IDLE mode. The flash module will be in standby and consume minimal power.

6.3.2 Flash 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.

6.3.3 Flash PROGRAM

In this 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.

6.3.4 Flash ERASE

Flash memory can be erased one page (512 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.

6.4 Flash memory maps

The flash module is partitioned into two spaces in memory. The first is the array memory which contains the main flash array. The second area allows supervisor access to module registers and is mapped into the 16-bit, IP-bus space. User access to the flash controller is via dedicated ROM functions. Direct user access to the controller register set is prohibited.

6.4.1 Array memory map

The main flash array is designed to support 16 KB of general program storage. Four bytes of this are reserved for use in storing non-volatile parameters.

Table 6-1. Flash array memory map

Address range	Function
(00) 00_0000 – (00) 00_3FFB 16380 (16K - 4) bytes	General storage
(00) 00_3FFC – (00) 00_3FFF 4 bytes	Reserved for nonvolatile options (4 bytes)

FOPT[7:0] is loaded from address 0x3FFF during each reset sequence.

The boot-to-flash flag (FOPT[FB]) is set to the inverse of Bit 5 of address 0x3FFE during the ROM boot process on power-on-reset. Thus, if Bit 5 of 0x3FFE 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 0x3FFE during the POR boot sequence by the ROM bootloader.

6.5 Flash registers and control bits

The last word of the flash array (at $3FFC) is reserved and should not be used by the application program. The least-significant byte of this location ($3FFF) is referred to as NVOPT. It contains bits that define flash security and write-protection levels.

Table 6-2. Reserved locations in the main array

	0x3FFC	0x3FFD	0x3FFE	0x3FFF
Identifier	CRC[15:8]	CRC[7:0]	NVBOPT	NVOPT
Used for	Expected CRC to be computed over 0x0000 to 0x3FFB		FOPT[15:8] The boot-to-flash flag (FOPT[FB]) is set to the inverse of Bit 5 of address 0x3FFE during the ROM boot process on power-on-reset. FOPT[10:8] are loaded from bits 2:0 of 0x3FFE 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 ($3FFE) 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 0x3FFC.

6.6 Flash memory map/register definition

Flash control registers are not available directly from User mode. Nevertheless, 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 Chapter 7, “ROM”.

Table 6-3. Flash memory map

Address offset	Register	Access	Reset	Details
0x0000	Flash Options Register (FOPT)	RW	0x00FE	page 66

6.6.1 Flash Options register

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.

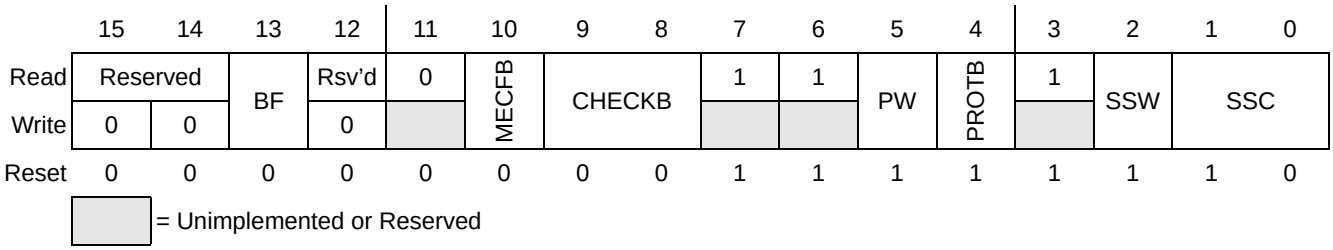


Figure 6-2. Flash Options register (FOPT)

Table 6-4. Flash Options register (FOPT) field descriptions

Bit(s)	Field	Description
15:14	—	Reserved Always write as “00.”
13	BF	Boot from Flash This is a simple R/W bit in the flash controller. **T**his bit is initialized to the inverse of Bit 5 of flash location 0x3FFE 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 Always write as “0.”
11	—	Reserved

Table 6-4. Flash Options register (FOPT) field descriptions (continued)

Bit(s)	Field	Description
10	MECFB	Mass Erase on CRC Failure 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 MECF=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 0x3FFE by the boot ROM on power-up. It is read by the ROM code in a later step of the reset process. 1 Do nothing. 0 Erase the user portion of the main flash array.
9:8	CHECKB	Perform Flash Checksum 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 0x3FFE 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.
7:6	—	Reserved
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
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.
1:0	SSC	Security State Code 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 0x3FFF 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

6.7 Initialization information

6.7.1 Factory

Devices are usually shipped with the lower portion of flash memory pre programmed with a sensor scheduler, trim algorithms and basic sensor functions included. The upper portion of the flash memory is normally shipped in an erased condition.

6.7.2 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 ($3FFF) 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.

In order to facilitate user, flash-area erase and program operations, Freescale will provide with the MMA955xL platform evaluation kit appropriate abstraction tools that will isolate the end user from the ROM routines.

6.8 Programming model

All user access to the flash controller is via Freescale supplied ROM routines which are described in “ROM” on page 71. 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, see “RMF_FLASH_PROGRAM” on page 105.

For details of the ROM function for flash erase, see “RMF_FLASH_ERASE” on page 108.

The user can control the state of the FOPT[PROTB] bit via RMF_FLASH_PROTECT and RMF_FLASH_UNPROTECT. (See “RMF_FLASH_PROTECT and RMF_FLASH_UNPROTECT” on page 111.)

Security can be temporarily suspended via RMF_FLASH_UNSECURE (See “RMF_FLASH_UNSECURE” on page 111.)

6.9 Security

This family of devices include 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 6-5. CPU resources available via BDM In Secure mode

Register	Field	Access	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 the device)**
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 the device)**
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 ($3FFF) are copied from flash into bits 7:0 of the working 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 = 11b) makes the MCU unsecure. When SSC bits of NVOPT are programmed to SECURE (10b), 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 MMA955xL platform to Supervisor mode. The SWI should perform the following functions:

1. If necessary, set PROTB = 1b.
2. Mass-erase the flash and verify that the contents have been erased.
3. Set SSC = 11b, 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 7 ROM

7.1 Introduction

There are several classes of functions stored in ROM:

- A boot program, including ROM-based, slave-port command interpreter
- A collection of utilities that can be invoked via the ROM-based slave port interpreter
- ROM functions that 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 result in a reset of the device.

7.2 Boot ROM

The MMA955xL platform boots from a standard routine in ROM. This boot function (shown in Figure 7-1) 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 7-1 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 0x3FFE and 0x3FFF. 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, RAM or the 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.

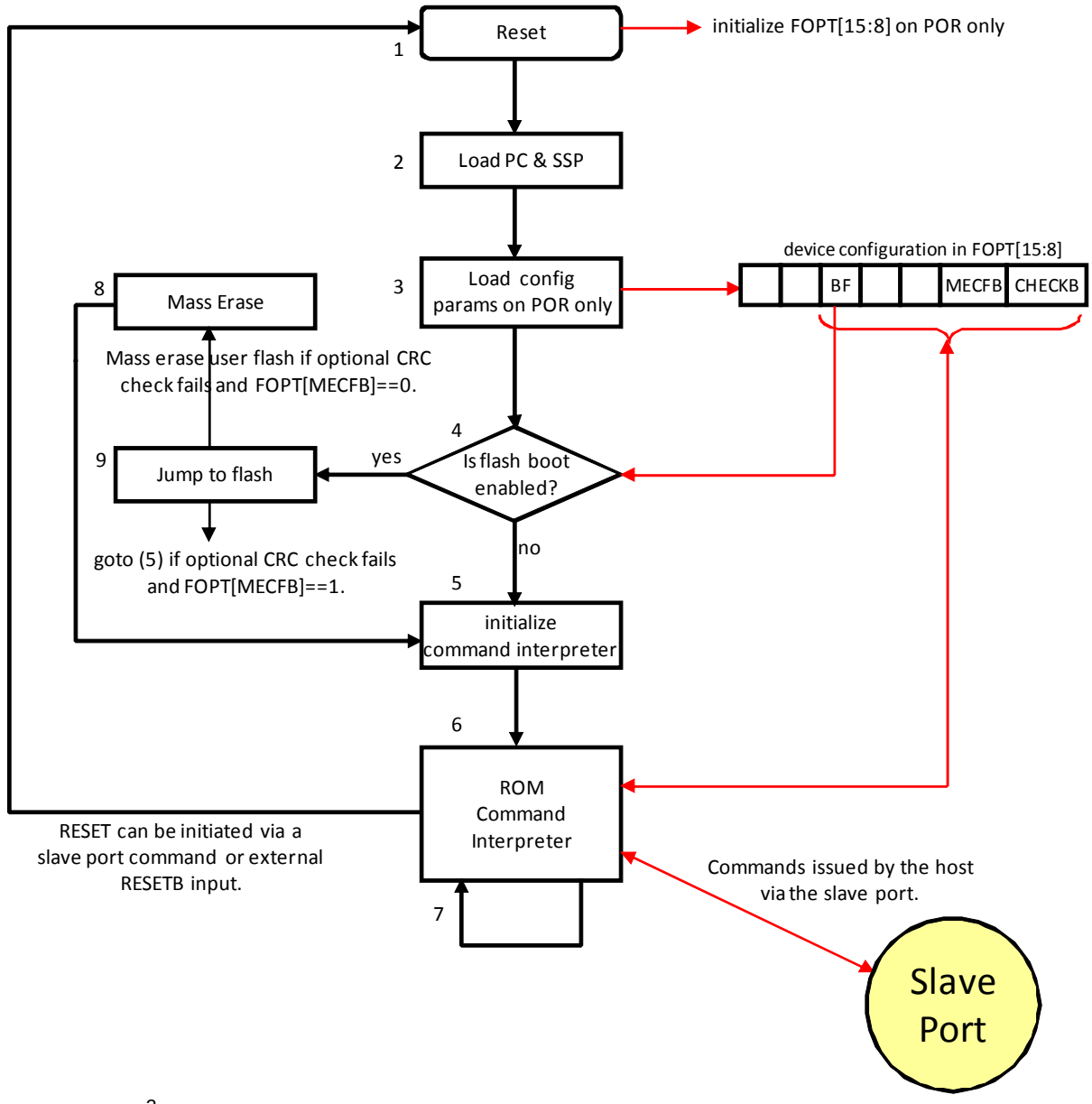
Because FOPT[15:8] is initialized only at power-up, it can be manipulated by the slave-port command interpreter and BDM to reconfigure device operation on subsequent reset operations.

For fielded applications, the normal control flow for the boot function is 1-2-3-4-9. (See Figure 7-1.) Other options are intended primarily for debug and development purposes.

7.2.1 Boot Step 1: RESET

Any hardware- or software-initiated reset will return the device to this phase. Hardware logic on the chip is returned to its default state.

During this phase, the FOPT[7:0] (which includes the device's security state) is reloaded from location 0x3FFF in flash memory. If the reset is a result of a power-on sequence, FOPT[15:8] will be initialized to all 0s. These register bits are not affected by subsequent reset operations. They are used to coordinate boot and flash operations across reset sequences.



The choice of I^2C or SPI communication is determined by the state of the SSB pin during the boot process. Low = SPI, High = I^2C.

Figure 7-1. Flow diagram for ROM boot routine

7.2.2 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 and start-up code initializes the status register to 0x2700 and sets the Vector Base Register (VBR) to point to the beginning of ROM (0x300000).

7.2.3 Boot Step 3: Load configuration parameters

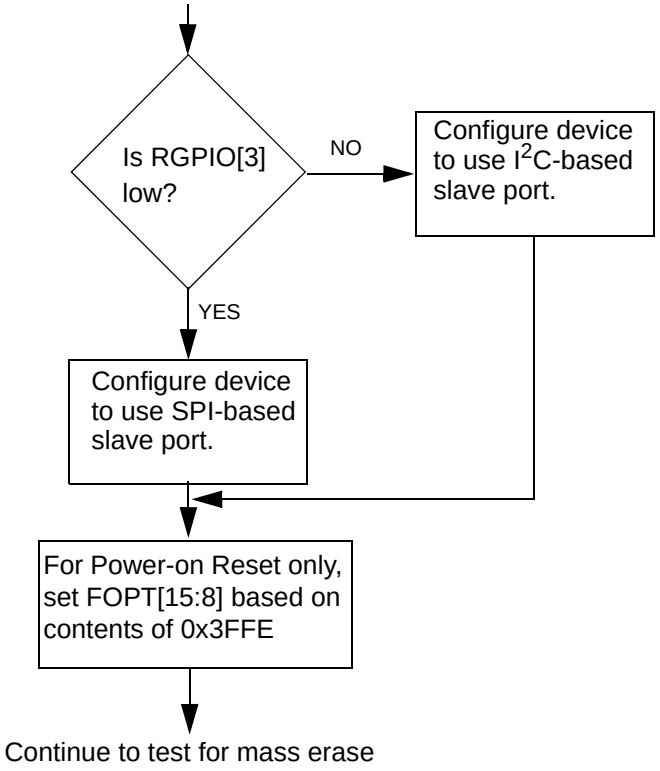


Figure 7-2. Boot step 3: Load configuration parameters

Subsequent to reset, configuration parameters are read from reserved locations in flash and are stored in specific fields of control registers in the memory map.

For power-up sequences only:

- FOPT[BF] is set to the inverse of Bit 5 of memory location 0x3FFE 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 0x3FFFE in flash.

7.2.4 Boot steps 4 and 9: For flash boots, jump to flash

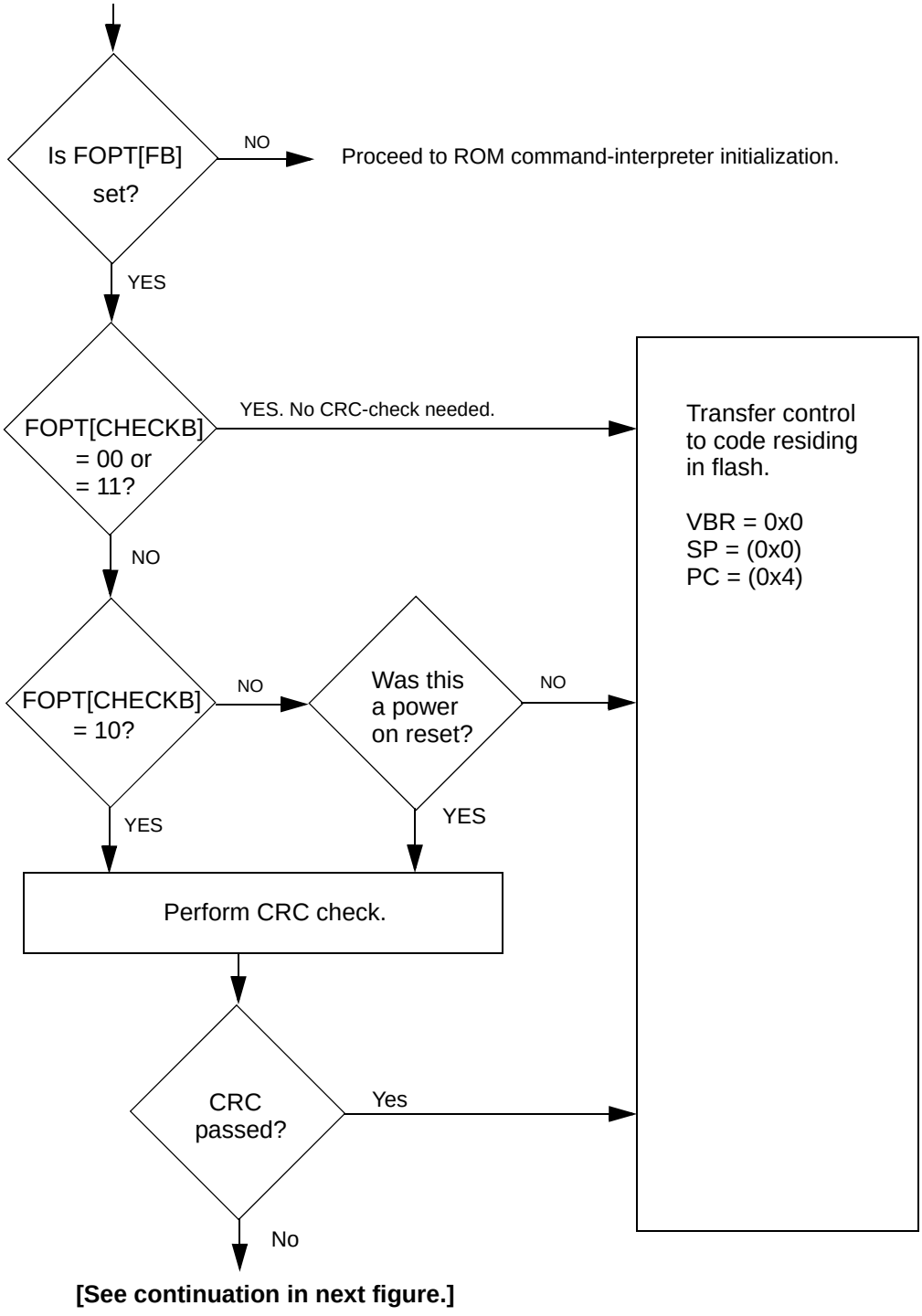


Figure 7-3. Boot-to-flash and associated checks (Part 1)

If FOPT[BF][1] has been set, 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 a check is run and succeeds, 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_0004. The ColdFire Vector Base Register (VBR) is reset to 0x(00)00_0000. If FOPT[FB] has not been set, control is transferred to Step 5 (Initialize Command Interpreter). If the CRC check fails, and FOPT[MECFB] is set, the device will be subjected to Mass Erase of User Portion Flash. Control then is transferred to Step 5 (Initialize Command Interpreter). For more details, see “Boot Step 5: 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

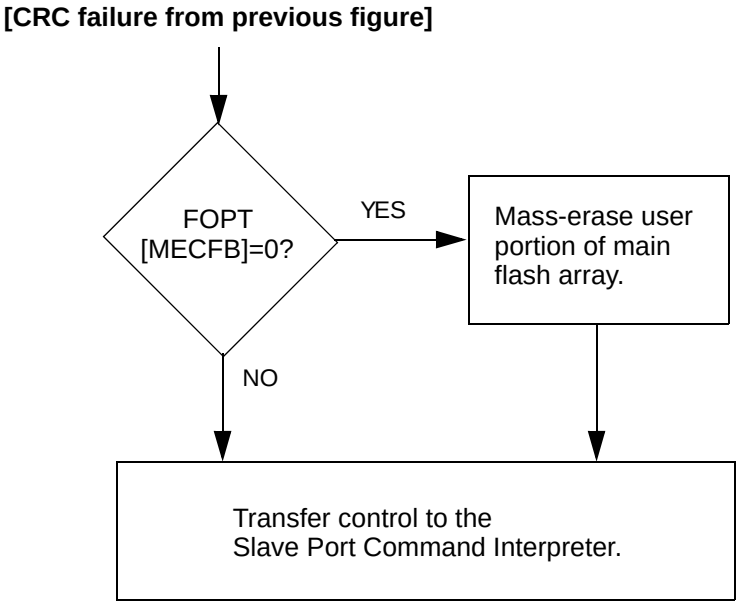


Figure 7-4. Boot-to-flash and associated checks (Part 2)

7.2.5 Boot Step 5: 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.

1. This was initialized in Step 3 to the inverse of Bit 5 of flash location 0x(00)00_3FFE.

7.2.6 Boot Step 6: 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. If, however, entered commands include a reset command, the state machine restarts at Step 1.

Commands are submitted and statuses returned via the slave-port mailbox registers. Related details are provided in “ROM Command Interpreter”.

RGPIO3/SDA1/SSB = LOW at start-up indicates that the SPI should be used as slave instead of I^2C. This is a function of the application boot code, not of the hardware. The boot routine needs to read the RGPIO3 input value and act accordingly.

7.3 Security and rights management

7.3.1 Access and security rules of thumb

- PROTB protects against accidental programming/erasures by software running on-chip. It does not prevent mass-erase via BDM or slave port CI.
- The Page Release Register (PRR) allocates the pages of the flash array to be used by Freescale code and the user application code. (See “Page-Release Register”.) Pages assigned to Freescale are protected from accidental erasure and can only be erased under tightly controlled conditions.
- Mass-erase operational requests supply a mask parameter of 0xFFFFFFFF.
- The following resources are restricted to use in Supervisor mode:
  - ROM code
  - AFE registers
  - Flash-controller registers
- Asserting security shuts down almost all access via the BDM and slave ports. The only supported operations in secure state are RESET, MASS ERASE and GET DEVICE INFO.

7.3.2 Security

Users may secure their 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. In addition, ROM-based, slave-port access is severely restricted.

Security may be cleared by mass-erasing the device. This can be done via BDM by setting XCSR[ERASE][1] and resetting the device. The ROM boot code will then erase all application pages (PRR = 1)[2] in flash memory, regardless of the setting of the flash-protection bit (FOPT[PROTB])[3].

Security may also be cleared by mass-erasing the part 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[4].

If an attempt is made to read/write any on-chip memory while the device is in a secure state, the ROM-based, slave-port functions will fail and return a security violation.

1. “Extended Configuration/Status Register” on page 357.
2. “Page-Release Register” on page 78.
3. “Flash Options register” on page 66.
4. “RMF_FLASH_PROTECT and RMF_FLASH_UNPROTECT” on page 111

7.4 Rights management

7.4.1 Memory-map restrictions

This section describes generic techniques for managing user access to restricted functions.

The MMA955xL 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 7-1. NVM memory allocations

Memory	Size	Freescale	Third-Party	Usage
ROM	4 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	16 KB	X	X	There are 32 512-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.

7.4.2 Rights-management variables

Non-volatile parameters used for rights management are shown in Table 7-2.

Table 7-2. 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]			
PRR	Page Release Register[1] (Factory Settings)	PE[31:0]			

1 Not available in User mode.

7.4.2.1 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 field will vary from device to device.

7.4.2.2 Page-Release Register

As previously mentioned, the main flash array on this device has 32 pages of 512 bytes each. User programming/erase access to these pages is controlled via a virtual “Page-Release Register” (PRR). 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 set to “0”, the page is allocated for Freescale use and will not be made available for customer programming. If set to “1”, the page is available

for customer use. Bit 0 corresponds to the page beginning at address 0x(00)00_0000. Bit 31 corresponds to the page beginning at 0x(00)00_3E00.

7.4.2.3 Hardware restrictions

The flash memory controller contains a non-volatile bit (FOPT[PROTB]) that can be used to protect flash memory from accidental programming/erase operations.

This bit is sourced from the NVOPT location in flash memory on reset. It can be temporarily switched in and out via software. Various mechanism for manipulating this value are described in the descriptions of the flash-access functions, later in this chapter.

7.5 ROM Command Interpreter

7.5.1 Callable utilities

Functions available via the ROM Command Interpreter are summarized in Table 7-3. For a general overview of the user model associated with these functions, see “Packet transfers and commands overview” on page 81. 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, however, you must unprotect flash memory.

Table 7-3. Functions callable via ROM Interpreter[1]

Command	Description	Five-bit command code	Secure mode operation	Details
CI_DEV_INFO	Return device information	0x00	Allowed	page 83
CI_READ_WRITE	Read/write memory (including flash programming)	0x01	Operation not performed. Security violation returned.	page 85
CI_ERASE	Erase flash memory (page and mass-erase)	0x02	Mass-erase only	page 90
CI_CRC	Calculate CRC over memory range	0x04	Operation not performed. Security violation returned.	page 94
CI_RESET	RESET	0x05	Allowed	page 97
CI_PROTECT	Protect flash memory	0x07	Allowed	page 99
CI_UNPROTECT	Unprotect flash memory	0x08	Allowed	page 99

[1]All other command codes return the RMF_ERROR_COMMAND code (bad command).

7.5.2 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 transfer is the response packet 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 MMA955xL platform 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
NAK	Not acknowledge bit
P	Stop bit

In the “example” tables, later in this chapter, green-shaded table cells indicate the bits written by the slave. Unshaded bits are written by the master. Gray-shaded entries are non-existent, for formatting purposes only. Heavy borders around a table cell indicate those bits in the sequence that map to specific mailbox locations.

7.5.3 Common error codes

All CI response packets utilize the same set of common return codes in the most-significant nibble of Mailbox 1. Bit 8 is used as “Command Complete” or “COCO.” It is set to 0 when the command interpreter first recognizes the incoming command, then 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 7-4. 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 supplied in CI read/write packet and 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	Most CI commands are unavailable when security has been set (FOPT[SSC] = 10). This error code will be returned when an attempt has been made to execute a prohibited function.
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	This code is returned when the command interpreter does not recognize a command code or an incomplete packet is recognized.

7.5.4 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.

7.5.4.1 CI_DEV_INFO command-packet format

The five-bit command code for the device-info command is 0x00.The extension bits are 0.

Table 7-5. 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							

7.5.4.2 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, not an error, as other packet information will be correct, regardless of 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)

Table 7-6. CI_DEF_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							

Table 7-6. CI_DEF_INFO response packet format at mailbox level (continued)

Mailbox #	Description	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
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							
12	0xFF	1	1	1	1	1	1	1	1
13	RESERVED	1	1	1	1	1	1	1	1
14-31	NOT USED	NOT USED							

7.5.4.3 CI_DEV_INFO access/security policies

Table 7-7 details security policies for the CI Return Device Info command.

Table 7-7. Access/security policies for CI return device info command

Security enabled	Security disabled
Available	Available

7.5.5 CI_READ_WRITE

7.5.5.1 Description

This function encapsulates all memory read/write functions, including those required for programming flash memory. Please note that flash memory must be erased prior to any program operation.

Memory mapped components, RAM, ROM and flash memory can be read/written with a common set of memory-access sequences. Read commands require eight mailbox locations. Write commands also require eight 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 which 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 two mailboxes plus the payload in length. Write response packets consume four mailbox values.

7.5.5.2 CI_READ_WRITE Read/Write memory command-packet format

The five-bit command code for the read/write command is 0x01.

Table 7-8. CI_READ_WRITE 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]							
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[1]	WDATA							

1 Not applicable to Read Operations

Table 7-9. CI_READ_WRITE 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 Values other than the following are reserved. 000 Flash memory 001 ROM (Valid CI_PW match required.) 010 RAM 011 RGPIO 100 8-bit peripherals 101 16-bit peripheral (Valid CI_PW match required.)
NUMBER	Number Values other than the following are reserved. Number of bytes to read/write. 0 NO-OP 1 to 28 for writes 1 to 30 for reads
CI_PW	Command Interpreter Password Certain restricted functions require a Freescale-supplied password to unlock access. The value of this parameter is ignored for non-restricted functions. “CI_READ_WRITE access/security policies” on page 88 for details.
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.

7.5.5.3 CI_READ_WRITE Read/Write memory response-packet format

There are two slightly different forms of the response packet. 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 7-10. CI_READ_WRITE 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[1]	RDATA							

[1] Not applicable to Write functions

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 7-11. CI_READ_WRITE 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 7-12. CI_READ_WRITE command response field descriptions

Field	Description
VERF	Verify Writes (not applicable in Read accesses) If a verify error is found, the address at which the first error is detected will be written to mailboxes 2 and 3. 0 Do not verify 1 Verify that written value matches intent.
TYPE	Type of Access 0 Write 1 Read

Table 7-12. CI_READ_WRITE command response field descriptions

Field	Description
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. 1 Previous command has been completed or aborted. (The ERR flag will be set for aborted sequences.) 0 Previous command not completed.
ERR	Error Flag For the set of common CI error codes, see Table 7-4 on page 82.
RDATA	Read Data If ERR = 000, this is the NUMBER of bytes of data transferred in read command. If ERR is any other value, the data contained in these bytes is not guaranteed.
VERF_ADDR	Verify Address[15:0] For write operations with verify, this is the lower 16 bits of the first location in which a verify error was detected.

7.5.5.4 CI_READ_WRITE access/security policies

Table 7-13 details security policies for the CI Read/Write command.

Table 7-13. 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 1b. Flash protection must be disabled prior to any attempt at programming.

CI_PW match required — A valid command interpreter password must be supplied.

7.5.5.5 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 I^2C slave port, mapped to location 0x03 on the I^2C bus

The Read packet must write four mailbox registers in the slave port.

Table 7-14. Command to read four bytes from RAM starting at offset 0x08 for Device 4C on I²C bus

Start/Stop	7	6	5	4	3	2	1	0	
S	Slave address = 0x4C							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 “Message format for reading the platform” on page 146. 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 7-15. Response to Previous Read Command on I²C bus

Start/Stop	7	6	5	4	3	2	1	0	
S	Slave address = 0x4C							R/W = 0	A
	Register address= 0x00								A
S	Slave address = 0x4C							R/W = 1	A
	Mailbox 0 = Read command = 0x09								A
	Mailbox 1 = Status = 0x84 (command complete, read 4 bytes)								A
	Mailbox 2 = Data Byte from RAM Location (00)80_0008								A
	Mailbox 3 = Data Byte from RAM Location (00)80_0009								A
	Mailbox 4 = Data Byte from RAM Location (00)80_000A								A
	Mailbox 5 = Data byte from RAM location (00)80_000B								NACK
P									

7.5.6 CI_ERASE

7.5.6.1 Erase-flash function description

This function encapsulates all functions for page- and mass-erase actions of the flash memory.

The command packet is six mailboxes in length. The response packets are two to four 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.

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, page-erase when not.

7.5.6.2 Erase-command packet format

The five-bit command code for the erase command is 0x02.

Table 7-16. 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	MASK	MASK[31:24]							
3		MASK [23:16]							
4		MASK [15:8]							
5		MASK [7:0]							
6 - 31	NOT USED	NOT USED							

Table 7-17. CI_ERASE command field descriptions

Field	Description
VERF	Verify Erase If a verify error is found, the address at which the first error is detected will be written to mailboxes 2 and 3. 0 Do not verify 1 Verify that written value matches intent.
PB	PB Constant value = 0xC5 Values other than 0xC5 will trigger a security error.

Table 7-17. CI_ERASE command field descriptions

Field	Description
MASK	**Page Mask** The main flash array on this device is composed of 32 512-byte pages of memory. A page is the minimum amount of flash memory that can be erased in a single operation. The 32 bits of the mask variable correspond to pages 0 through 31. Page MASK[0] corresponds to the page starting at 0x(00)00_0000. MASK[31] corresponds to the page starting at 0x(00)00_3E00. For each page, these bits have the following function: 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. “Page-Release Register” on page 78 for additional details. The Page Mask must be set to 0xFFFFFFFF for mass-erase requests. Other values will result in security violations if the device is secured. It is necessary to unprotect flash memory[1] before attempting to erase it. 0 Do not erase. 1 Erase requested.

1 “RMF_FLASH_PROTECT and RMF_FLASH_UNPROTECT” on page 111.

7.5.6.3 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 7-18. CI_ERASE 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[15:8]							
3	Verify error addr LSB	VERF_ERR_ADDR[7:0]							
4 - 31	Not Used	Not Used							

Table 7-19. CI_ERASE command field descriptions

Field	Description
VERF	Verify Erase If a verify error is found, the address at which the first error is detected will be written to mailboxes 2 and 3. 0 Do not verify. 1 Verify that written value matches intent.
COCO	Command Complete 0 Previous command 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 been completed or aborted. (The ERR flag will be set for aborted sequences.)

Table 7-19. CI_ERASE command field descriptions (continued)

Field	Description
ERR	Error Flag See Table 7-4 on page 82 for the set of common CI error codes. Of those, the following error code interpretations apply to this device: 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_PROT – FOPT[PROTB] needs to be reset to 1 before erase.
VERF_ADDR	Verify Address[15:0] This is the lower 16 bits 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.

7.5.6.4 CI_ERASE access/security policies

Table 7-20 details security policies for the CI Erase command.

Table 7-20. 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[1]	Not supported	Erased when (PB = 0xC5 and Mask=0xFFFFFFFF)	Subject to PRR	Erased when (Mask=0xFFFFFFFF)

1 The PRR bits for the upper portion of flash are all ones. This section is available for application use.

Policy descriptions are:

Subject to PRR — Erasures of flash memory are restricted to those in which the PRR [page number] bit is 1b. The flash protection must be disabled prior to any attempts to page-erase.

7.5.6.5 Erase example

This example performs a mass-erase of the upper portion of the main array in flash memory.

The command packet must write six mailbox registers in the slave port.

Table 7-21. Command to mass-erase flash on Device 4C on I²C bus

Start/Stop	7	6	5	4	3	2	1	0	
S	Slave address = 0x4C							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 = 0xFF								A
	Mailbox 3 = 0xFF								A
	Mailbox 4 = 0xFF								A
	Mailbox 5 = 0xFF								A
P									

The response packet uses the I²C "combined format" that is described in "Message format for reading the platform" on page 146. 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. Table 8-10 shows the case where only status information was retrieved. Diagnostic information in mailbox 2 and 3 was ignored.

Table 7-22. Response to previous mass-erase command on I²C bus

Start/Stop	7	6	5	4	3	2	1	0	
S	Slave address = 0x04C							R/W = 0	A
	Register address= 0x00								A
S	Slave address = 0x4C							R/W = 1	A
	Mailbox 0 = Mass erase main array only command = 0x12								A
	Mailbox 1 = Status = 0x80 (command complete, no errors)								NACK
P									

7.5.7 CI_CRC

CodeWarrior has the ability to calculate a CRC over a range of code and include it as part of the flash or ROM image. This function replicates the same algorithm, which can be used to confirm code integrity over time.

The CRC function will fail with a security violation if the device has security enabled.

7.5.7.1 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 7-23.

Table 7-23. 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]							
7	Ending offset [7:0]	OFFH[7:0]							
8 - 31	NOT USED	NOT USED							

Table 7-24. CI_CRC command field descriptions

Field	Description
MEM	Memory Space All values other than the following are reserved. 000 Flash memory 001 ROM 010 RAM
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.

Table 7-24. CI_CRC command field descriptions

Field	Description
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.

7.5.7.2 CRC response-packet format

The response packet for the CRC calculation has a length of four mailboxes. These include:

- The first byte of the response packet, that 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 7-25. CI_CRC 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 7-26. CI_CRC command field descriptions

Field	Description
COCO	Command Complete 0 Previous command 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 been completed or aborted. (The ERR flag will be set for aborted sequences.)
ERR	Error Flag For the set of common CI error codes, see Table 7-4 on page 82.
SIG	Signature[15:0] 16-bit signature calculated by the CRC function.

7.5.7.3 CI_CRC access/security policies

Table 7-27 details security policies for the CI CRC command.

Table 7-27. Access/security policies for CI CRC command

Security enabled	Security disabled
Not Available	Available

7.5.7.4 CRC example

This example calculates a CRC across the entire range of the ROM.

The command packet must write eight mailbox registers in the slave port.

Table 7-28. CI_CRC I²C command packet to calculate the ROM CRC

Start/Stop	7	6	5	4	3	2	1	0	
S	Slave address = 0x4C							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								
	Mailbox 3 = LSB of Seed = 0x0F								
	Mailbox 4 = 0x00								
	Mailbox 5 = 0x00								
	Mailbox 6 = 0x10								
	Mailbox 7 = 0x00								
P									

The minimum response packet uses the I²C "combined format", which is described in "Message format for reading the platform" on page 146. ("Slave Port Interface" on page 115.) 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 7-29. Response to Previous Read Command on I²C bus

Start/Stop	7	6	5	4	3	2	1	0	
S	Slave address = 0x4C							R/W = 0	A
	Register address= 0x00								A
S	Slave address = 0x4C							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									

7.5.8 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 cases of an error, the "standard," two-mailbox response packet is generated.

7.5.8.1 CI_RESET command-packet format

The command packet requires two mailboxes.

The five-bit command code for the reset command is 0x05.

Table 7-30. 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 7-31. 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 at what address the device boots on reset.

Table 7-32. Reset boot options

FL	Memory	Base Address
0	ROM	0x(00) 30_0000
1	Flash	0x(00) 00_0000

7.5.8.2 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. Otherwise the device resets itself.

Table 7-33. 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 7-34. CI_RESET response-packet field descriptions

Field	Description
COCO	Command complete 0 Command not complete 1 Command complete
ERR	Error Flag For the set of common CI error codes, see Table 7-4 on page 82.

7.5.8.3 CI_RESET access/security policies

Table 7-35 details security policies for the CI_RESET command.

Table 7-35. Access/security policies for CI_RESET command

Security Enabled	Security Disabled
Available	Available

7.5.9 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 this bit prior to undertaking any changes to the flash array. If FOPT[PROTB] = 0, 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.

7.5.9.1 CI_PROTECT command-packet format

The five-bit command code for the CI_PROTECT command is 0x07 The extension bits are 0.

Table 7-36. 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							

7.5.9.2 CI_UNPROTECT command-packet format

The five-bit command code for the CI_UNPROTECT command is 0x08. The extension bits are 0.

Table 7-37. 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							

7.5.9.3 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. COCO is set to 1 when the command response is complete.

7.5.9.4 CI_PROTECT and CI_UNPROTECT access/security policies

Table 7-38 details security policies for the CI _PROTECT and CI_UNPROTECT commands.

Table 7-38. Access/Security Policies for CI Return Device Info Command

Security enabled	Security disabled
Available	Available

7.6 User-callable ROM functions

The primary function of the MMA955xL ROM is to provide a repository 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 calling hierarchy is illustrated in Figure 7-5.

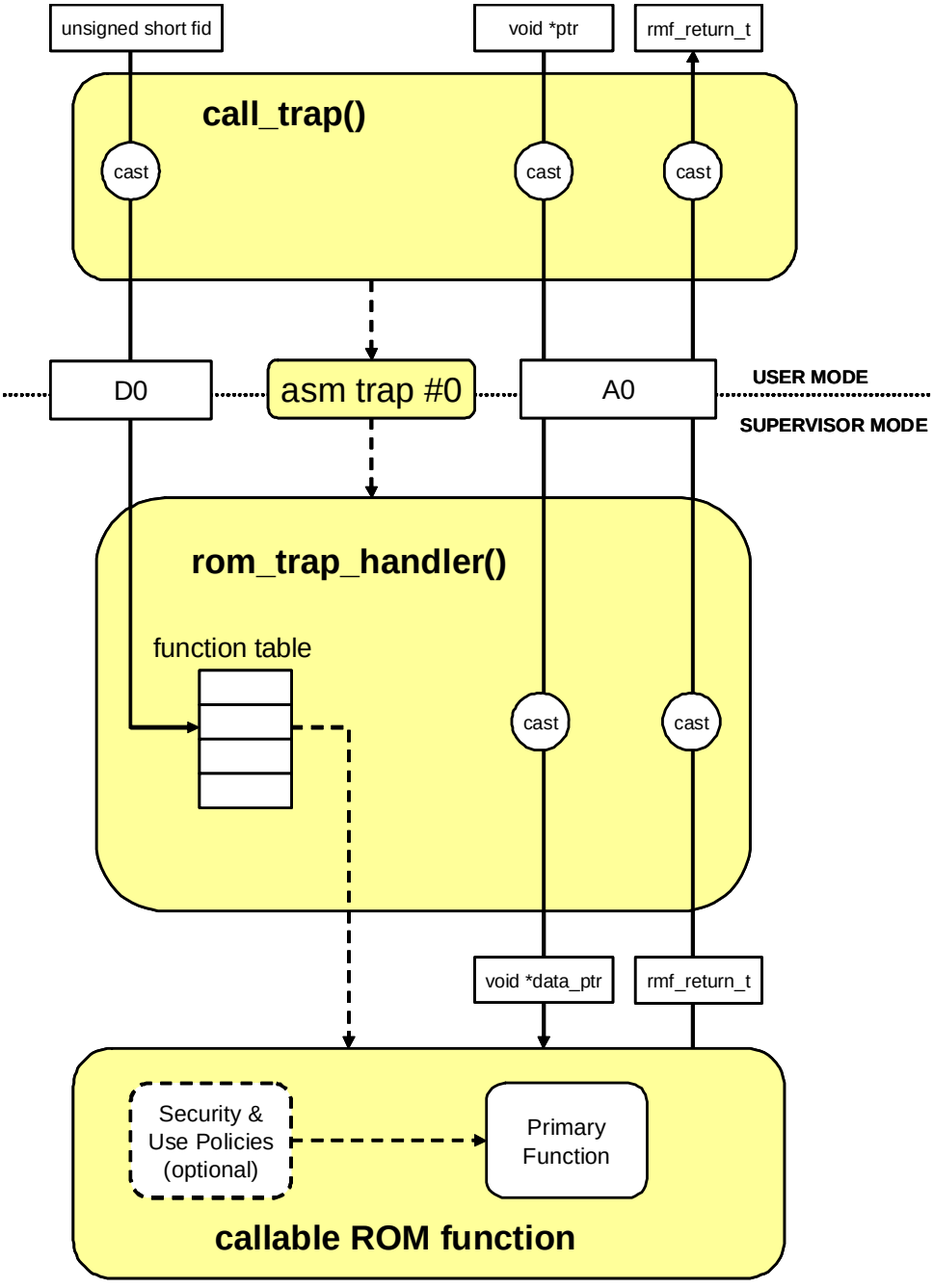


Figure 7-5. Call hierarchy for ROM functions

All user-callable ROM functions are invoked via the call_trap() function. The C-prototype for this function is:

Example 7-1. Invoking user-callable ROM functions

```
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, the user will need to specify varName.ptr or varName.val, depending on the data type into which the user needs 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 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 7-2. 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 are defined in Table 7-39.

Table 7-39. ROM Functions callable via call_trap()

Function ID	Description	Input/Output Structures	Details
RMF_DEV_INFO	Retrieve 32-bit device identifier	In: NULL Out: rmf_design_info_t	page 104
RMF_FLASH_PROGRAM	Program flash	In: rmf_flash_prog_params_t Out: rmf_flash_op_sts_t	page 105
RMF_FLASH_ERASE	Erase flash	In: rmf_flash_erase_params_t Out: rmf_flash_op_sts_t	page 108
RMF_CRC	Calculate a checksum over a range of memory.	In: rmf_crc_params_t Out: rmf_crc_sts_t	page 112 and page 113
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	page 111
RMF_FLASH_UNPROTECT	Enable program/erase.	In: NULL Out: NULL	page 111
RMF_FLASH_UNSECURE	Temporarily unsecure the device.	In: NULL Out: NULL	page 111

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. However, the function of the COCO bit is different. For RMF calls, COCO = 0 indicates that an error occurred. COCO = 1 indicates that the function completed properly.

Table 7-40. Common RMF error codes

Error name	Error = Bits 6:4	Description
RMF_ERROR_PARAM	001	An input parameter did not pass muster. Examples include: incorrect MEM field supplied in CI read/write packet and 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). Call the CI function to unprotect flash before any attempt to program/erase the flash.
RMF_ERROR_SECURITY	011	Most CI commands are unavailable when security has been set (FOPT[SSC]=10). This 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 mailboxes 2 and 3. This error only occurs when the VERF 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 which attempted to program flash memory across physical rows of the device.
RMF_ERROR_COMMAND	111	This code is returned anytime that the command interpreter does not recognize a command code or when an incomplete packet is recognized.

7.6.1 RMF_GET_DEVICE_INFO

7.6.1.1 Description

This function returns the 32-bit device ID (DID). The device ID is a part of the “rights management” system described in “Security and rights management” on page 77. In addition to the device ID, this function also returns hardware and software version numbers.

Typically, a master controller will request this information of the MMA95XX as the first step in initiating an upgrade.

7.6.1.2 RMF_GET_DEVICE_INFO structure syntax

No parameters are necessary. Supply a NULL structure pointer to the call_trap() function when invoking this function.

Example 7-3. Output structure syntax

```
typedef struct {
   unsigned long id;
   char rom_major;
   char rom_minor;
   char ft_flash_major;
   char ft_flash_minor;
   char hw_major;
   char hw_minor;
} rmf_device_info_t;
```

7.6.1.3 RMF_GET_DEVICE_INFO error codes

This function always succeeds. There are no error codes.

7.6.1.4 RMF_GET_DEVICE_INFO operation

This function simply returns a number of values stored within the flash-information row.

Table 7-41. Return parameters for RMF_GET_DEVICE_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

7.6.1.5 RMF_GET_DEVICE_INFO access/security policies

This function may be called at any time.

Example 7-4. RMF_GET_DEVICE_INFO

```
rmf_device_info_t device_data;
device_data =  (call_trap(RMF_GET_NFO, NULL)).val;
```

7.6.2 RMF_FLASH_PROGRAM

7.6.2.1 Description

All user access to the flash-controller programming functions is via this function. Interrupts are disabled when this function executes and STOP-mode operation is temporarily disabled. System clocks will remain in their high-speed states (8 MHz) during program operations.

Because flash operations interfere with STOP-mode operation, their use is not consistent with normal sensor operation as described in Chapter 4, “Operational Phases and Modes of Operation”. **Frame operation will need to be suspended during use of this function.**

NOTE

The flash-controller hardware is not accessible outside of Supervisor mode. Accessing flash-controller hardware by a Supervisor-mode method other than this function (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.

7.6.2.2 RMF_FLASH_PROGRAM input structure

Example 7-5. 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;
```

Table 7-42. rmf_flash_prog_params_t input 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 other value than RMF_ENABLE_FLASH_PROGRAM[1] is used as the value for this parameter, the function will return with a failed status code.
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 128 bytes or less and fit within a single row of the flash array. The value of num_lword should not exceed 32, 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.
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.

1 Defined in rom_functions.h for this device. Required header files will be provided and described in a specific Application Note.

7.6.2.3 RMF_FLASH_PROGRAM output structure

Example 7-6. 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;
```

Table 7-43. rmf_flash_prog_params_t output 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 7-40 for possible values.
first_error	Address of first error found in any verify operations.

7.6.2.4 RMF_FLASH_PROGRAM access/security policies

Flash program operations are allowed only on those pages in which the associated PRR[1] bit is 1.

The following 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[2].

Example 7-7. Function access/security

```
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
}
```

1. “Page-Release Register” on page 78 for additional details.
2. Protection status only survives until the next reset, at which time it is reloaded from the NVOPT byte.

7.6.3 RMF_FLASH_ERASE

7.6.3.1 Description

All user access to the flash-controller erase functions (both page-erase and mass-erase) is via this function. Interrupts are disabled when this function executes and STOP-mode operation is temporarily disabled. System clocks will remain in their high speed states (8 MHz) during erase operations.

Because flash operations interfere with STOP-mode operation, their use is not consistent with normal sensor operation as described in Chapter 4, “Operational Phases and Modes of Operation”. Frame operation will need to be suspended during use of this function.

NOTE

The flash-controller hardware is not accessible outside of Supervisor mode. Accessing flash-controller hardware by a Supervisor-mode method other than this function (and companion functions, listed herein) is strongly discouraged.

This main flash array is composed of 32 pages of 512 bytes each. One page is the minimum amount of flash that can be erased. The primary input attribute to this function is a 32-bit, unsigned long used to identify pages to be erased or not. Bit 0 corresponds to the page beginning at address 0x(00)00_0000. Bit 31 corresponds to the page beginning at 0x(00)00_3E00. Set each bit to “1” to protect it from erase or to “0” to erase.

Flash protection must be disabled prior to calling RMF_FLASH_ERASE. This is done by calling RMF_UNPROTECT. Once the erase operation is done, re-assert flash protection by calling RMF_PROTECT.

7.6.3.2 RMF_FLASH_ERASE input structure

Example 7-8. Input structure syntax

```
typedef struct {
   unsigned long pw;
   unsigned long mask;
   unsigned short verify;
   unsigned short reserved;  // write as 0
   unsigned short reserved2;  // write as 0
   } rmf_flash_erase_params_t;
```

Table 7-44. rmf_flash_erase_params_t input 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 this function. If any other value than RMF_ENABLE_FLASH_ERASE[1] is used as the value for this parameter, the function will return with a failed status code.
mask	Erase/Protect Mask Bit 0 corresponds to the page beginning at address 0x(00)00_0000. Bit 31 corresponds to the page beginning at 0x(00)00_3E00. Set each bit to “0” to protect it from erase or to “1” to erase. The mask parameter must be 0xFFFFFFFF for all mass-erase requests. The mask value is ANDed with the PRR to determine what pages can be legally erased. If the ANDed value is anything less than 0xFFFFFFFF (the normal case in most applications), the request will be converted to a series of page-erase operations.
verify	TRUE = Once program operation is complete, run a verification of the erased area is “all 1s”. FALSE = Do not run verify check.
reserved	Write as 0x0.
reserved2	Write as 0x0.

1 Defined in rom_functions.h for this device

7.6.3.3 RMF_FLASH_ERASE output structure

Example 7-9. 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;
```

Table 7-45. rmf_flash_erase_params_t output 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 7-40 for additional values.
first_error	Address of first error found in any verify operations

7.6.3.4 RMF_FLASH_ERASE access/security policies

Table 7-46 details security policies for RMF_FLASH_ERASE.

Table 7-46. Access/Security Policies for RMF_FLASH_ERASE

Main flash array	
Security enabled	**Security disabled**
Subject to PRR	

Table 7-47. RMF_FLASH_ERASE parameters

Parameter	Description
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” on page 78 for additional details.

This example attempts to perform a mass erase of the main flash array.

Example 7-10. RMF_FLASH_ERASE access/security

```
static rmf_flash_erase_params_t eparams;
rmf_flash_op_sts_t *ests;

eparams.pw = RMF_ENABLE_FLASH_ERASE;
eparams.mask = 0xFFFFFFFF;   // mass erase
eparams.verify = TRUE;
eparams.reserved = 0;
eparams.reserved2 = 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
}
```

7.6.4 RMF_FLASH_PROTECT and RMF_FLASH_UNPROTECT

7.6.4.1 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 effect any permanent change in the protection state of the part. “Flash Options register” on page 66 for additional details.

7.6.4.2 RMF_FLASH_PROTECT and RMF_FLASH_UNPROTECT input structure

No parameters are necessary for either function. Supply a NULL structure pointer to the call_trap() function when invoking these functions.

7.6.4.3 RMF_FLASH_PROTECT and RMF_FLASH_UNPROTECT output structure

These functions always succeed. They return a NULL pointer.

7.6.4.4 RMF_FLASH_PROTECT/UNPROTECT access/security policies

These functions may be called at any time.

Example 7-11. Function access/security

```
call_trap(RMF_FLASH_UNPROTECT, NULL);// unprotect flash
call_trap(RMF_FLASH_PROTECT, NULL);   // protect flash
```

7.6.5 RMF_FLASH_UNSECURE

7.6.5.1 Description

This function 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 effect any permanent change in the security state of the part. “Security” on page 69 for additional details.

7.6.5.2 RMF_FLASH_UNSECURE input structure

No parameters are necessary. Supply a NULL structure pointer to the call_trap() function when invoking this function.

7.6.5.3 RMF_FLASH_UNSECURE output structure

This functions always succeeds. It returns a NULL pointer.

7.6.5.4 RMF_FLASH_UNSECURE access/security policies

This function may be called at any time.

External parties (slave port and BDM port) normally mass erase the device to clear security.

Example 7-12. Function access/security

```
call_trap(RMF_FLASH_UNSECURE, NULL);// clear security until next reset operation
```

7.6.6 RMF_CRC

7.6.6.1 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.

Features of the CRC function include:

- 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

7.6.6.2 RMF_CRC input structure

Example 7-13. 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;
```

Table 7-48. rmf_crc_params_t input parameters

Parameter	Description
seed	This is 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.

7.6.6.3 RMF_CRC output structure

Example 7-14. Output structure syntax

```
typedef struct {
   unsigned short coco;
   unsigned short sts;
   unsigned long crc;
} rmf_crc_sts_t;
```

7.6.6.4 RMF_CRC error codes

This function always returns a value.

Example 7-15. Error codes

```
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   = 0x00003FFF;
crc_results = (call_trap(RMF_CRC, &crc_params)).ptr;
if (crc_results->sts == RMF_ERROR_NONE) {
   crc = crc_results->crc;
}
```

7.6.6.5 RMF_CRC access/security policies

This function can be called at any time.

Chapter 8 Slave Port Interface

8.1 Introduction

The MMA955xL 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 RGPIO3 is found to be low during the boot process, then SPI mode will be programmed; see "Slave memory map"

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 (STOPNC). 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 8-1.

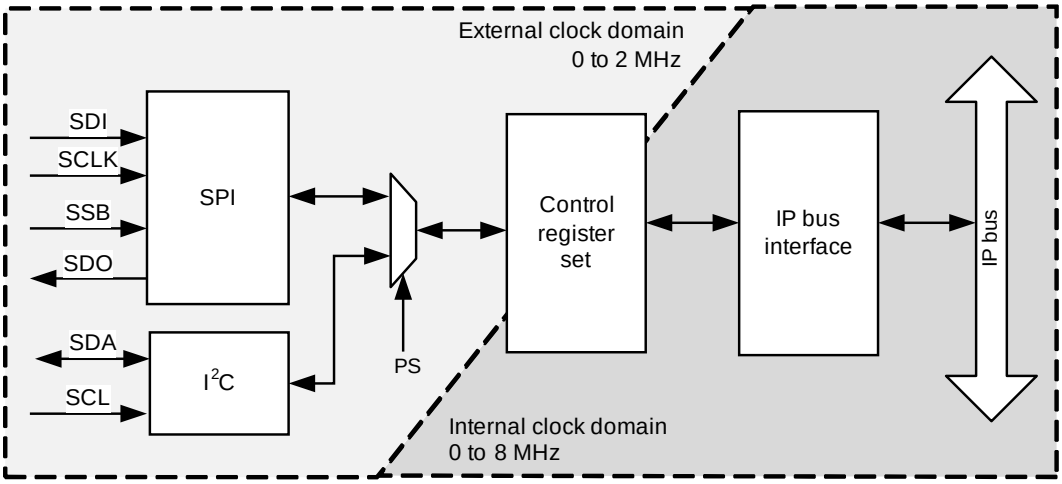


Figure 8-1. Combination SPI and I^2C slave port

NOTE

The MMA955xL platform includes both interfaces. The selection of the operating mode between I^2C and SPI is initialized at startup. If SSB (RGPIO3) is found to be low during the boot process, SPI mode will be programmed. For details, see "Slave Port Interface register descriptions" on page 125.

Immediately after a device reset, the state of pin number 8 (RGPIO3 / SDA1 / SSB functions) is used to select the slave port interface mode. This implies important rules in the way the host controller, or more generally, the complete system should be handling this pin.

First of all, whenever a reset occurs on MMA955xL, RGPIO3 pin level shall be consistent with the interface mode of operation. This is particularly important if this pin is driven from external devices. If RGPIO3 level does not match the current mode of operation, an alternate mode is selected and communication with the host is lost.

If I^2C mode is used, a good practice is to tie RGPIO3 to a pull-up resistor so that it defaults to high level. Note that such a connection exists when the Master I^2C interface is used (SDA1 function for pin 8).

When using SPI mode for the slave interface, the situation is more complex as the same pin plays two roles: SSB and mode selection. Moreover, after a SPI read or write operation, the SSB line returns to high level. Consequently, if the host sends a command to the MMA955xL that induces a subsequent reset immediately after the write transaction, it forces the SSB line to low level. This means that SPI mode is still selected after reset. The duration for the SSB line to be kept low typically depends on the latency between the write transaction and the execution of the reset command. Such latency can be significant for the MMA9553L pedometer firmware as the Command Interpreter and Scheduler applications are running at 30Hz which gives a 33ms typical latency.

The rule obviously applies also when a hardware reset is issued by the host through MMA955xL pin 3 (RESETB active low). Again the host has to drive the SSB line low prior to releasing the hardware reset line to high level, which immediately triggers the MMA955xL reset and boot sequence. Keeping the SSB line low during 1 ms duration after RESETB is released is enough for the MMA955xL slave device to re-boot into SPI mode.

8.2 I²C features and limitations

If RGPIO3 is found to be high during the boot process, then I²C mode will be programmed.

Figure 8-2 shows typical connection of master/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.

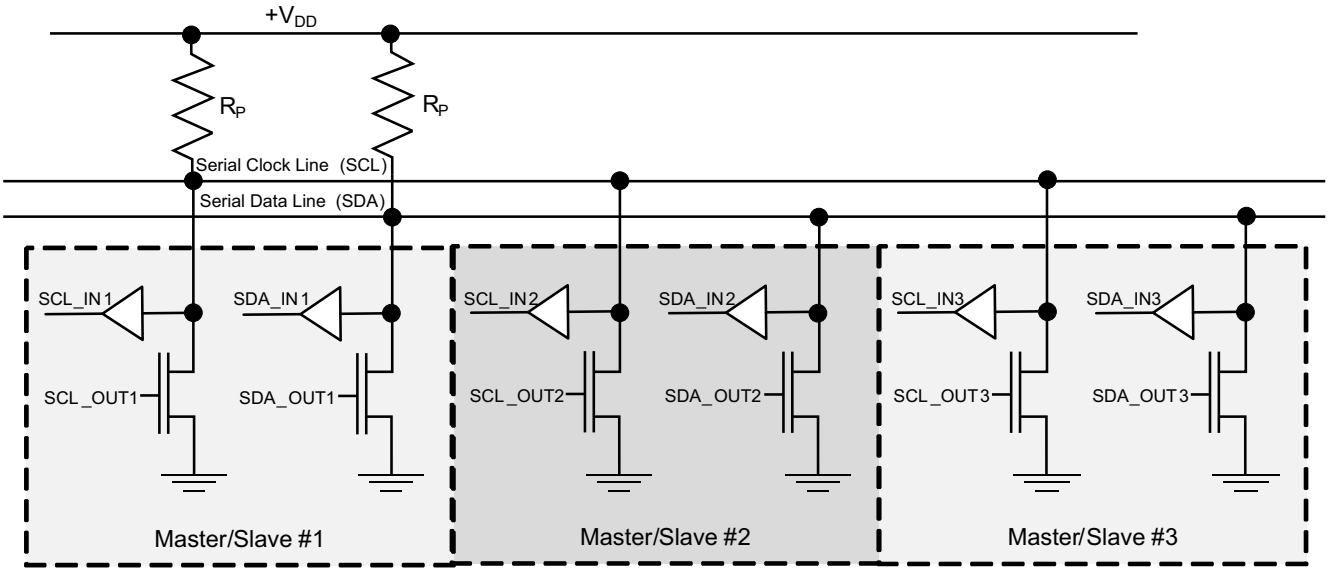


Figure 8-2. I²C wired-and/open-drain bus

8.2.1 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
  - — 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 I²C communications occurring 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.

8.2.2 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:

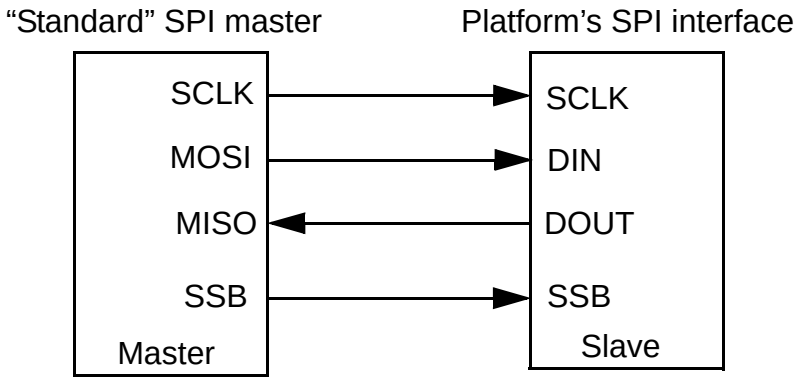
- Seven-bit addressing only
- Maximum SCL frequency of operation equals 2 Mbps
- General call is not supported
- “Start Byte” is not supported
- Slave-only operation
- Input filters are not implemented
- Use of standard, digital-I/O impose loading limitations on the bus

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 MMA955xL platform slave I²C port can be used with Hs-mode devices operating up to 2 Mbps.

8.2.3 SPI features and limitations

The MMA955xL architecture also supports Serial Peripheral Interface (SPI) communication for digital communication. The SPI is used for synchronous, serial communication between a master device and one or more slave devices. See Figure 8-3 for an example of how to configure one master with one MMA955xL device.

The MMA955xL platform is always operated as a slave device. Typically, the master device is the host microcontroller, which drives the clock (SCLK) and chip-select (SSB) signals.



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)
DIN	Data in
MISO	Data out

Figure 8-3. Dedicated connection to SPI host

The SPI interface consists of two control lines and two data lines: SSB, SCLK, SDI and SDO. 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. SDI and SDO are the SPI Data Input and the SPI Data Output.

Figure 8-4 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.

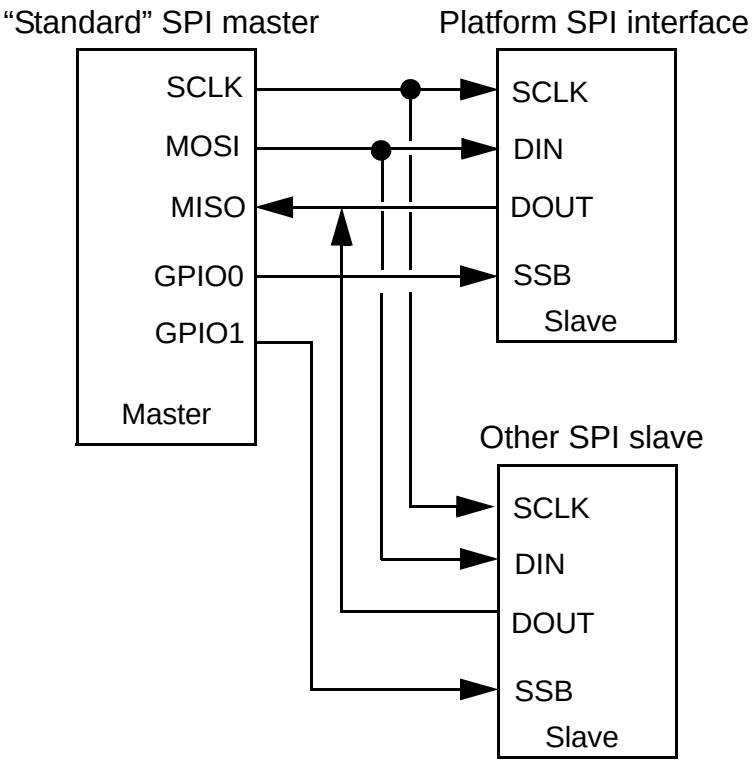


Figure 8-4. Shared Connection to SPI Host

8.2.4 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.

8.2.5 SPI limitations

- Fixed clock polarity: The SDI and SDO 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 (“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.

8.3 Data coherency issues

Mailbox registers are shared by the CPU and external master. Both CPU and external master have the ability to read and write these registers. 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- and 32-bit data.

For consistency, the SPI interface is designed to support a protocol similar to the I^2C, as well as previous (non-MCU-based) Freescale sensors.

From the CPU’s perspective, 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 MMA955xL device 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.
1. Two binary semaphores (mutex operators) are available for applications requiring more rigorous control of shared resources.

8.3.1 Read buffer

When any byte is read by the master, the entire four-byte region in which that byte resides will be cached in a four-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. The user is encouraged to view the mailbox area as shown in Table 8-1. The four-byte read buffer can be used to make simultaneous reads of the mailboxes in any row of the table.

Table 8-1. 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 8-2.

Table 8-2. 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 8-2, variables 1, 2 and 3 are 32-bit variables. Variables 4 through 9 are 16-bit variables and variables 10 through 17 are eight-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 8-3.

Table 8-3. 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 8-3 highlights the improperly aligned variables. Depending on when reads and writes occur, it is possible that the master and CPU would see inconsistent values.

The four-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.

8.3.2 Binary semaphore (mutex) operation

The MMA955xL platform includes two semaphore registers that can be used by the CPU and system master to negotiate ownership of shared assets. These can be mailbox registers or any other shared item. These registers can be read by only one of the two parties at any point in time. Simultaneous attempts will be serialized by the module.

Each semaphore register has several possible actions associated with it, as shown in Table 8-4.

Table 8-4. 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 8-4 are atomic, assuring that the CPU and master can unambiguously negotiate ownership of any asset.

Operation of the two semaphore registers is identical, allowing for simultaneous negotiations of 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.

8.4 Slave memory map/register definitions

The slave-port module is organized as a memory-mapped peripheral on the eight-bit IP bus. Table 8-5 specifies the module memory map. Details of each register are provided in the following section.

8.4.1 Slave memory map

Table 8-5. SLAVE memory map

Offset address	Register	Access	Reset	Details
0x00	Slave Port Mailbox Register *n* (SP_MB*n*)	Read/Write	0x00	8.4.2.1/8-125
0x20	Slave Port Binary Semaphore (Mutex) Register *n* (SP_MUTEX*n*)	Read/Write	0x00	8.4.2.2/8-126
0x22	Slave Port I^2C Address Register (SP_ADDR)	Read/Write	0x23	8.4.2.3/8-126
0x23	Slave Port Status and Control Register (SP_SCR)	Read/Write	0x00	8.4.2.4/8-127
0x24	Slave Port Write Status Register 0 (SP_WSTS0)	Read	0x00	8.4.2.5/8-129
0x25	Slave Port Write Status Register 1 (SP_WSTS1)	Read	0x00	8.4.2.6/8-130
0x26	Slave Port Write Status Register 2 (SP_WSTS2)	Read	0x00	8.4.2.7/8-131
0x27	Slave Port Write Status Register 3 (SP_WSTS3)	Read	0x00	8.4.2.8/8-132
0x28	Slave Port Read Status Register 0 (SP_RSTS0)	Read	0x00	8.4.2.9/8-133
0x29	Slave Port Read Status Register 1 (SP_RSTS1)	Read	0x00	8.4.2.10/8-134
0x2A	Slave Port Read Status Register 2 (SP_RSTS2)	Read	0x00	8.4.2.11/8-135
0x2B	Slave Port Read Status Register 3 (SP_RSTS3)	Read	0x00	8.4.2.12/8-136
0x2C	Slave Port Mutext Timeout Register *n* (SP_MTOR*n*)	Read/Write	0x00	8.4.2.13/8-137
0x2E	Slave Port Output Interrupt Control Register (SP_OIC)	Read/Write	0x00	8.4.2.14/8-138

8.4.2 Slave Port Interface register descriptions

8.4.2.1 Slave Port Mailbox Register *n*

The slave port is the main communication channel between the system master and the MMA955xL. 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 sensor master. Reads by both CPU and 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.

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*X* registers or none, as desired.

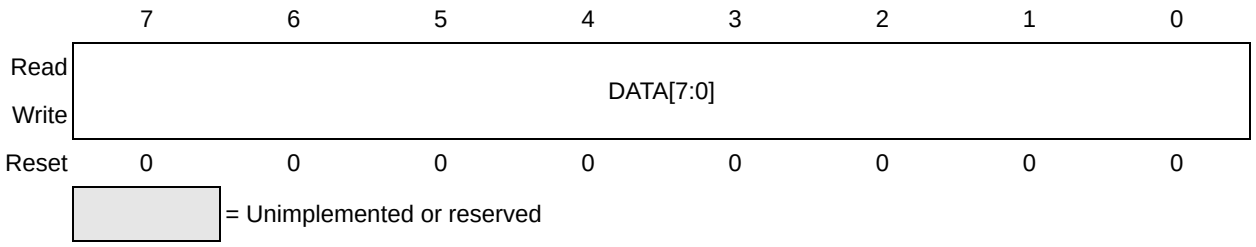


Figure 8-5. Slave Port Mailbox Register *n* (SP_MBn)

Table 8-6. Slave Port Mailbox Register *n* (SP_MBn) field descriptions

Bit(s)	Field	Description
7-0	DATA	See detailed description above.

8.4.2.2 Slave Port Binary Semaphore (Mutex) Register *n*

For additional information regarding operation of the semaphore registers, see Binary Semaphore Mutex Register. SP_MUXTEX0 and SP_MUXTEX1 are identical in form and operation.

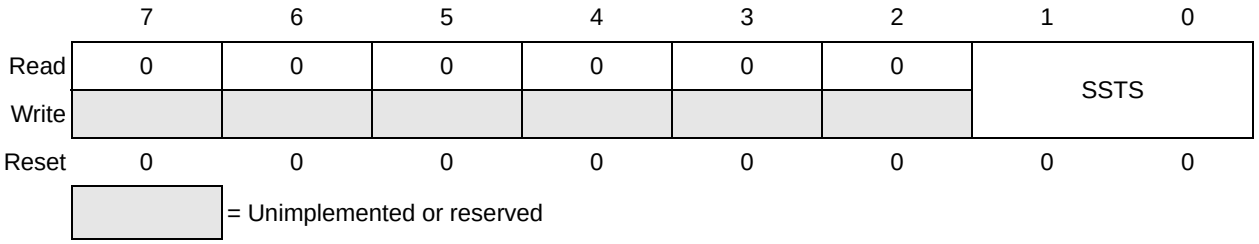


Figure 8-6. Slave Port Binary Semaphore Mutex*n* register (SP_MUTEXn)

Table 8-7. Slave Port Binary Semaphore Mutex*n* register (SP_MUTEXn) field descriptions

Bit(s)	Field	Description
7-2	—	Reserved. RO bits (always read as 0)
1-0	SSTS	Semaphore Status **Note:** 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. 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.

8.4.2.3 Slave Port I^{2}C address register

The seven-bit address for this port can be specified by writing to this register which should be configured in advance of any I^{2}C 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.*

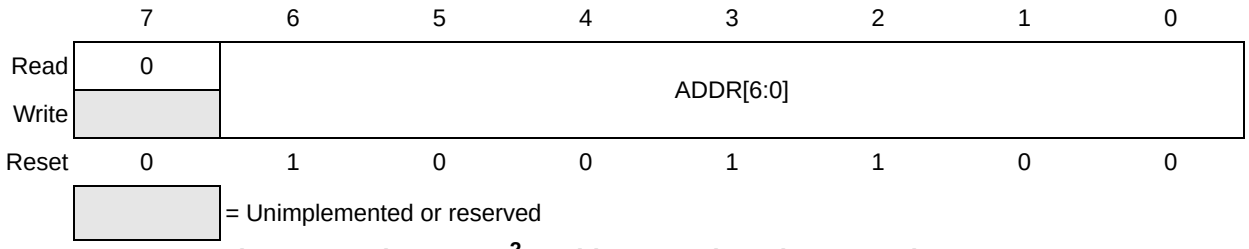


Figure 8-7. Slave Port I^{2}C Address Register (SP_ADDR)

Table 8-8. Slave Port I²C Address Register (SP_ADDR) field descriptions

Bit(s)	Field	Description
7	—	Reserved
6-0	ADDR	See detailed description above.

8.4.2.4 Slave Port Status and Control register

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

= Unimplemented or reserved

Figure 8-8. Slave Port Status and Control register (SP_SCR)

Table 8-9. Slave Port Status and Control register (SP_SCR) field descriptions

Bit(s)	Field	Description
7	EN	Slave Port Enable **Note:** Note: On devices that support only the of SPI or I²C interface, this register bit is read-only. **Note:** This bit it initialized to 1 by the boot manager at startup. **Note:** 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 I²C slave port is enabled (based on choice of PS bit).
6	PS	Port Select On devices that support only the of SPI or I²C interface, this register bit is read-only and fixed to the appropriate value as per Table 8-8. On the MMA955xL, the PS bit is initialized by the boot manager at startup. If SSB (RGPIO3) is found to be low during the boot process (immediately following any reset), the PS is set to 1 (SPI mode). Otherwise, I²C 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 I²C interface has been selected for use as slave port. 1 The SPI interface has been selected for use as slave port.
5	ACTIVE	ACTIVE/CSR 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 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.

Table 8-9. Slave Port Status and Control register (SP_SCR) field descriptions

Bit(s)	Field	Description
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*X* 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_MBX registers have been written by the system master. Interrupt operation is further qualified by the WUP bits.
1-0	WUP	Wake-up Configuration 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

Table 8-10. Slave Port Active bit descriptions

PS	ACTIVE	Function
0	0	The I^2C finite state machine is in the SLEEP state.
0	1	The I^2C port is active.
1	0	The SPI SSB input is high. The SPI is not in use.
1	1	The SPI SSB input is low (enabled) and the SPI is selected.

8.4.2.5 Slave Port Write Status Register 0

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*X* 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.

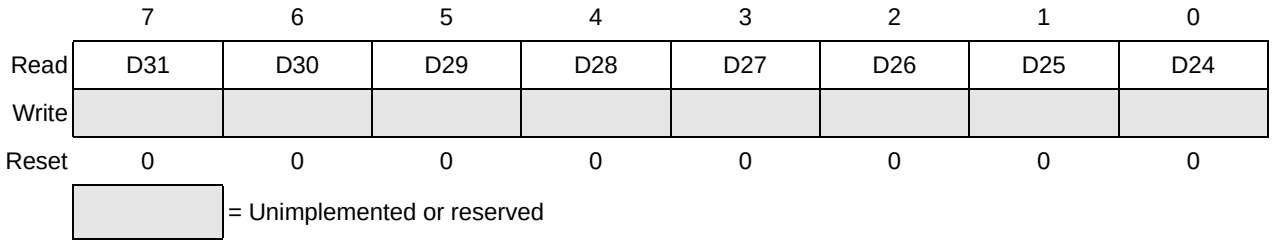


Figure 8-9. Slave Port Write Status Register 0

Table 8-11. SP_WSTS0 Register 0 field descriptions

Bit(s)	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.
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.

8.4.2.6 Slave Port Write Status Register 1

	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

= Unimplemented or reserved

Figure 8-10. Slave Port Write Status Register 1 (SP_WSTS1)

Table 8-12. Slave Port Write Status Register 1 (SP_WSTS1) field descriptions

Bit(s)	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.
0	D16	Maps to SP_MB16 Bit sets when SP_MB16 is read by the system master.

8.4.2.7 Slave Port Write Status Register 2

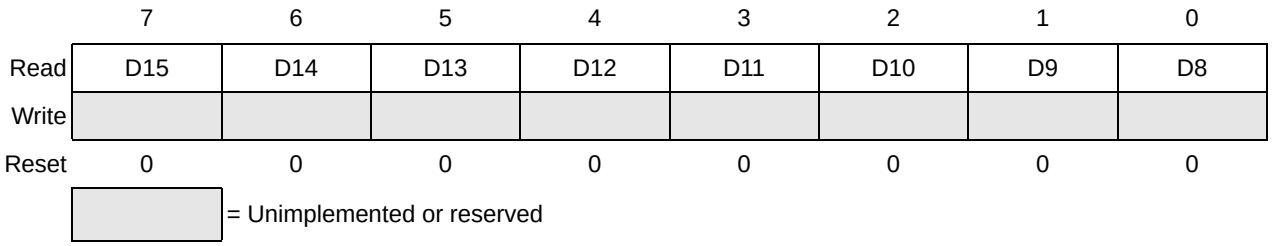


Figure 8-11. Slave Port Write Status Register 2 (SP_WSTS2)

Table 8-13. Slave Port Write Status Register 2 (SP_WSTS2) field descriptions

Bit(s)	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.

8.4.2.8 Slave Port Write Status Register 3

	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

= Unimplemented or reserved

Figure 8-12. Slave Port Write Status Register 3 Format (SP_WSTS3)

Table 8-14. Slave Port Write Status Register 3 Format (SP_WSTS3) field descriptions

Bit(s)	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.

8.4.2.9 Slave Port Read Status Register 0

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*X* 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.

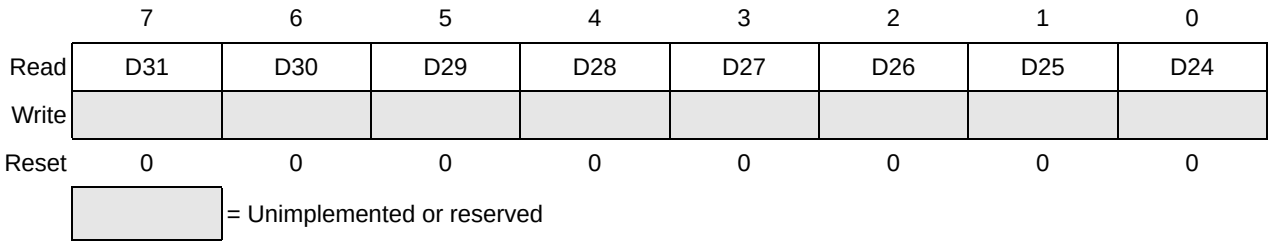


Figure 8-13. Slave Port Read Status Register 0 Format (SP_RSTS0)

Table 8-15. Slave Port Read Status Register 0 Format (SP_RSTS0) field descriptions

Bit(s)	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.
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.

8.4.2.10 Slave Port Read Status Register 1

	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

= Unimplemented or reserved

Figure 8-14. Slave Port Read Status Register 1 Format (SP_RSTS1)

Table 8-16. Slave Port Read Status Register 1 Format (SP_RSTS1) field description

Bit(s)	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.
0	D16	Maps to SP_MB16 Bit sets when SP_MB16 is read by the system master.

8.4.2.11 Slave Port Read Status Register 2

	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

= Unimplemented or reserved

Figure 8-15. Slave Port Read Status Register 2 Format (SP_RSTS2)

Table 8-17. Slave Port Read Status Register 2 Format (SP_RSTS2) field descriptions

Bit(s)	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.

8.4.2.12 Slave Port Read Status Register 3

	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

= Unimplemented or reserved

Figure 8-16. Slave Port Read Status Register 3 Format (SP_RSTS3)

Table 8-18. Slave Port Read Status Register 3 Format (SP_RSTS3) field descriptions

Bit(s)	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.

8.4.2.13 Slave Port Mutext Timeout Register *n*

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.

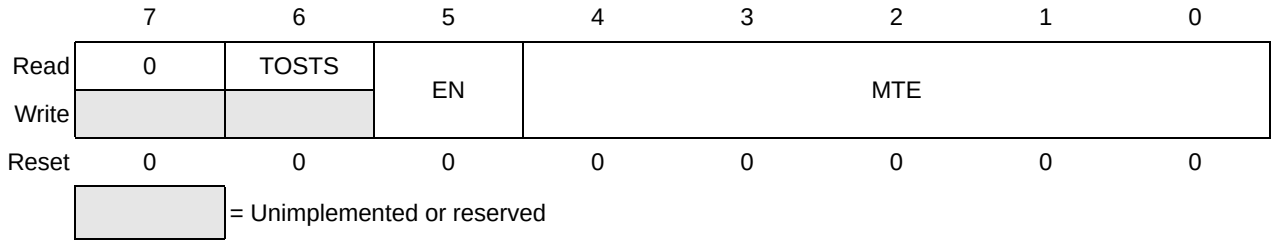


Figure 8-17. Slave Port Mutext Timeout Register *n* Format (SP_MTORn)

Table 8-19. Slave Port Mutext Timeout Register *n* Format (SP_MTORn) field descriptions

Bit	Name	Description
7	—	Reserved
6	TOSTS	Mutext Timeout Status **Note:** Note: 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.
4-0	MTE	Mutext Timeout Exponent The length of the timeout = 128 X 2^{MTE} X $P_{osc\text{-}high}$.

8.4.2.14 Slave Port Output Interrupt Control Register

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 MMA955xL 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)

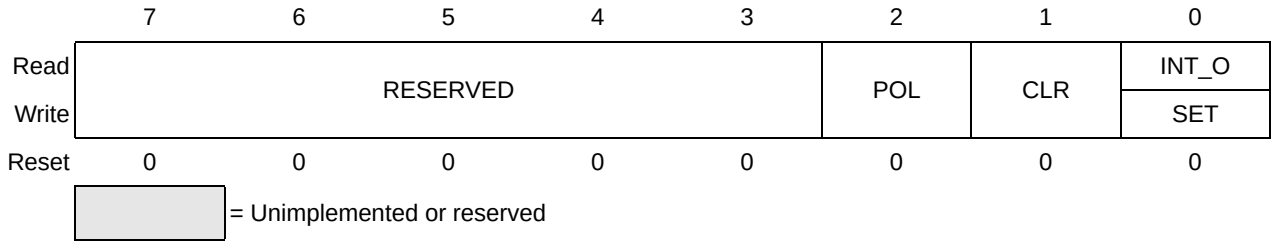


Figure 8-18. Slave Port Output Interrupt Control Register (SP_OIC)

Table 8-20. Slave Port Output Interrupt Control Register (SP_OIC) field descriptions

Bit(s)	Field	Description
7-3	R	Reserved
2	POL	Output Polarity **Note:** The POL bit should only be changed when CLR is being written as 1. This forces INT_O to its new deasserted 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) 0 No action 1 De-assert INT_O (INT_O = POL)
0	INT_O	INT_O Assert INT_O (INT_O = NOT POL) This bit reflects the current state of the INT_O function. The INT_O Functional Truth Table defines INT_O behavior as a function of SET, CLR and the hardware-clear events. SET Set/Assert (INT_O = NOT POL 0 Writing 0 No action 1 Writing 1 Assert INT_O (INT_O = NOT POL)

8.4.3 Output interrupt timing

This bit reflects the current state of the INT_O function.

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

Table 8-21. Interrupt logic value as a function of the POL bit

Table 8-22 defines INT_O behavior as a function of SET, CLR and the hardware-clear events.

Table 8-22. 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[1]	NOT POL
1	No change	0	1	POL
1	No change	0	0	Previous INT_O

1 Hardware-clear events should never occur at the same time as Write 1 to SET, assuming that the following communication protocols are followed.

Figure 8-19 on page 139 and Figure 8-20 on page 140 show the I^2C slave port traffic versus interrupt assertion and de-assertion for both active high and active low interrupt outputs.

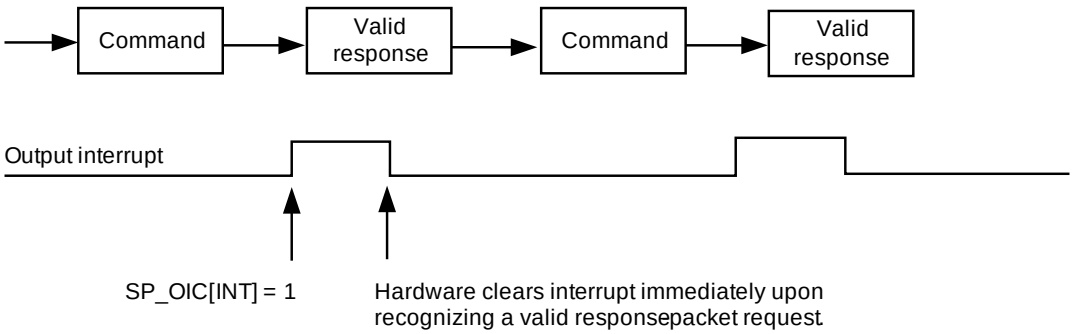


Figure 8-19. Output interrupt timing (POL = 0)

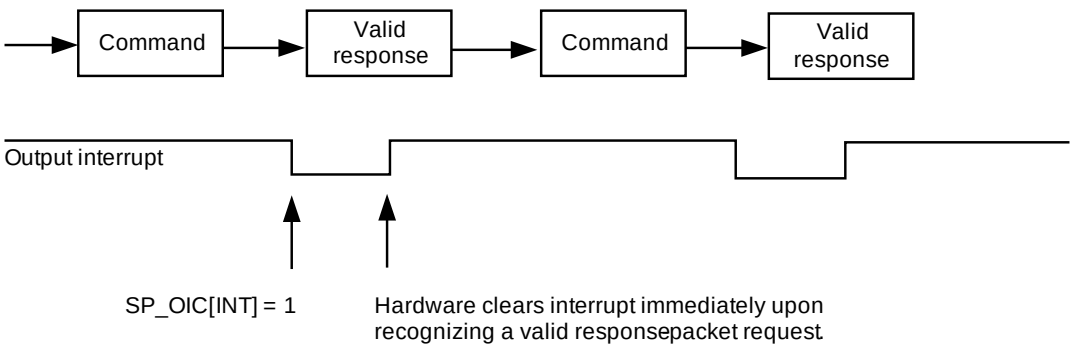


Figure 8-20. Output interrupt timing (POL = 1)

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. Table 8-23 reproduces the SIM_PMCR2 settings required to program Pin 11 as INT_O. (“Pin Mux Control Register0” on page 212 for additional details.)

Table 8-23. Pin-11 max controls

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

1 Actual function is a combination of RGPIO controls and port controls.

8.5 I²C serial protocol and timing

8.5.1 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 MMA955xL platform utilizes an internal, 8-MHz CPU and peripheral clock, yielding an internal clock rate of 125 ns. At this speed, a minimum (60 ns) SCL, high signal cannot be reliably sampled in high-speed mode. Therefore, the communication rate for this module is limited to 2 MHz or less.

The MMA955xL data sheet summarizes timing options and requirements for the slave I²C module.

8.5.2 Serial-addressing

The MMA955xL device 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 I-directional communication between master(s) and slave(s). A master (typically a microcontroller) initiates all data transfers to and from the MMA955xL platform and generates the SCL clock that synchronizes the data transfer.

The SDA line operates as both an input and an open-drain output. A pull-up resistor, typically 4.7 kΩ, is required on SDA. The SCL line operates only as an input. A pull-up resistor, typically 4.7 kΩ, is required on SCL 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.

8.5.3 Start, Stop and Repeated Start conditions

Each transmission consists of a START condition (Figure 8-21) 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.

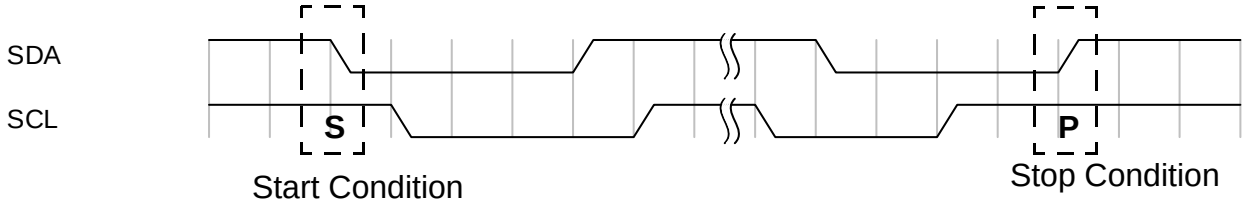


Figure 8-21. Start (S) and Stop (P) conditions

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 8-22.

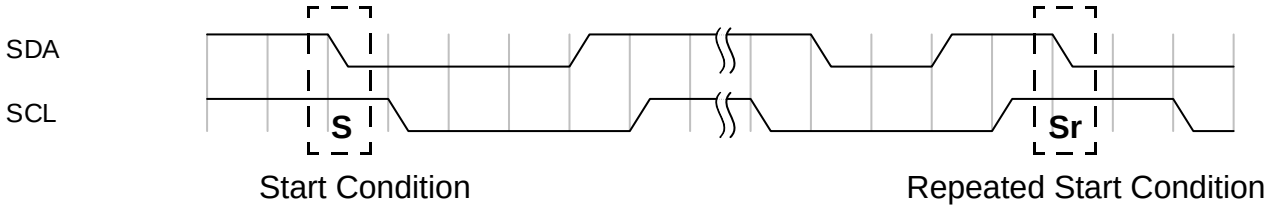


Figure 8-22. Start (S) and Repeated Start (Sr) conditions

8.5.4 Bit transfer

One data bit is transferred during each clock pulse (Figure 8-23). The data on SDA must remain stable while SCL is high.

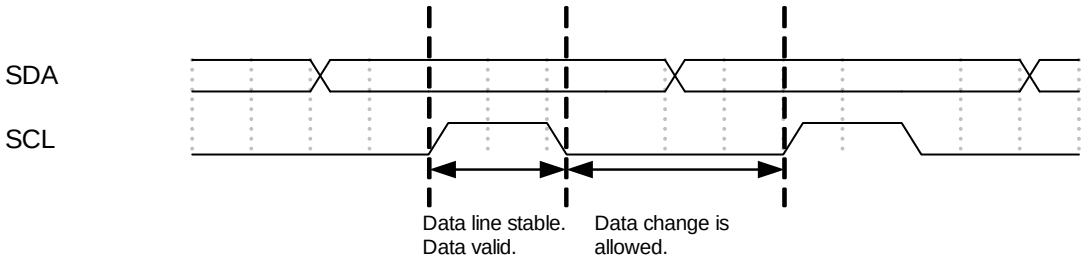


Figure 8-23. Bit transfer

8.5.5 Acknowledge

The acknowledge bit is the clocked ninth bit (Figure 8-24) 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 MMA955xL platform, the device generates the acknowledge bit because it is the recipient.

When the MMA955xL device is transmitting to the master, the master generates the acknowledge bit because the master is the recipient.

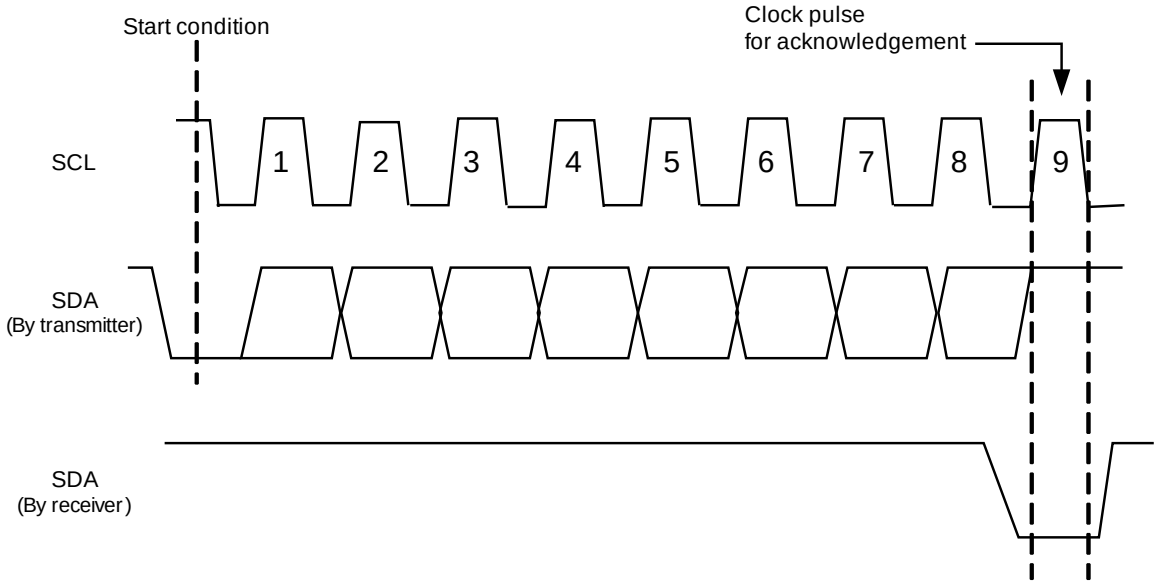


Figure 8-24. Acknowledge

8.5.6 Slave address

The MMA955xL platform has a seven-bit long slave address (Figure 8-25). 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 MMA955xL platform is software-programmable via the SP_ADDR register. This value must be programmed prior to start of any I^2C communications unless the default value (0x4C) is used.

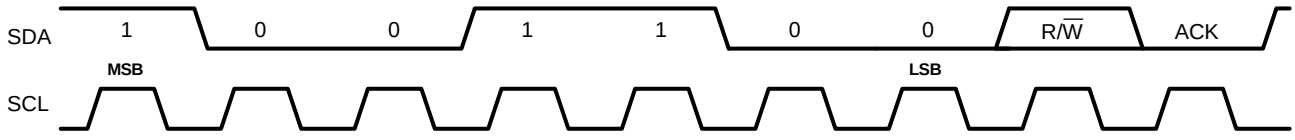


Figure 8-25. Slave address

The MMA955xL device 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.

8.5.7 Message format for writing

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

S	Start Bit/Repeated Start Bit
A	Acknowledge Bit
$\overline{A}$	Not-Acknowledge Bit
P	Stop Bit

A write to the MMA955xL platform comprises the transmission of the 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 MMA955xL device takes no action. This is shown in Figure 8-26.

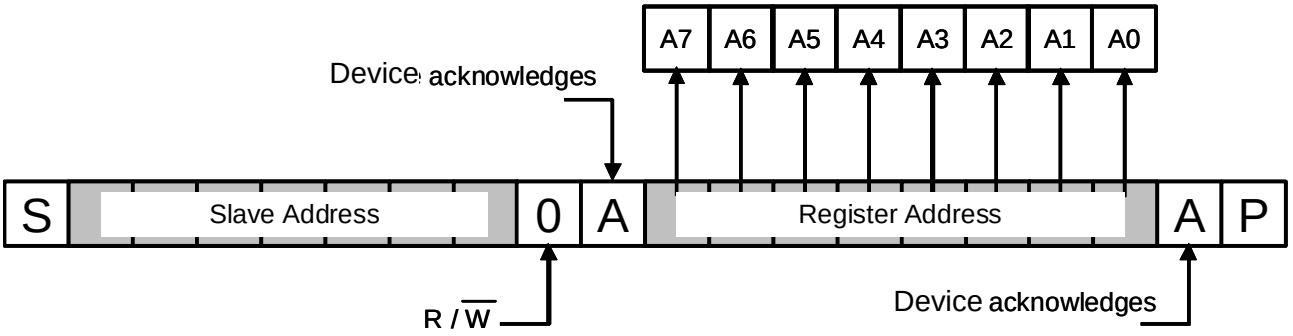


Figure 8-26. Minimal I²C command has no effect

The MMA955xL device 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 8-27 on page 145 shows the simplest case where a single register value is read.

.

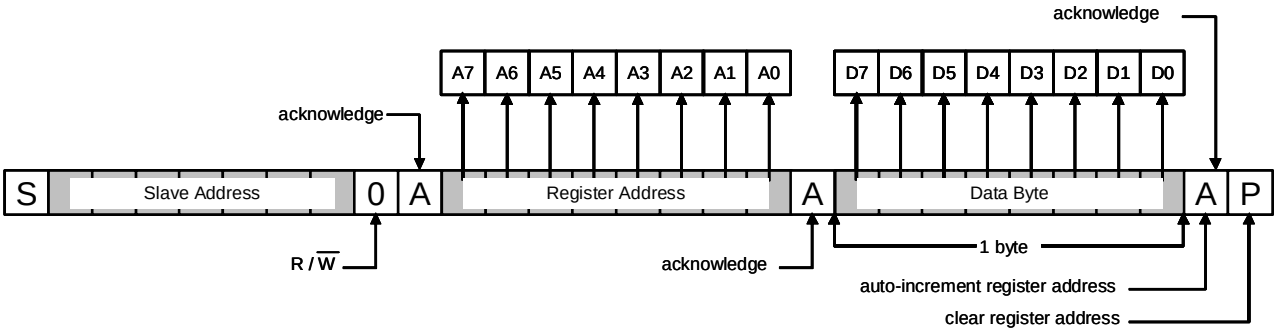


Figure 8-27. Writing one byte of information into slave I²C

Any bytes received after the register address are data bytes. The first data byte goes into the internal register of the MMA955xL platform 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 MMA955xL device 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 8-28 illustrates the case where three bytes of information are written into three consecutive, slave I²C registers.

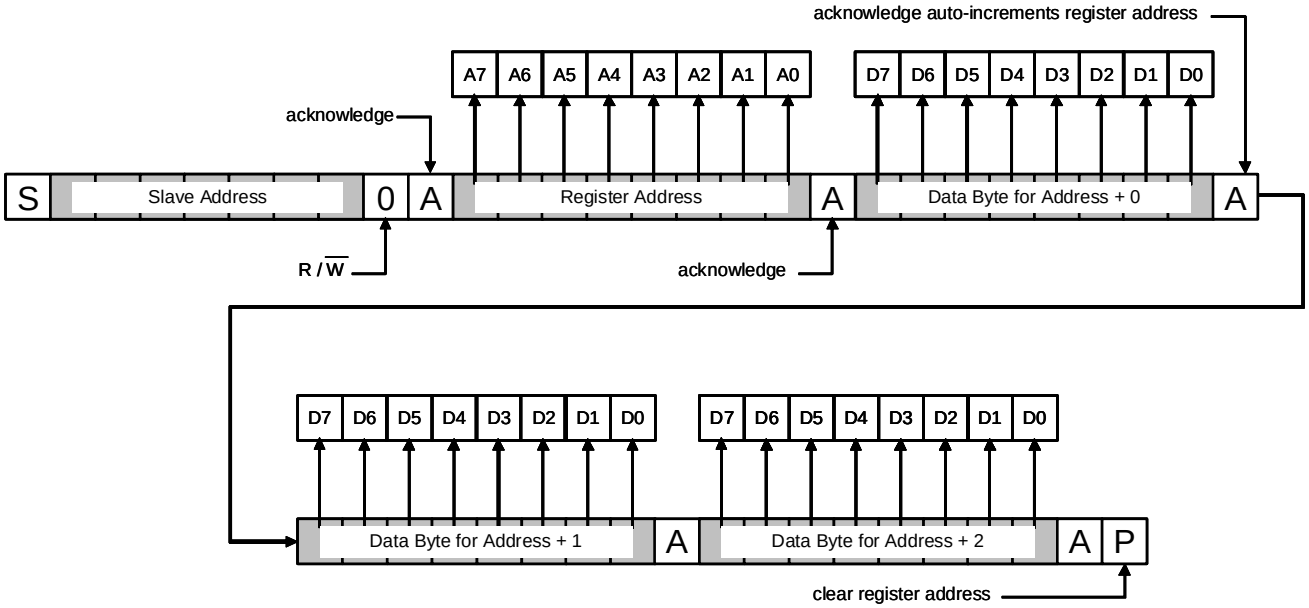


Figure 8-28. Three consecutive byte writes to the slave I²C data registers

8.5.8 Message format for reading the platform

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 MMA955xL platform is placing information on the bus. The I^2C 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 8-29.

The MMA955xL platform 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 the MMA955xL device, with the first data byte being read from the register addressed by the initialized register address.

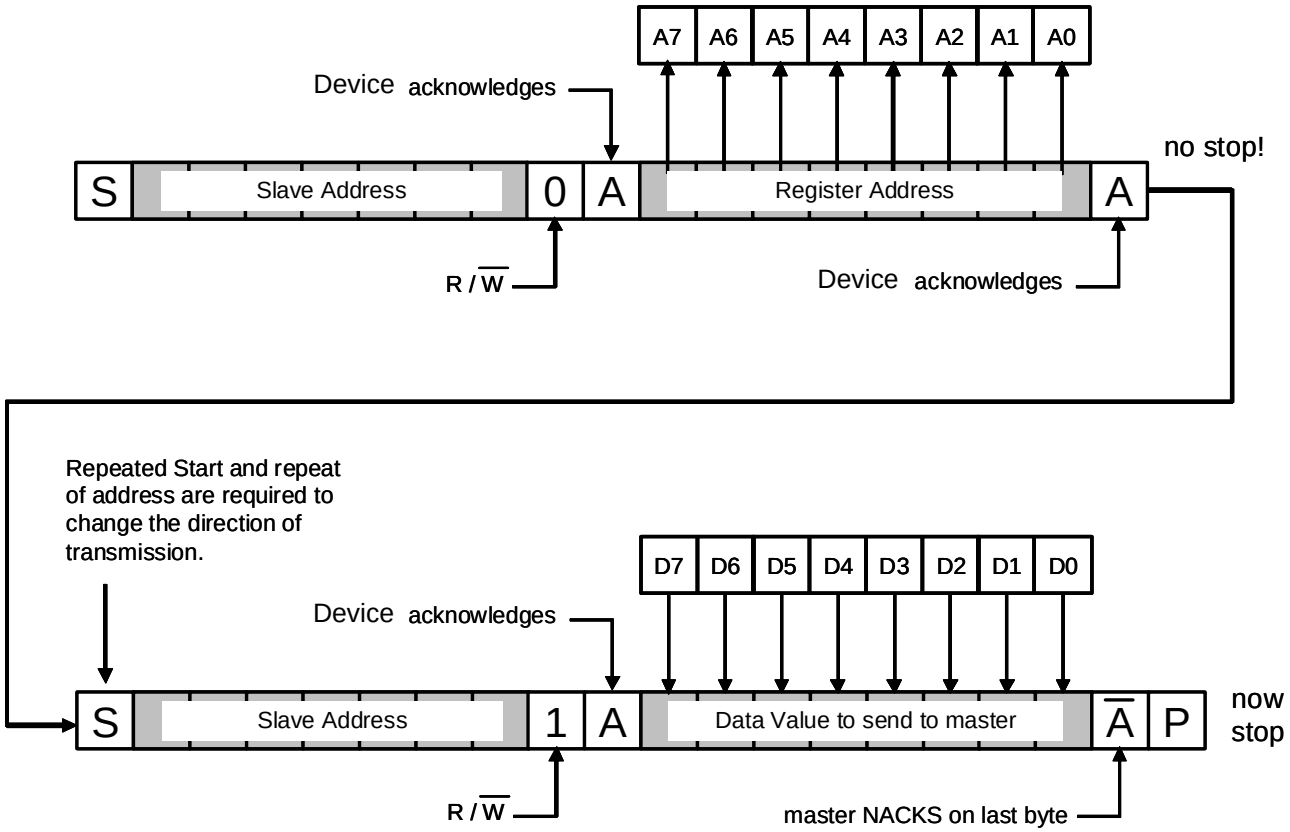


Figure 8-29. Read of a single byte of information

The I^2C 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 8-29.

Figure 8-30 extends the read sequence to access three bytes of information utilizing auto-incrementing of the address register.

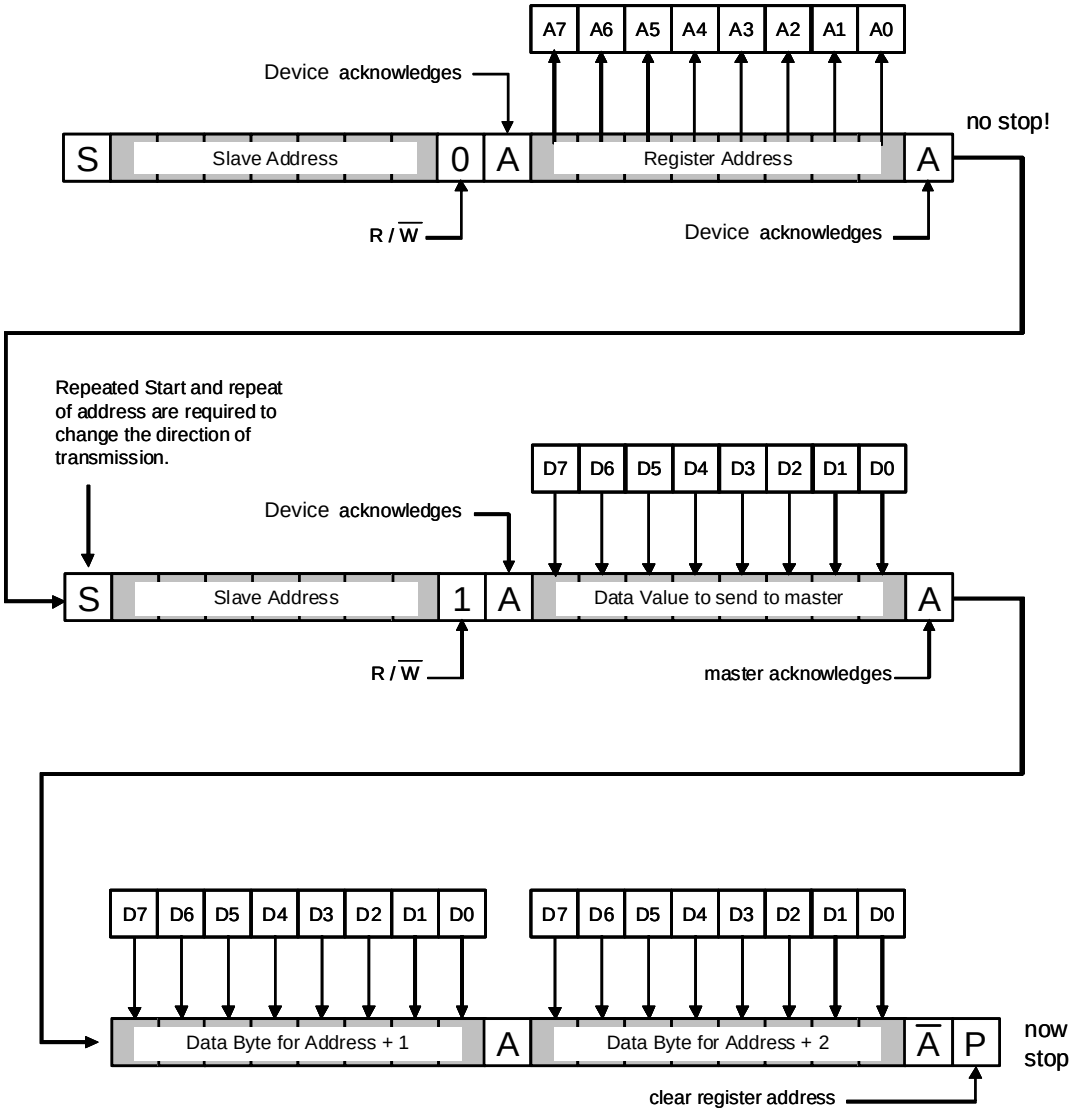


Figure 8-30. Repeated read of three bytes of information

8.6 SPI serial protocol and timing

The SPI interface consists of two control lines and two data lines: SSB, SCLK, SDI and SDO. The SSB, also known as “Slave Select” (active low), is the slave device enable which is controlled by the SPI master. SSB is driven low at the start of a transmission. SSB is then driven high at the end of a transmission.

SCLK is the SPI clock that is also controlled by the SPI master. SDI and SDO are the SPI Data Input and the SPI Data Output. The SDI and SDO 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 eight, in the case of a multiple-byte read/write.

8.6.1 SPI read operation

Figure 8-31 shows a read of a single register on the SPI port. A SPI read transfer requires that two bytes be transmitted on SDI by the host. The first consists of:

- A one-bit Read/Write signal (0 = Read, 1 = Write)
- A six-bit address
- A 1-bit don’t-care bit

The second byte on SDI is discarded by the MMA955xL platform and is shown as “dc” (Don’t Care).

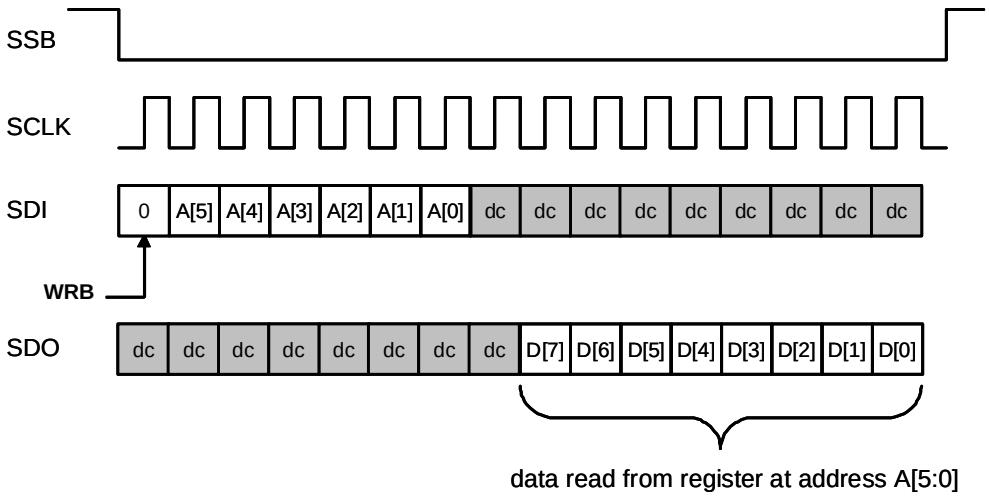


Figure 8-31. Single byte read by host using SPI

At the beginning of the transmission, the MMA955xL device does not know the type of transfer that is coming. The first byte of information on SDO 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 SDI by the master. Subsequent read addresses are automatically indexed by one.

Figure 8-32 illustrates the case where the payload is three bytes long.

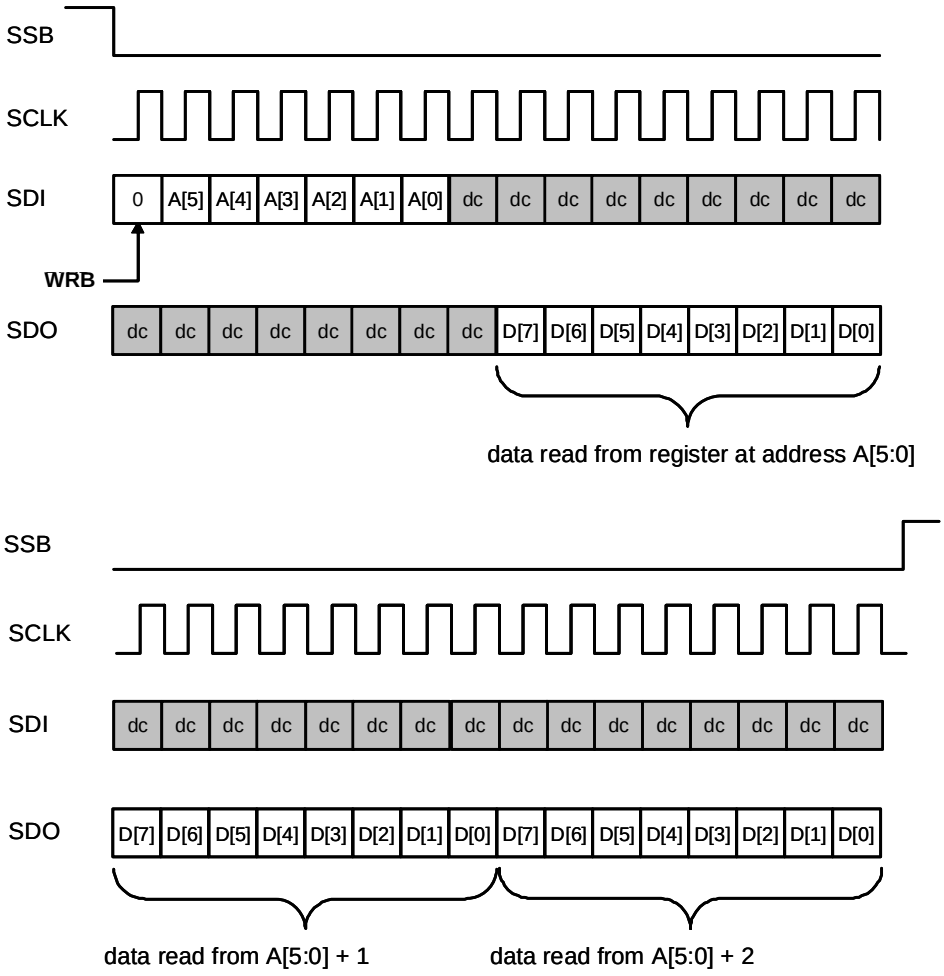


Figure 8-32. Three-byte read by host using SPI

8.6.2 SPI write operation

In order to write to one of the eight-bit registers, an eight-bit write command must be sent to the MMA955xL device. The write command consists of an MSB (0 = read, 1 = write) to indicate a write is to be made to the platform's register, followed by a six-bit address and a one-bit, don't-care bit.

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

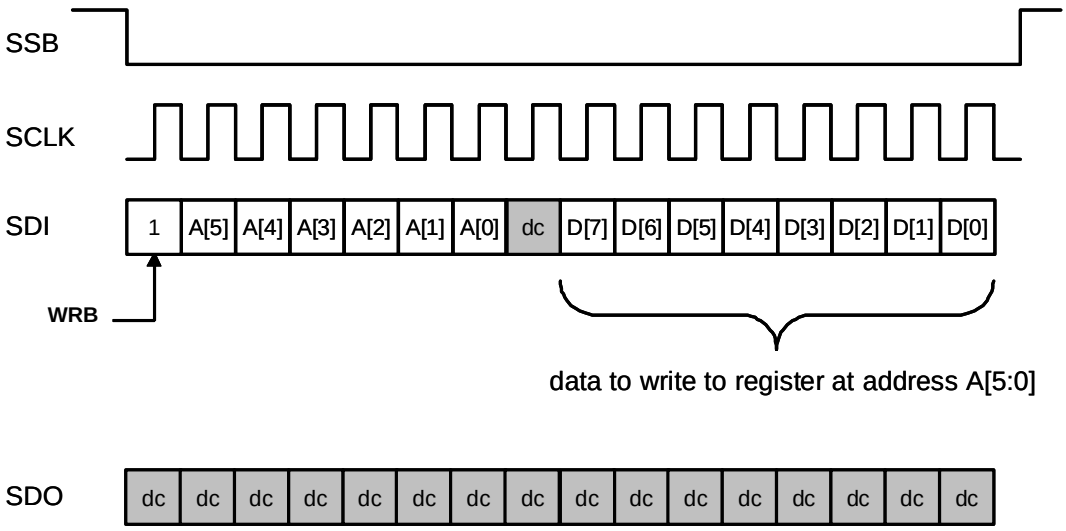


Figure 8-33. One-byte write by host using SPI

Figure 8-34 on page 151 illustrates the case where the host is writing three sequential bytes of information to the MMA955xL platform.

Note that the MMA955xL SDO line is not required for write operations. Data received on SDO by the master device should be ignored.

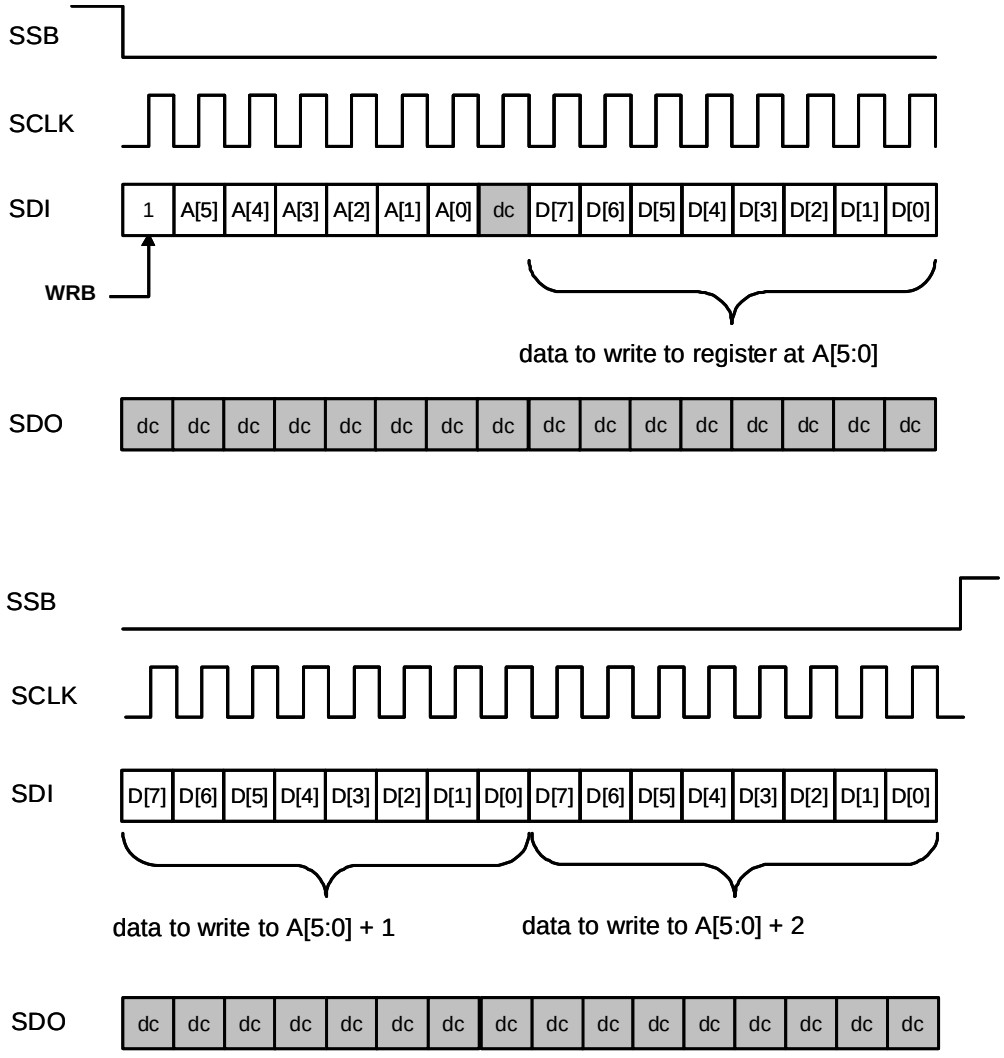


Figure 8-34. Three-byte write by host using SPI

8.7 Interrupts

The slave port module can generate three different interrupts to the CPU:

- An interrupt to notify the CPU that the system master has updated information in the mailbox registers
- Semaphore Time-out 1
- Semaphore Time-out 2

The slave port can also generate an interrupt to the host controller via the SP_OIC register.

8.7.1 Mailbox interrupt

The SP_SCR[WIE] register bit can be used to enable/disable the mailbox interrupt. SP_SCR[WWUP] can be used to configure specific operation of the mailbox interrupt.

WWUP = 00 is the default operation. In this case, assertion of interrupts is possible only in RUN mode. If the system master were to write to the mailbox registers during STOP, no action would be taken until STOP mode was exited. At that point (assuming WIE=1), the interrupt would be issued and the CPU would process the new data. This mode of operation will yield the most accurate results from the on-chip data converters, at the price of additional latency (determined by the sample frame rate) in handling of any new command/data from the system master.

WWUP = 01 includes the operation above. In addition, SP_MB31 now takes on a special role as a “wake-up register.” A write by the master to this register will force the CPU into RUN mode if it is not already there. This minimizes latency, at the possible cost of decreased accuracy on the affected set of data conversions.

WWUP = 10 takes the “wake-up register” concept to the extreme. Any write to any of the mailbox registers will result in an immediate wake-up interrupt assertion.

8.7.2 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.

8.8 Reset operation

This module is reset along with the rest of the chip. I^2C and SPI communications are not possible during reset (specifically sim_chip_resetb) assertion.

Chapter 9 Inter-Integrated Circuit

9.1 Introduction

The Inter-Integrated Circuit (IIC or I^2C) provides a method of communication between a number of devices. The interface is designed to operate up to 100 kbps with maximum bus loading and timing. The device is capable of operating at higher baud rates, up to a maximum of clock/20 (or 400 kHz), 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. System Management Bus Specification (SMBus), Version 2 is supported.

This module is the same master/slave I^2C module found on many Freescale devices. The MMA955xL platform devices already have a dedicated slave interface to communicate with a host processor as detailed in “Slave Port Interface” on page 115. Consequently, this additional I^2C circuit will be used mainly as a master I^2C for controlling and communicating with external peripherals such as any sensors. Doing so, the MMA955xL device will behave as an effective and autonomous sensor hub.

The I^2C 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 $STOP_{SC}$, this module should not be used in that mode.

NOTE

On MMA955xL devices, the SDA1 and SCL1 I^2C lines share the same pins as SDO and SSB slave SPI lines. Consequently, this additional I^2C circuit can only be used when the MMA955xL slave interface is operating under I^2C mode.

9.1.1 Features

The I^2C module includes these features:

- Compatible with I^2C bus standard
- Multi-master operation
- Software programmable for one of 64 serial clock frequencies
- Software-selectable acknowledge bit
- Interrupt-driven, byte-by-byte data transfer
- Arbitration-lost interrupt with automatic mode-switching from master to slave
- Calling-address identification interrupt
- START and STOP signal generation/detection
- Repeated START signal generation/detection
- Acknowledge-bit generation/detection

- Bus-busy detection
- General call recognition
- 10-bit address extension
- Programmable glitch input filter
- Address matching causes wake-up when MCU is in Stop mode

9.1.2 Modes of operation

The I^2C module's MCU modes include:

- Run mode — The basic mode of operation. To conserve power in this mode, disable the module.
- Wait mode — The module continues to operate when the MCU is in Wait mode and can provide a wake-up interrupt.
- Stop mode — The I^2C is inactive in Stop mode for reduced power consumption except address-matching is enabled in Stop mode. The STOP instruction does not affect I^2C register states.

9.1.3 Block diagram

Figure 9-1 on page 155 provides a block diagram of the I^2C module.

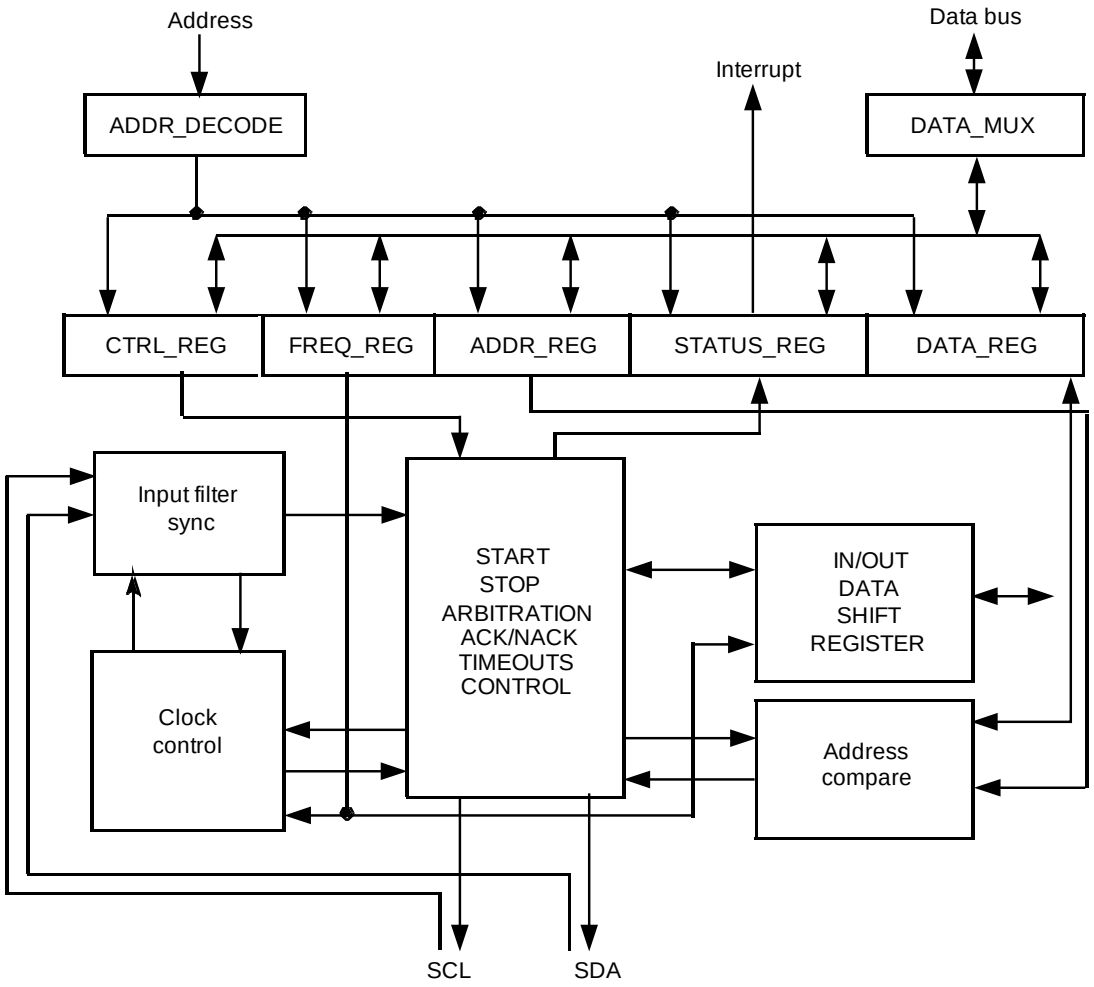


Figure 9-1. I²C functional block diagram

9.2 External signal description

This section describes each user-accessible pin signal.

9.2.1 Serial clock line

The bidirectional serial clock line (SCL) is the serial clock line of the I²C system.

9.2.2 Serial data line

The bidirectional serial data line (SDA) is the serial data line of the I²C system.

9.3 Register definitions

9.3.1 I²C memory map

The I²C has ten eight-bit registers. The base address of the module is hardware programmable. The I²C register map is fixed and begins at the module's base address. The following table summarizes the I²C module's address space. The following section describes the bit-level arrangement and functionality of each register.

Table 9-1. I²C memory map

Address	Register	Access	Reset	Details
Base + 0x0000	I²C Address Register 1 (IICA1)	Read/Write	0x00	page 157
Base + 0x0001	I²C Frequency Divider Register (IICF)	Read/Write	0x00	page 158
Base + 0x0002	I²C Control Register 1 (IICC1)	Read/Write	0x00	page 161
Base + 0x0003	I²C Status Register (IICS)	Read	0x80	page 163
Base + 0x0004	I²C Data IO Register (IICD)	Read/Write	0x00	page 165
Base + 0x0005	I²C Control Register 2 (IICC2)	Read/Write	0x00	page 166
Base + 0x0006	I²C input programmable filter (IICFLT)	Read/Write	0x00	page 167

This section consists of the I²C register descriptions in address order.

Refer to the direct-page register summary in the "Memory Maps" on page 45 for the absolute address assignments for all I²C registers. This section refers to registers and control bits only by their names. A Freescale-provided equate or header file is used to translate these names into the appropriate absolute addresses.

9.3.2 I²C register details

9.3.2.1 I²C Address Register 1

	7	6	5	4	3	2	1	0
Read	AD7	AD6	AD5	AD4	AD3	AD2	AD1	0
Write								
Reset	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 9-2. I²C Address Register 1 (IICA1)

Table 9-2. ²C Address Register 1 (IICA1) field descriptions

Bit(s)	Field	Description
7–1	AD[7:1]	Slave Address 1 The AD[7:1] field contains the slave address to be used by the I2C module. This field is used on the seven-bit address scheme and the lower seven bits of the 10-bit address scheme.
0	—	Reserved.

9.3.2.2 I²C Frequency Divider register

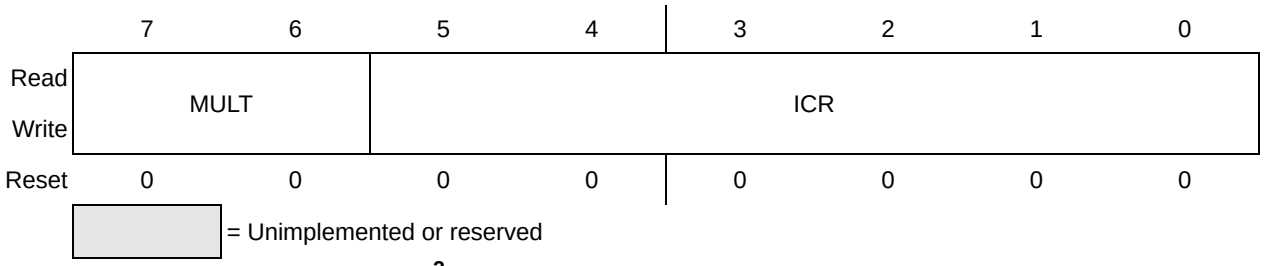


Figure 9-3. I²C Frequency Divider register (IICF)

Table 9-3. I²C Frequency Divider register (IICF) field descriptions

Bit(s)	Field	Description
7–6	MULT	**I²C** Multiplier Factor The MULT bits define the multiplier factor mul. This factor is used along with the SCL divider to generate the I²C baud rate. The multiplier factor mul as defined by the MULT bits is provided below. 00 mul = 01 01 mul = 02 10 mul = 04 11 Reserved
5–0	ICR	**I²C** Clock Rate The ICR bits are used to prescale the bus clock for bit rate selection. These bits and the MULT bits are used to determine the IIC baud rate, the SDA hold time, the SCL Start hold time and the SCL Stop hold time. Table 9-6 provides the SCL divider and hold values for corresponding values of the ICR. The SCL divider multiplied by multiplier factor mul is used to generate IIC baud rate. **I²C Baud Rate = Bus Speed (Hz)/(mul × SCL Divider)** ***Eqn. 9-1*** SDA hold time is the delay from the falling edge of SCL (IIC clock) to the changing of SDA (IIC data). **SDA Hold Time = Bus Period (s) × mul × SDA Hold Value** ***Eqn. 9-2*** SCL start hold time is the delay from the falling edge of SDA (IIC data) while SCL is high (Start condition) to the falling edge of SCL (IIC clock). **SCL Start Hold Time = Bus Period (s) × mul × SCL Start Hold Value** ***Eqn. 9-3*** SCL stop hold time is the delay from the rising edge of SCL (IIC clock) to the rising edge of SDA SDA (**IIC** data) while SCL is high (Stop condition). **SCL Stop Hold Time = Bus Period (s) × mul × SCL Stop Hold Value** ***Eqn. 9-4***

For example, if the bus speed is 8 MHz, Table 9-4 shows the possible hold time values with different ICR and MULT selections to achieve an I²C baud rate of 100 kbps.

Table 9-4. Example of IICF hold times

MULT	ICR	Hold times (μs)		
		SDA	SCL start	SCL stop
0x2	0x00	3.500	3.000	5.500
0x1	0x07	2.500	4.000	5.250
0x1	0x0B	2.250	4.000	5.250
0x0	0x14	2.125	4.250	5.125
0x0	0x18	1.125	4.750	5.125

Table 9-5. I^2C divider and hold values

ICR (hex)	SCL divider	SDA hold value	SCL hold (start) value	SCL Hold (stop) value
00	20	7	6	11
01	22	7	7	12
02	24	8	8	13
03	26	8	9	14
04	28	9	10	15
05	30	9	11	16
06	34	10	13	18
07	40	10	16	21
08	28	7	10	15
09	32	7	12	17
0A	36	9	14	19
0B	40	9	16	21
0C	44	11	18	23
0D	48	11	20	25
0C	44	11	18	23
0D	48	11	20	25
0E	56	13	24	29
0F	68	13	30	35
10	48	9	18	25
11	56	9	22	29

ICR (hex)	SCL divider	SDA hold value	SCL hold (start) value	SCL Hold (stop) value
20	160	17	78	81
21	192	17	94	97
22	224	33	110	113
23	256	33	126	129
24	288	49	142	145
25	320	49	158	161
26	384	65	190	193
27	480	65	238	241
28	320	33	158	161
29	384	33	190	193
2A	448	65	222	225
2B	512	65	254	257
2C	576	97	286	289
2D	640	97	318	321
2C	576	97	286	289
2D	640	97	318	321
2E	768	129	382	385
2F	960	129	478	481
30	640	65	318	321
31	768	65	382	385

Table 9-5. I^2C divider and hold values (continued)

ICR (hex)	SCL divider	SDA hold value	SCL hold (start) value	SCL Hold (stop) value
12	64	13	26	33
13	72	13	30	37
14	80	17	34	41
15	88	17	38	45
16	104	21	46	33
17	128	21	58	65
18	80	9	38	41
19	96	9	46	49
1A	112	17	54	57
1B	128	17	62	65
1C	144	25	70	73
1D	160	25	78	81
1E	192	33	94	97
1F	240	33	118	121

ICR (hex)	SCL divider	SDA hold value	SCL hold (start) value	SCL Hold (stop) value
32	896	129	446	449
33	1024	129	510	513
34	1152	193	574	577
35	1280	193	638	641
36	1536	257	766	769
37	1920	257	958	961
38	1280	129	638	641
39	1536	129	766	769
3A	1792	257	894	897
3B	2048	257	894	897
3C	2304	385	1150	1153
3D	2560	385	1278	1281
3e	3072	513	1534	1537
3f	3840	513	1918	1921

9.3.2.3 I²C Control register

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

= Unimplemented or Reserved

Figure 9-4. I²C Control register (IICC1)

Table 9-6. IICC1 I²C Control register (IICC1) descriptions

Bit(s)	Field	Description
7	IICEN	I2C Enable The IICEN bit determines whether the I²C module is enabled. 0 I²C is not enabled. 1 I²C is enabled.
6	IICIE	I2C Interrupt Enable The IICIE bit determines whether an I²C interrupt is requested. 0 I²C interrupt request not enabled. 1 I²C interrupt request enabled.
5	MST	Master Mode Select When the MST bit is changed from 0 to 1, a START signal is generated on the bus and master mode is selected. When this bit changes from 1 to 0 a STOP signal is generated and the mode of operation changes from master to slave. 0 Slave mode. 1 Master mode.
4	TX	Transmit Mode Select The TX bit selects the direction of master and slave transfers. In master mode this bit must be set according to the type of transfer required. Therefore, for address cycles, this bit is always high. When addressed as a slave this bit must be set by software according to the SRW bit in the status register. 0 Receive. 1 Transmit.
3	TXAK	Transmit Acknowledge Enable This bit specifies the value driven onto the SDA during data acknowledge cycles for both master and slave receivers. There are two conditions that effect NACK/ACK generation. **Note:** SCL is held to low until TXAK is written. • If FACK (fast NACK/ACK) is cleared: 0 An acknowledge signal is sent to the bus on the following receiving data byte. 1 No acknowledge signal response is sent to the bus on the following receiving data byte. • If FACK bit is set. no ACK or NACK is sent after receiving one data byte until this TXAK bit is written: 0 An acknowledge signal is sent out to the bus on the current receiving data byte 1 No acknowledge signal response is sent to the bus on the current receiving data byte

Table 9-6. IICC1 I²C Control register (IICC1) descriptions (continued)

Bit(s)	Field	Description
2	RSTA (Write only, Read always 0)	Repeat START Writing 1 to this bit generates a repeated START condition provided it is the current master. Attempting a repeat at the wrong time results in loss of arbitration. 0 No repeat start detected in bus operation. 1 Repeat start generated.
1	WUEN	Wake-up Enable I²C can wake the MCU from stop mode when slave address or general call address matching occurs. 0 Normal operation. No interrupt generated when address matching in stop mode. 1 Enables the wake-up function in stop mode.
0	—	Reserved.

9.3.2.4 I²C Status register

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

= Unimplemented or Reserved

Figure 9-5. I²C Status Register (IICS)

Table 9-7. ²C Status Register (IICS) field descriptions

Bit(s)	Field	Description
7	TCF	Transfer Complete Flag This bit is set on the completion of a byte and acknowledge bit transfer. This bit is only valid during or immediately following a transfer to the I²C module or from the I²C module.The TCF bit is cleared by reading the IICD register in receive mode or writing to the IICD in transmit mode. 0 Transfer in progress. 1 Transfer complete.
6	IAAS	Addressed as a Slave The IAAS bit is set when one of the following conditions is met: • When the calling address matches the programmed slave address • If the GCAEN bit is set and a general call is received • If SIICAEN bit is set, when the calling address matches the 2nd programmed slave address • If ALERTEN bit is set and SMBus alert response address is received This bit is set before ACK bit. The CPU needs to check the SRW bit and set TX/RX bit accordingly. Writing the IICC1 register with any value clears this bit. 0 Not addressed. 1 Addressed as a slave.
5	BUSY	Bus Busy The BUSY bit indicates the status of the bus regardless of slave or master mode. The BUSY bit is set when a START signal is detected and cleared when a STOP signal is detected. 0 Bus is idle. 1 Bus is busy.
4	ARBL	Arbitration Lost This bit is set by hardware when the arbitration procedure is lost. The ARBL bit must be cleared by software or by writing 1 to it. 0 Standard bus operation. 1 Loss of arbitration.
3	—	Reserved.

Table 9-7. ^{2}C Status Register (IICS) field descriptions (continued)

Bit(s)	Field	Description
2	SRW	Slave Read/Write When addressed as a slave, the SRW bit indicates the value of the R/$\overline{W}$ command bit of the calling address sent to the master. 0 Slave receive, master writing to slave. 1 Slave transmit, master reading from slave.
1	IICIF	I^2C Interrupt Flag The I^2CIF bit is set when an interrupt is pending. This bit must be cleared by software or by writing a 1 to it in the interrupt routine. One of the following events can set the I^2CIF bit: • One byte transfer including ACK/NACK bit completes if FACK = 0 • One byte transfer excluding ACK/NCAK bit completes if FACK = 1. an ACK or NACK is sent on the bus by writing 0 or 1 to TXAK after this bit is set as receive mode. • Match of slave addresses to calling address including primary slave address, general call address, alert response address, and second slave address. (When address matching happens in stop mode, the I^2CIF is cleared automatically after core gets out of STOP.) • Arbitration lost • Time-outs in SMBus mode except both SCL and SDA high time-out 0 No interrupt pending. 1 Interrupt pending.
0	RXAK	Receive Acknowledge When the RXAK bit is low, it indicates an acknowledge signal has been received after the completion of one byte of data transmission on the bus. If the RXAK bit is high it means that no acknowledge signal is detected. 0 Acknowledge received. 1 No acknowledge received.

9.3.2.5 I²C Data I/O register

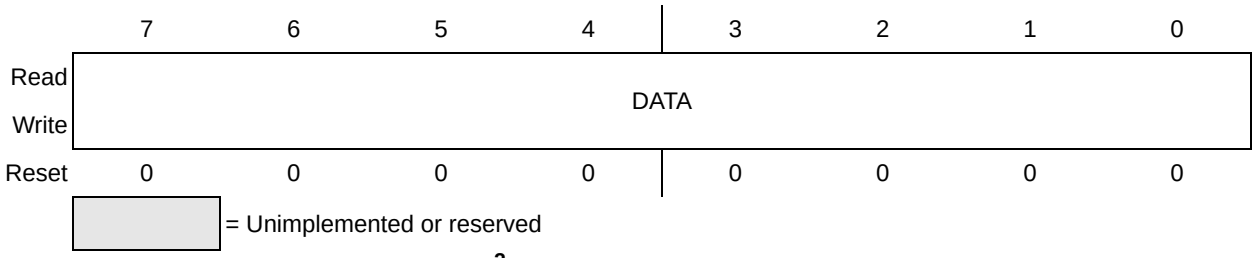


Figure 9-6. I²C Data I/O register (IICD)

Table 9-8. ²C Data I/O register (IICD) field descriptions

Bit(s)	Field	Description
7–0	DATA	Data In master transmit mode, when data is written to the IICD, a data transfer is initiated. The most significant bit is sent first. In master receive mode, reading this register initiates receiving of the next byte of data.

NOTE

When transitioning out of master receive mode, the I²C mode must be switched before reading the IICD register to prevent an inadvertent initiation of a master receive data transfer.

In slave mode, the same functions are available after an address match has occurred.

TX bit in IICC must correctly reflect the desired direction of transfer in master and slave modes for the transmission to begin. For instance, if the I²C is configured for master transmit but a master receive is desired, then reading the IICD does not initiate the receive.

Reading the IICD returns the last byte received while the I²C is configured in either master receive or slave receive modes. The IICD does not reflect every byte that is transmitted on the I²C bus, nor can software verify that a byte has been written to the IICD correctly by reading it back.

In master transmit mode, the first byte of data written to IICD following assertion of MST (start bit) or assertion of RSTA bit (repeated start) is used for the address transfer and must comprise of the calling address (in bit 7 to bit 1) concatenated with the required R/$\overline{W}$ bit (in position bit 0).

9.3.2.6 I²C Control Register 2

	7	6	5	4	3	2	1	0
Read	GCAEN	ADEXT	0	0	0	AD10	AD9	AD8
Write								
Reset	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 9-7. I²C Control Register 2 (IICC2)

Table 9-9. I²C Control Register 2 (IICC2) field descriptions

Bit(s)	Field	Description
7	GCAEN	General Call Address Enable The GCAEN bit enables or disables general call address. 0 General call address is disabled. 1 General call address is enabled.
6	ADEXT	Address Extension The ADEXT bit controls the number of bits used for the slave address. 0 7-bit address scheme. 1 10-bit address scheme.
5–3	—	Reserved.
2–0	AD[10:8]	Slave Address The AD[10:8] field contains the upper three bits of the slave address in the 10-bit address scheme. This field is only valid when the ADEXT bit is set.

9.3.2.7 I²C Programmable Input Glitch Filter register

	7	6	5	4	3	2	1	0
Read	0	0	0	FLT4	FLT3	FLT2	FLT1	FLT0
Write								
Reset	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 9-8. I²C Programmable Input Glitch Filter register (IICFLT)

Table 9-10. I²C Programmable Input Glitch Filter register (IICFLT) Field Descriptions

Bit(s)	Field	Description
7–5	—	Reserved.
4–0	FLT	I²C Programmable Filter Factor Contains the programming controls for the width of glitch (in terms of bus clock cycles) the filter must absorb; in other words, the filter does not let glitches less than or equal to this width setting pass. For instance: FLT[3:0] 0000 No Filter/Bypass 0001 Filter glitches up to width of 1 (half) IPBUS clock cycle 0010 Filter glitches up to width of 2 (half) IPBUS clock cycles 0011 Filter glitches up to width of 3 (half) IPBUS clock cycles 0100 Filter glitches up to width of 4 (half) IPBUS clock cycles 0101 Filter glitches up to width of 5 (half) IPBUS clock cycles 0110 Filter glitches up to width of 6 (half) IPBUS clock cycles 0111 Filter glitches up to width of 7 (half) IPBUS clock cycles 1000 Filter glitches up to width of 8 (half) IPBUS clock cycles 1001 Filter glitches up to width of 9 (half) IPBUS clock cycles 1010 Filter glitches up to width of 10 (half) IPBUS clock cycles 1011 Filter glitches up to width of 11 (half) IPBUS clock cycles 1100 Filter glitches up to width of 12 (half) IPBUS clock cycles 1101 Filter glitches up to width of 13 (half) IPBUS clock cycles 1110 Filter glitches up to width of 14 (half) IPBUS clock cycles 1111 Filter glitches up to width of 15 (half) IPBUS clock cycles

9.4 Functional description

This section provides a complete functional description of the I^{2}C module.

9.4.1 I^{2}C protocol

The I^{2}C bus system uses a Serial Data Line (SDA) and a Serial Clock Line (SCL) for data transfer. All devices connected to it must have open drain or open collector outputs. A logic AND function is exercised on both lines with external pull-up resistors. The value of these resistors is system dependent.

Normally, a standard communication is composed of four parts:

- START signal
- Slave address transmission
- Data transfer
- STOP signal

The STOP signal should not be confused with the CPU STOP instruction. The I^{2}C bus system communication is described briefly in the following sections and is illustrated in Figure 9-9.

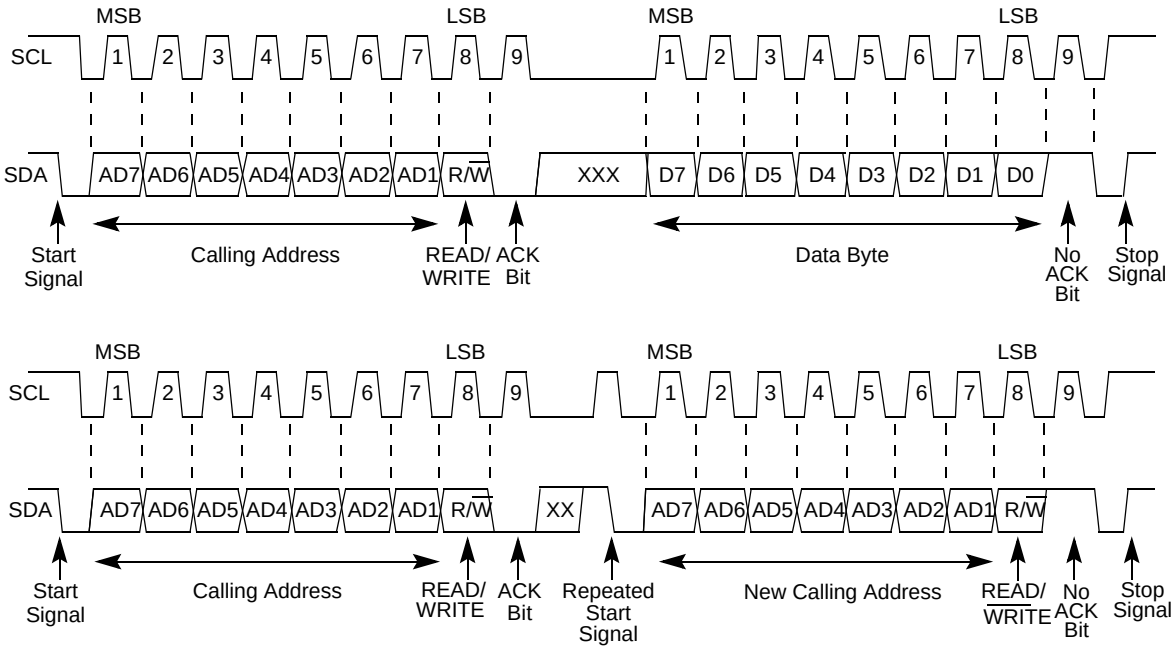


Figure 9-9. I^{2}C bus transmission signals

9.4.1.1 START signal

When the bus is free that is, no master device is engaging the bus (both SCL and SDA lines are at logical high), a master may initiate communication by sending a START signal. As shown in Figure 9-9, a START signal is defined as a high-to-low transition of SDA while SCL is high. This signal denotes the beginning

of a new data transfer (each data transfer may contain several bytes of data) and brings all slaves out of their idle states.

9.4.1.2 Slave address transmission

The first byte of data transferred immediately after the START signal is the slave address transmitted by the master. This is a seven-bit calling address followed by a R/$\overline{W}$ 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 back an acknowledge bit. This is done by pulling the SDA low at the 9th clock (see Figure 9-9 on page 168).

No two slaves in the system may have the same address. If the I^2C module is the master, it must not transmit an address that is equal to its own slave address. The I^2C cannot be master and slave at the same time. However, if arbitration is lost during an address cycle, the I^2C reverts to slave mode and operates correctly even if it is being addressed by another master.

9.4.1.3 Data transfer

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

All transfers that come after an address cycle are referred to as data transfers, even if they carry sub-address information for the slave device.

Each data byte is 8 bits long. Data may be changed only while SCL is low and must be held stable while SCL is high as shown in Figure 9-9 on page 168. There is one clock pulse on SCL for each data bit, the MSB being transferred first. Each data byte is followed by a 9th (acknowledge) bit that is signalled from the receiving device. An acknowledge is signalled by pulling the SDA low at the ninth clock. In summary, one complete data transfer needs nine clock pulses.

If the slave receiver does not acknowledge the master in the 9th bit time, the SDA line must be left high by the slave. The master interprets the failed acknowledge as an unsuccessful data transfer.

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

In either case, the data transfer is aborted and the master does one of two things:

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

9.4.1.4 STOP signal

The master can terminate the communication by generating a STOP signal to free the bus. However, the master may generate a START signal followed by a calling command without generating a STOP signal first. This is called repeated START. A STOP signal is defined as a low-to-high transition of SDA while SCL at logical 1 (see Figure 9-9 on page 168).

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

9.4.1.5 Repeated START signal

As shown in Figure 9-9 on page 168, a repeated START signal is a START signal generated without first generating a STOP signal to terminate the communication. This is used by the master to communicate with another slave or with the same slave in different mode (transmit/receive mode) without releasing the bus.

9.4.1.6 Arbitration procedure

The I^2C bus is a true multi-master bus that allows more than one master to be connected on it. If two or more masters try to control the bus at the same time, a clock synchronization procedure determines the bus clock for which the low period is equal to the longest clock low period and the high is equal to the shortest one among the masters. The relative priority of the contending masters is determined by a data arbitration procedure, a bus master loses arbitration if it transmits logic 1 while another master transmits logic 0. The losing masters immediately switch over to slave-receive mode and stop driving SDA output. In this case, the transition from master to slave mode does not generate a STOP condition. Meanwhile, a status bit is set by hardware to indicate loss of arbitration.

9.4.1.7 Clock synchronization

Because wire *and* logic is performed on the SCL line, a high-to-low transition on the SCL line affects all the devices connected on the bus. The devices start counting their low period and after a device's clock has gone low, it holds the SCL line low until the clock high state is reached. However, the change of low to high in this device clock may not change the state of the SCL line if another device clock is still within its low period. Therefore, synchronized clock SCL is held low by the device with the longest low period.

Devices with shorter low periods enter a high wait state during this time. (See Figure 9-10 on page 171.) When all devices concerned have counted off their low period, the synchronized clock SCL line is released and pulled high. There is then no difference between the device clocks and the state of the SCL line and all the devices start counting their high periods. The first device to complete its high period pulls the SCL line low again.

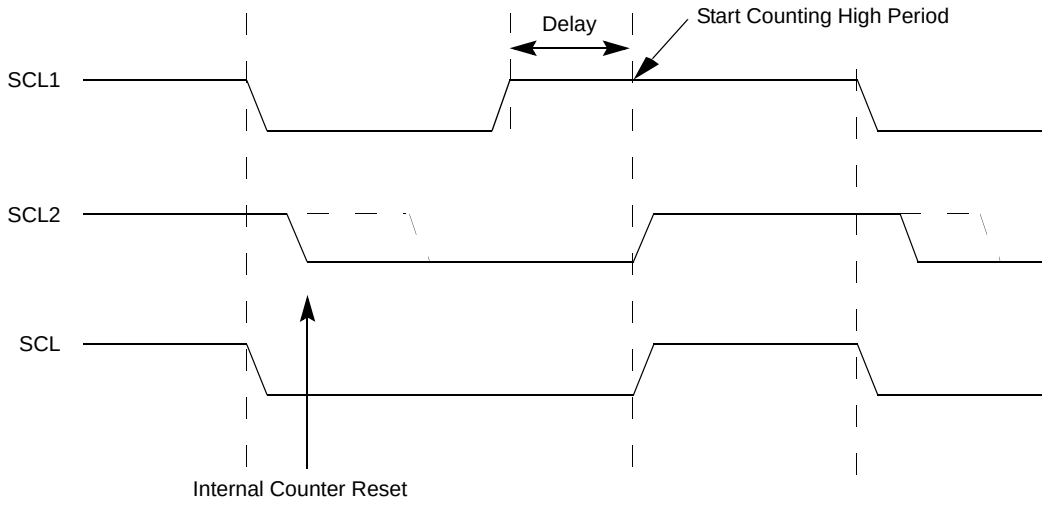


Figure 9-10. I²C clock synchronization

9.4.1.8 Handshaking

The clock synchronization mechanism can be used as a handshake in data transfer. Slave devices may hold the SCL low after completion of one byte transfer (nine bits). In such case, it halts the bus clock and forces the master clock into wait states until the slave releases the SCL line.

9.4.1.9 Clock stretching

The clock synchronization mechanism can be used by slaves to slow down the bit rate of a transfer. After the master has driven SCL low, the slave can drive SCL low for the required period and then release it. If the slave SCL low period is greater than the master SCL low period then the resulting SCL bus signal low period is stretched.

9.4.2 10-bit address

For 10-bit addressing, 0x11110 is used for the first five bits of the first address byte. Various combinations of read/write formats are possible within a transfer that includes 10-bit addressing.

9.4.2.1 Master-transmitter addresses a slave-receiver

The transfer direction is not changed (see Table 9-11). When a 10-bit address follows a START condition, each slave compares the first seven bits of the first byte of the slave address (11110XX) with its own address and tests whether the eighth bit (R/$\overline{W}$ 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 eight bits of the second byte of the slave address with its own address, but only one slave finds a match and generate an acknowledge (A2). The matching slave remains addressed by the master until it receives a STOP condition (P) or a repeated START condition (Sr) followed by a different slave address.

Table 9-11. Master-transmitter addresses slave-receiver with a 10-bit address

S	Slave Address first seven bits 11110 + AD10 + AD9	R/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 I^2C interrupt. User software must ensure that for this interrupt, the contents of IICD are ignored and not treated as valid data.

9.4.2.2 Master-receiver addresses a slave-transmitter

The transfer direction is changed after the second R/$\overline{W}$ bit. (See Table 9-12.) Up to and including Acknowledge Bit A2, the procedure is the same as that described for a master-transmitter addressing a slave-receiver. After the repeated START condition (Sr), a matching slave remembers that it was addressed before. This slave then checks whether the first seven bits of the first byte of the slave address following the Sr are the same as they were after the START condition (S) and 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 seven-bit devices) does not match.

Table 9-12. Master-Receiver Addresses a Slave-Transmitter with a 10-Bit Address

S	Slave Address First seven bits 11110 + AD10 + AD9	R/W 0	A1	Slave Address Second byte AD[8:1]	A2	Sr	Slave Address First seven bits 11110 + AD10 + AD9	R/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 I^2C interrupt. User software must ensure that for this interrupt, the contents of IICD are ignored and not treated as valid data.

9.4.3 Address matching

All received addresses can be requested in seven-bit or 10-bit address. I^2C address register 1, that contains the I^2C primary slave address, always participates in the address-matching process. If the GCAEN bit is set, general call participates the address matching process. When the I^2C module responds to one of the above mentioned address, it acts as a slave-receiver and the IAAS bit is set after the address cycle.

Software needs to read the IICD register, after the first byte transfer, to determine that the address is matched.

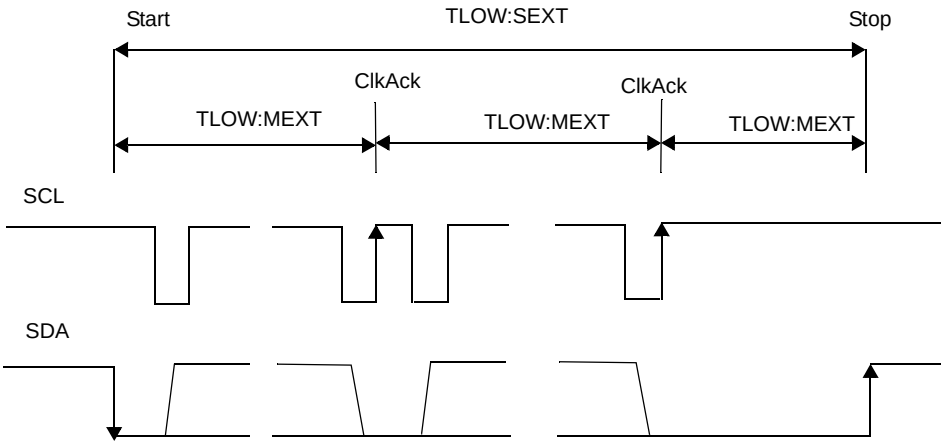


Figure 9-11. I^2C address matching

9.5 Resets

The I^2C is disabled after reset. The I^2C cannot cause an MCU reset.

9.6 Interrupts

The I^2C generates a single interrupt.

An interrupt from the I^2C module is generated when any of the events in Table 9-13 occur, provided the IICIE bit is set. The interrupt is driven by bit IICIF (of the I^2C status register) and masked with bit IICIE (of the I^2C control register). The IICIF bit must be cleared by software by writing a 1 to it in the interrupt routine.

You can determine the interrupt type by reading the status register. For SMBus time-outs interrupt, the interrupt is driven by SLTF and masked with bit IICIE. The SLTF 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.

NOTE

In master receive mode, the FACK must be set to 0 before the last byte transfer.

Table 9-13. Interrupt summary

Interrupt Source	Status	Flag	Local Enable
Complete one-byte transfer	TCF	IICIF	IICIE
Match of received calling address	IAAS	IICIF	IICIE
Arbitration lost	ARBL	IICIF	IICIE
Wake-up form sop3 interrupt	IAAS	IICIF	IICIE and WUEN

9.6.1 Byte-transfer interrupt

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

9.6.2 Address-detect interrupt

When the calling address matches the programmed slave address (I^2C address register) or when the GCAEN bit is set and a general call is received, the IAAS bit in the status register is set.

The CPU is interrupted, provided the IICIE is set. The CPU must check the SRW bit and set its Tx mode accordingly.

9.6.3 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.

9.6.4 Arbitration-lost interrupt

The I^2C is a true multi-master bus that allows more than one master to be connected on it. If two or more masters try to control the bus at the same time, the relative priority of the contending masters is determined by a data arbitration procedure. The I^2C module asserts this interrupt when it loses the data arbitration process and the ARBL bit in the status register is set.

Arbitration is lost in the following circumstances:

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

This bit must be cleared by software by writing a 1 to it.

9.6.5 Programmable, input-glitch filter

An I^2C glitch filter has been added outside the I^2C legacy modules, but within the I^2C package. This filter can absorb glitches on the I^2C clock and data lines for the I^2C module. The width of the glitch to absorb can be specified in terms of the number of (half) bus clock cycles.

A single glitch filter control register is provided as IICFLT. Effectively, any down-up-down or up-down-up transition on the data line that occurs within the number of clock cycles programmed here is ignored by the I^2C. The programmer only needs to specify the size of glitch (in terms of bus clock cycles) for the filter to absorb and not pass it.

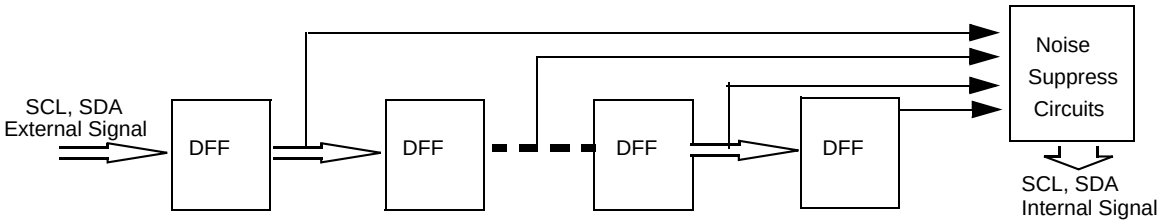


Figure 9-12. Programmable, input-glitch filter

9.6.6 Address-matching wakeup

When address-matching happens as I^2C works in Slave Receive mode, the MCU wakes from Stop mode. After the address-matching IAAS bit is set, an interrupt is sent out at the end of address-matching to wake up the MCU.

The ACK will also be sent from slave if address is correctly matched. The IAAS bit must be cleared after the clock recovery.

NOTE

After the system was recovered to Run mode, the I^2C module must restart if it is needed to work. The SCL line will not be hold low until the I^2C module resets after address-matching.

9.7 Initialization/application information

Module initialization (slave)

1. Write: IICC2
   — To enable or disable general call
   — To select 10-bit or seven-bit addressing mode
2. Write: IICA1
   — To set the slave address
3. Write: IICC1
   — To enable I^2C and interrupts
4. Initialize RAM variables (IICEN = 1 and IICIE = 1) for transmit data
5. Initialize RAM variables used to achieve the routine shown in Figure 9-13 on page 178

Module initialization (master)

1. Write: IICF
   — To set the I^2C baud rate (example provided in this chapter)
2. Write: IICC1
   — To enable I^2C and interrupts
3. Initialize RAM variables (IICEN = 1 and IICIE = 1) for transmitting data
4. Initialize RAM variables used to achieve the routine shown in Figure 9-13 on page 178
5. Write: IICC1
   — To enable TX
6. Write: IICC1
   — To enable MST (Master mode)
7. Write: IICD
   — With the address of the target slave. (The LSB of this byte determines whether the communication is master receive or transmit.)

Module use

The routine shown in Figure 9-13 on page 178 can handle both master and slave I^2C operations. For slave operation, an incoming I^2C message that contains the proper address begins I^2C communication. For master operation, communication must be initiated by writing to the IICD register.

Register model

IICA1	AD[7:1]							0

Address to which the module responds when addressed as a slave (in Slave mode)

IICF	MULT	ICR

Baud rate = BUSCLK/(2 x MULT x (SCL DIVIDER))

IICC1	IICEN	IICIE	MST	TX	TXAK	RSTA	WUEN	0

Module configuration

IICS	TCF	IAAS	BUSY	ARBL	0	SRW	IICIF	RXAK

Module status flags

IICD	DATA

Data register; Write to transmit I^2C data read to read I^2C data

IICC2	GCAEN	ADEXT	0	0	0	AD10	AD9	AD8

Address configuration

IICFLT	0	0	0	O	FLT3	FLT2	FLT1	FLT0

I^2C Programmable Input Glitch Filter

IICSMB	FACK	ALERTEN	SIICAEN	TCKSEL	SLTF	SHTF1	SHTF2	SHTF2IE

I^2C SMBus Control and Status Register

IICA2	SAD[7:1]	0

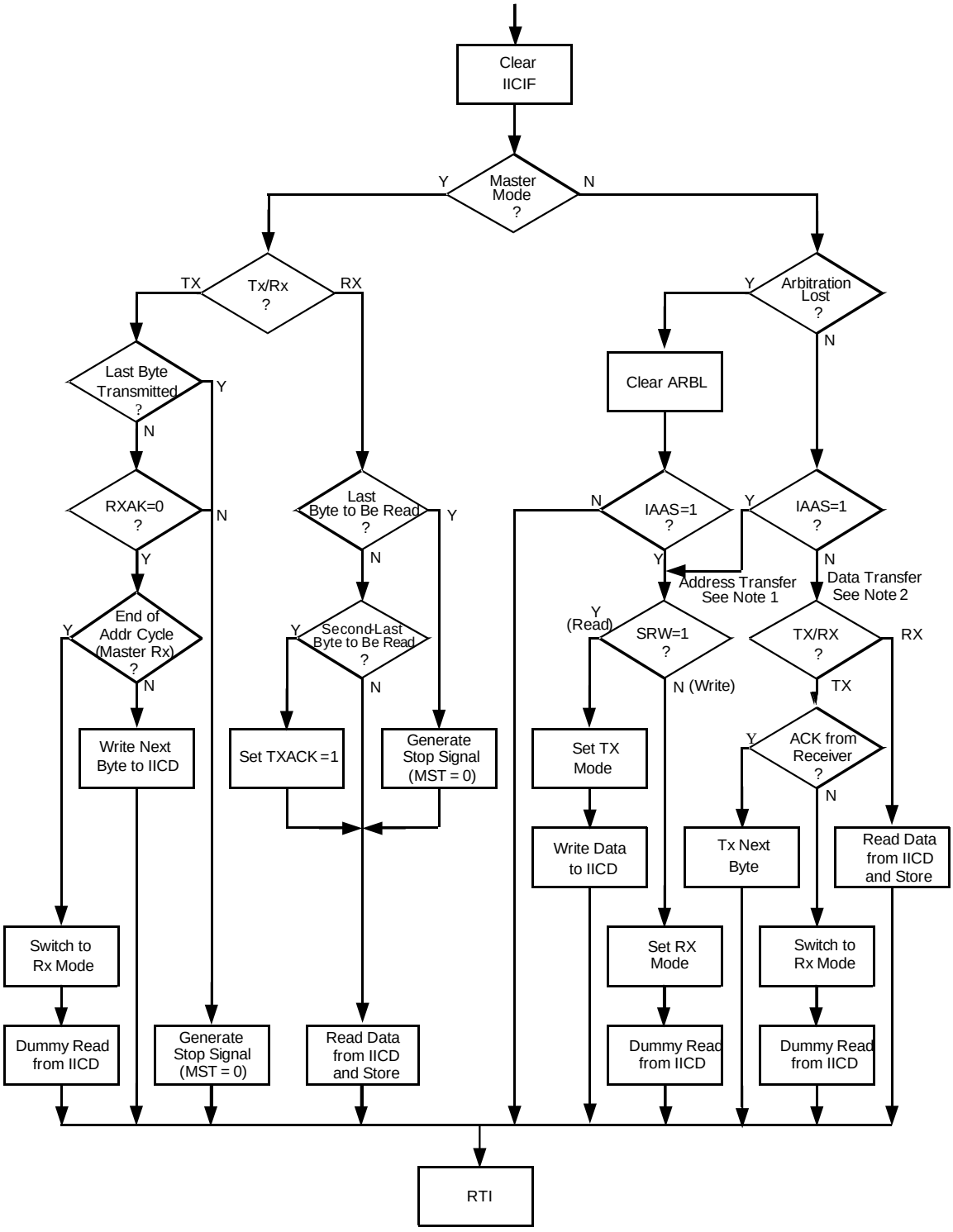
I^2C Address Register 2

IICSLTH	SSLT[15:8]

I^2C SCL Low Time Out Register High

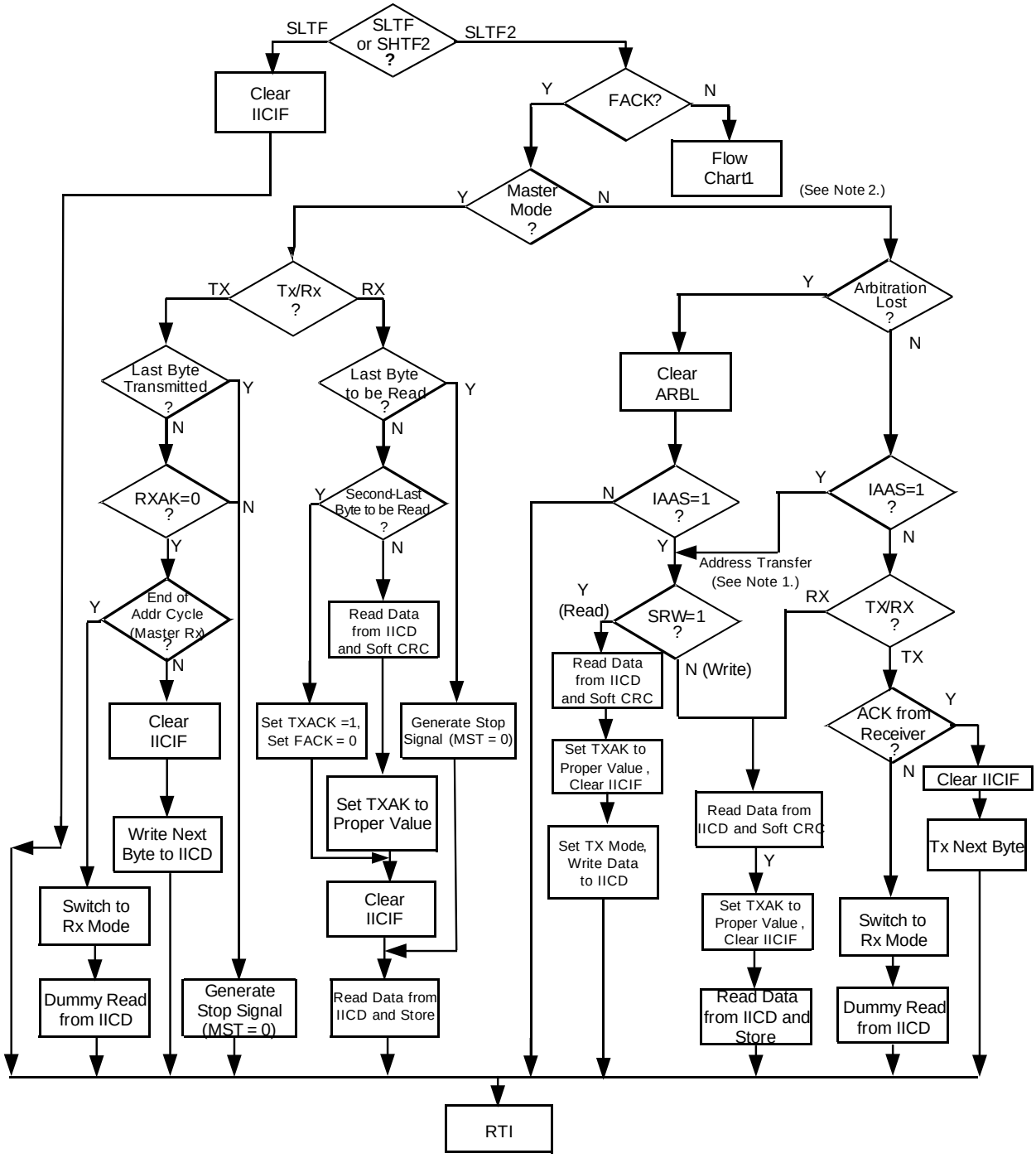
IICSLTL	SSLT[7:0]

I^2C SCL Low Time Out Register Low



NOTES:
1. If general call is enabled, a check must be done to determine whether the received address was a general call address (0x00). If the received address was a general call address, then the general call must be handled by user software.
2. When 10-bit addressing is used to address a slave, the slave sees an interrupt following the first byte of the extended address. User software must ensure that for this interrupt, the contents of IICD are ignored and not treated as a valid data transfer.

Figure 9-13. Typical I^2C Interrupt Routine



NOTES:
1. If a general call or SIICAEN is enabled, a check must be done to determine whether the received address was a general-call address (0x00) or a SMbus-device, default address. If the received address was one of them, it must be handled by user software.
2. Flow chart 1 means Figure 9-13 on page 178. Typical I^2C Interrupt Routine.
3. As receive side the second data reading should be done after ninth SCL cycle.

Figure 9-14. Typical I^2C SMBus interrupt routine

Chapter 10 Analog Front End

10.1 Introduction

The Analog Front End (AFE) includes the accelerometer, signal conditioning blocks, off-chip analog inputs, analog multiplexor and an analog-to-digital converter 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 Features

- Up to 3906 conversion frames per second support for a 3906-Hz output data rate
  (A higher data rate is available with the MMA9559L device.)
- 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 allow 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[1]
- External differential ADC input range: 0.65V ± 0.45V

1. The temperature sensor is designed for a 200 °C, full-scale range. This does not imply that the device is intended to operate over this entire range. This device is guaranteed to operate over the commercial (–40 to 85 °C) temperature range.

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 1.8 V span, from -0.9 V to +0.9 V 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.

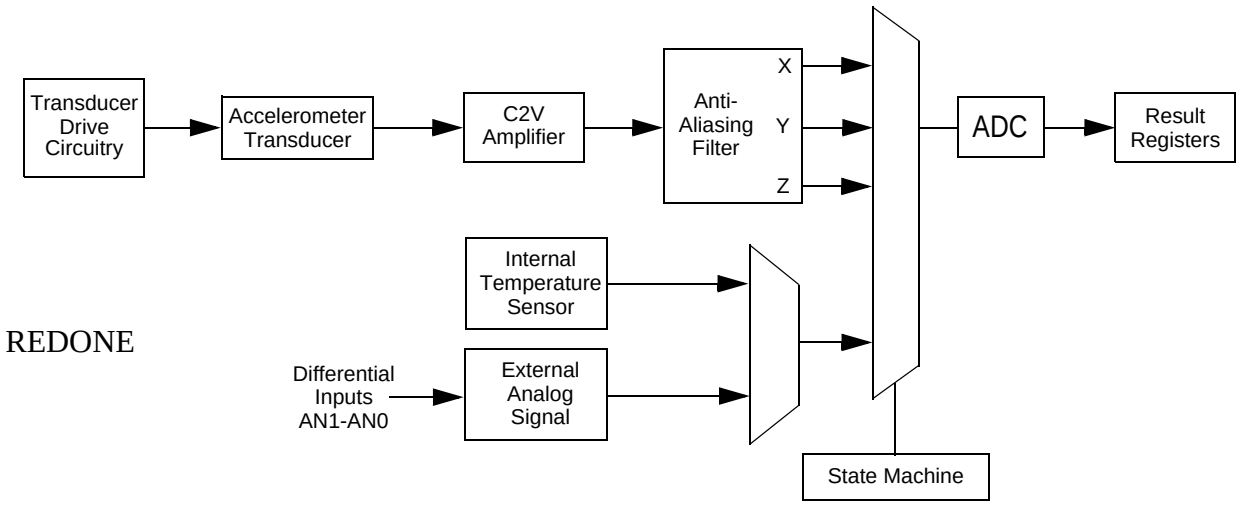


Figure 10-1. AFE data flow

The ADC contains a number of options to trade off power consumption versus resolution and accuracy.

Table 10-1 shows how resolution can be traded for reduced analog-phase duration. The options are lower consumption or more time 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 will 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, 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[1]	ADC number of bits	Analog phase[2] (μs)	
	N	Three samples	Four samples
11	10	135	154
10	12	159	186
01	14	207	250
00	16	303	378

1 See the AFE Control and Status Register (AFE_CSR), Table 10-3.

2 Nominal Values

Besides the ADC resolution, the Full Scale range of the Analog Front End line-up 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 per LSB associated to a given range. The appropriate scaling is realized by the AFE Software Routine that are further described in the *MMA955xL Intelligent, Motion-Sensing Platform Software Reference Manual* (MMA955XLSWRM).

Table 10-2. AFE scaling selection

FS[1]	Full-scale data range	XYZ acceleration scaling[2] (mg/LSB)
00	+/-8g	0.244
01	+/-2g	0.061
10	+/-4g	0.122
11	+/-8g	0.244

1 See the AFE Control and Status Register (AFE_CSR), Table 10-3.

2 Nominal values, once Freescale trim algorithms have been run on the data.

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 changes.

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 signals conditions and filters the signal, providing a high-level output voltage that is ratiometric and proportional to acceleration.

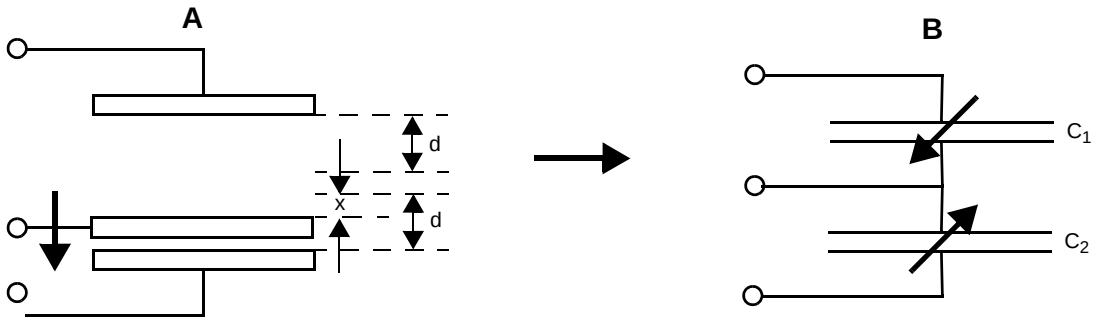


Figure 10-2. Transducer physical model (A) and equivalent circuit model (B)

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.[1]

1. Actual waveforms may vary in the actual application, but the theory presented here is still applicable from a high-level perspective.

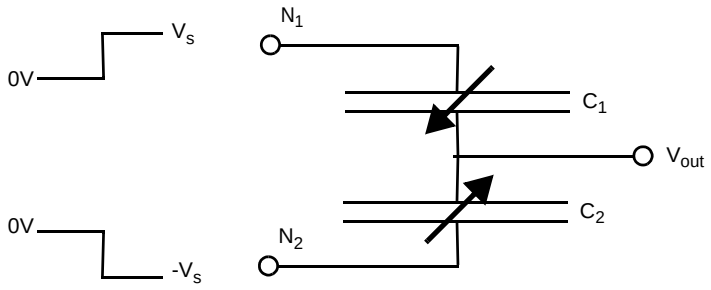


Figure 10-3. Dividers N_1 and N_2

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:

$$V_{out} = -V_s + [\ 2\ V_s\ C_1\ /\ (C_1 + C_2)\ ] \qquad \textit{Eqn. 10-1}$$

$$V_{out} = [\ -\ (C_1 + C_2)\ V_s\ + 2\ V_s\ C_1\ ]\ /\ (C_1 + C_2) \qquad \textit{Eqn. 10-2}$$

$$V_{out} = [\ -\ V_s\ C_2\ +\ V_s\ C_1\ ]\ /\ (C_1 + C_2) \qquad \textit{Eqn. 10-3}$$

$$V_{out} = V_s\ (C_1 - C_2)\ /\ (C_1 + C_2) \qquad \textit{Eqn. 10-4}$$

Capacitance theory

This can be converted 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, we begin by noting:

$$K = N\ A\ \varepsilon \qquad \textit{Eqn. 10-5}$$

$$C_1 = K\ /\ (d + x) \qquad \textit{Eqn. 10-6}$$

$$C_2 = K\ /\ (d - x) \qquad \textit{Eqn. 10-7}$$

$$C_0 = K\ /\ d \qquad \textit{Eqn. 10-8}$$

Where:

- N = The number of beams in the g-cell (unitless)
- A = The area of the facing side of the beam in meters^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

$$V_{out} = V_s\ (C_1 - C_2)\ /\ (C_1 + C_2) \qquad \textit{Eqn. 10-9}$$

$$V_{out} = V_s (K / (d + x) - K / (d - x)) / (K / (d + x) + K / (d - x))$$ *Eqn. 10-10*

$$V_{out} = V_s (1 / (d + x) - 1 / (d - x)) / (1 / (d + x) + 1 / (d - x))$$ *Eqn. 10-11*

$$V_s (((d - x) - (d + x)) / (d^2 - x^2)) / (((d + x) + (d - x)) / (d^2 - x^2))$$ *Eqn. 10-12*

$$V_{out} = V_s ((d - x) - (d + x)) / ((d + x) + (d - x))$$ *Eqn. 10-13*

$$V_{out} = V_s \frac{\frac{K}{(d + x)} - \frac{K}{(d - x)}}{\frac{K}{(d + x)} + \frac{K}{(d - x)}}$$ *Eqn. 10-14*

$$V_{out} = V_s \frac{\frac{(d - x) - (d + x)}{d^2 - x^2}}{\frac{(d - x) + (d + x)}{d^2 - x^2}}$$ *Eqn. 10-15*

$$V_{out} = -V_s \frac{x}{d}$$ *Eqn. 10-16*

$$V_{out} = -V_s (x / d)$$ *Eqn. 10-17*

Newton's Second Law

$$F = mass * Acceleration$$ *Eqn. 10-18*

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

$$F = -k * x$$ *Eqn. 10-19*

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 :

$$x = mass * Acceleration / -k$$ *Eqn. 10-20*

Finally

Replace x in $V_{out} = – V_s$ using the expression above produces:

$$V_{out} = V_s (mass * Acceleration) / (k * d)$$ *Eqn. 10-21*

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)
- 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 Memory map overview

NOTE

The AFE Register set can be accessed (both READ and WRITE) from Supervisor Mode only. ADC output values are made available via reserved RAM locations once Freescale trim algorithms have been run on the data. For further details, see Application Identifier 0x06 (XYZ_DATA_FBID) in the *MMA955xL Intelligent, Motion-Sensing Platform Software Reference Manual* (MMA955XLSWRM).

Table 10-3 summarizes the few specific fields of the AFE_CSR register that can be updated by the AFE application and how it is used to configure the AFE for the next conversion sequence.

Table 10-3. AFE_CSR register field descriptions

Field	Description
FS	**Full-scale selection** AFE_CSR[FS] control AFE Full Scale selection as per Table 10-2
CM	**Conversion mode** These bits control ADC resolution/accuracy versus power and conversion time trade-offs. Values available for end user applications are listed in Table 10-1.

Chapter 11 System Integration Module

11.1 Introduction

The System Integration Module (SIM) provides a central mechanism for managing:

- Reset generation
- Mode control
- Oscillator control
- Clock gating

Figure 11-1 on page 190 illustrates some of the major interactions with other on-chip components. These will be discussed in more detail in the sections that follow.

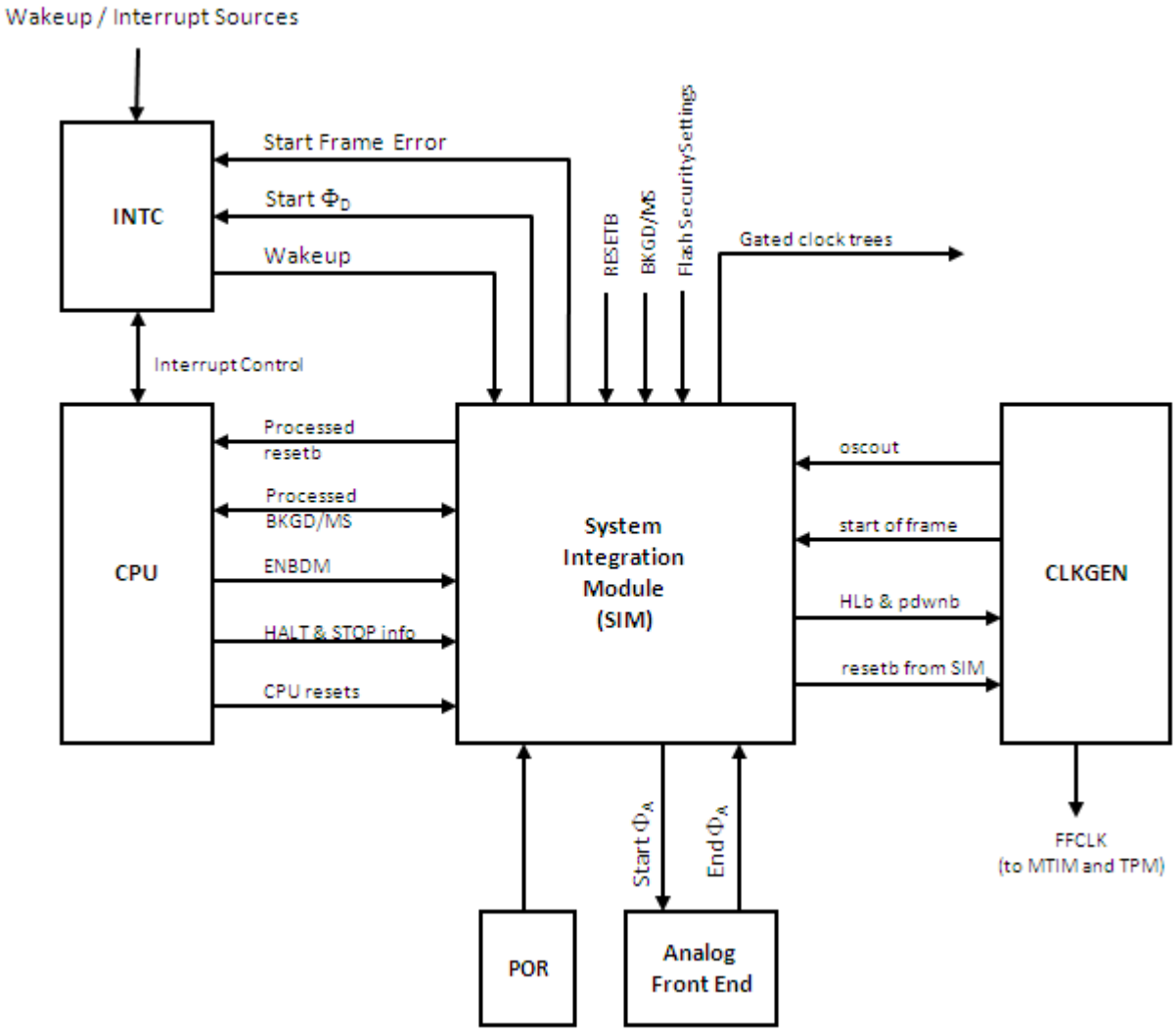


Figure 11-1. Major SIM interactions

11.2 Reset generation

11.2.1 Reset sources

There are a number of sources that may cause the device to reset itself:

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 Reset	The device can be reset via the Background debug port.
Software Reset	The CPU can be explicitly shut down by writing a 1 to the software reset bit of the SIM Control Register (RSCR[ASR]).
ILOP Reset	The CPU may request a reset in the event of an illegal operation.
ILAD Reset	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 11-2. 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]).

11.2.2 Reset outputs

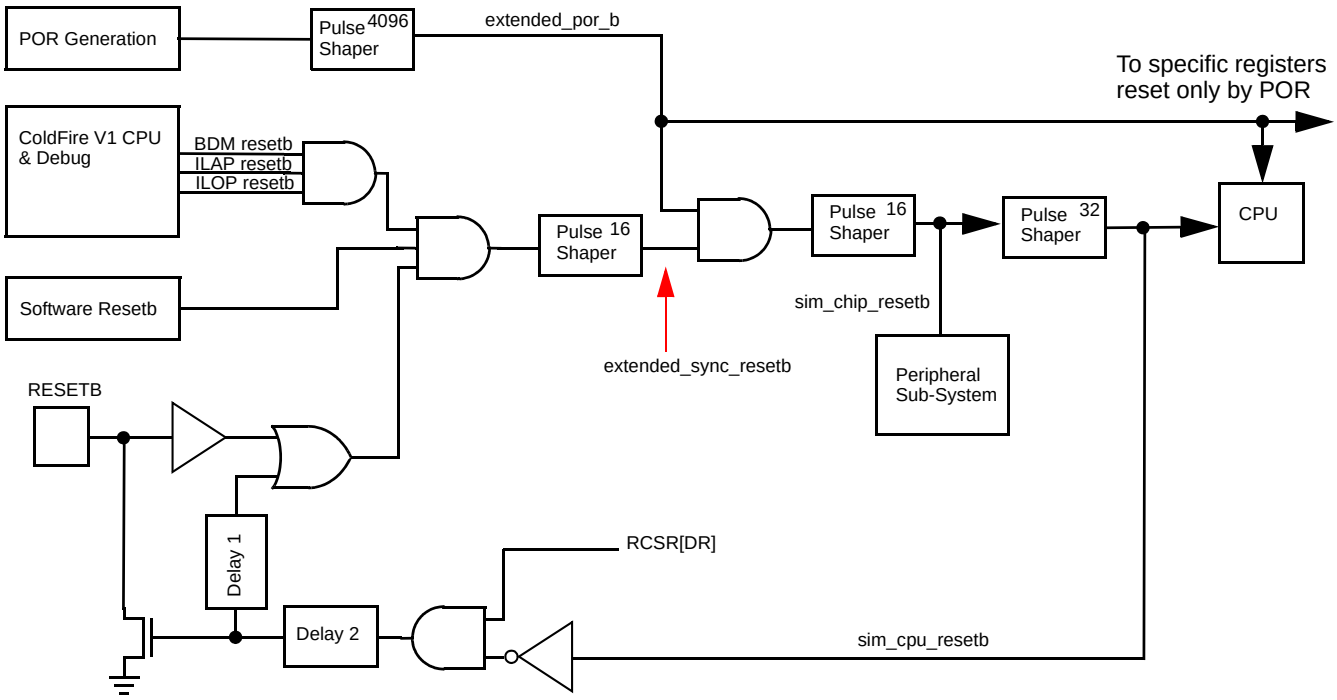


Figure 11-2. Reset generation (functional block diagram)

Figure 11-2 on page 191 illustrates how the six reset sources outlined in the previous section are processed to generate three 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 de-assertion. (*N* is specified as a number in the upper right corner of each block.)

These functions ensure that internal resets assert asynchronously and de-assert synchronously. They also guarantee that the internal resets are stable 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. The external RESETB input is only sampled when the internal CPU reset has been deasserted for two cycles or more.

Internal reset domains are the following:

extended_por_b	This signal is a minimum of 4096 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 de-assertion 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 11-3 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 4,096 cycles after that.

NOTE

In order for the selected boot from flash to happen, the RESETB pin has to be cleared before the end of the 4,096-cycle, extended period (about 500 µs). This bounds the time constant of any external filtering circuit that may be added for EMC and noise immunity.

.

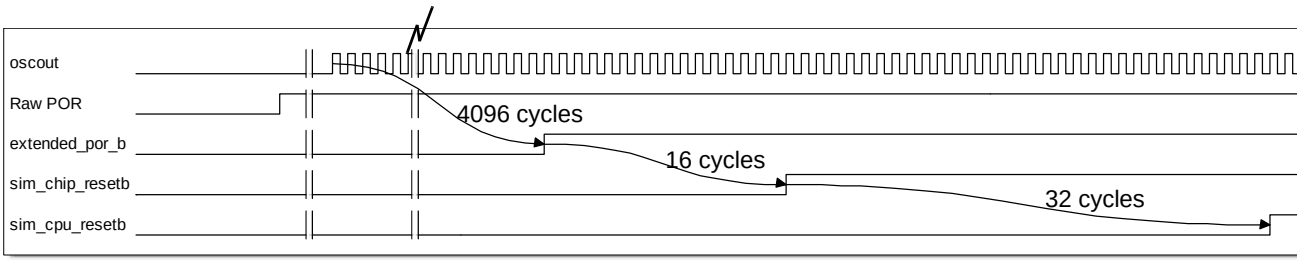


Figure 11-3. Power-On-Reset sequence

Figure 11-4 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 de-assertion sequence shown in the previous figure.

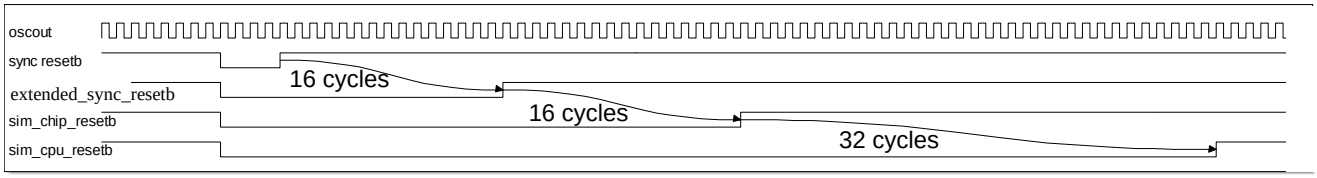


Figure 11-4. Synchronous Reset Sequence

As depicted in Figure 11-2 on page 191, 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 RCS[DR] field to 1 in the SIM Reset Control & Status Register. This will result in RESETB being pulled low for a minimum of 64 clock cycles in response to any internally generated reset event. Using this feature, the MMA955xL platform can reset external devices whenever it is reset for any purpose other than power-on-reset.

11.3 Mode control

11.3.1 STOP mode

"Operational Phases and Modes of Operation" on page 35 discusses how the various phases of operation are mapped into the Version 1 ColdFire CPU's operating modes. The STOP Control and Status Register (STOPSCR) 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. These are:

$STOP_{FC}$	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. 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 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 allow 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).

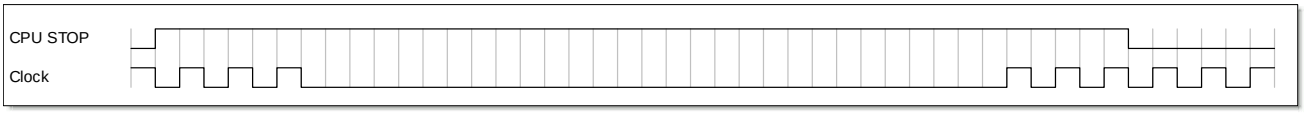


Figure 11-5. High-speed clock operation extends into STOP region

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 11-5.

Please note that interrupt operation may be limited on a module by module basis as a function of clock availability/asynchronous features for each module.

11.3.2 DEBUG modes

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

1. BKGD = 0 during POR
2. BKGD = 0 during BDM reset
3. CSR2[BFHBR] = 1 during BDM reset
4. Illegal op code reset and CSR2[IOPHR] = 1
5. Illegal address reset and CSR2[IADHR] = 1
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 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.

11.4 Oscillator control

11.4.1 General

Because the frequency of operation is coupled with the mode of operation, oscillator control is shared between the CLKGEN module (“On-Chip Oscillator” on page 215) 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 is affected by the settings specified in the STOP Control and Status Register.

11.4.2 CPU

The ColdFire core receives two clocks from the system integration module.

- 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 MMA955xL platform are active during reset. Both are derived from the CLKGEN module oscout signal.

11.5 Clock gating

The MMA955xL device includes a powerful, Version 1 ColdFire CPU and peripheral subsystem. Many applications may not need all the capabilities the device offers. Unneeded functions may have their clocks disabled in order 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 11-1 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 11-1. 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[1]

[1] Only applicable in $STOP_{FC}$ and $STOP_{SC}$ modes.

11.6 SIM memory map and registers

11.6.1 SIM memory map

Table 11-2. Module Memory Map

Offset Address	Register Name	Access	Reset	Section/Page
0x0	STOP Control and Status Register (STOPCR)	RW	0x00	page 198
0x1	Frame Control and Status Register (FCSR)	RW	0x20	page 200
0x2	Reset Status and Control Register (RSCR)	RW	0x01	page 201
0x4	Peripheral Clock Enable Register 0 for $STOP_{FC}$ mode (PCESFC0)	RW	0xFF	page 203
0x5	Peripheral Clock Enable Register 1 for $STOP_{FC}$ mode (PCESFC1)	RW	0xFF	page 205
0x6	Peripheral Clock Enable Register 0 for $STOP_{SC}$ mode (PCESSC0)	RW	0xFF	page 206
0x7	Peripheral Clock Enable Register 1 for $STOP_{SC}$ mode (PCESSC1)	RW	0xFF	page 208
0x8	Peripheral Clock Enable Register 0 for RUN mode (PCERUN0)	RW	0xFF	page 209
0x9	Peripheral Clock Enable Register 1 for RUN mode (PCERUN1)	RW	0xFF	page 211
0xA	Pin Mux Control Register0 (PMCR0)	RW	0x00	page 212
0xB	Pin Mux Control Register1 (PMCR1)	RW	0x00	page 213
0xC	Pin Mux Control Register2 (PMCR2)	RW	0x00	page 214

11.6.2 SIM registers descriptions

11.6.2.1 STOP control register

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

= Unimplemented or reserved

Figure 11-6. STOP mode control and status register (STOPCR)

Table 11-3. STOPCR register (STOPCR) field descriptions

Bit(s)	Field	Description
7-5	—	Reserved
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. (For details, “CPU Configuration Register” on page 322.) 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

Table 11-3. STOPCR register (STOPCR) field descriptions (continued)

Bit(s)	Field	Description
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. (For details, “CPU Configuration Register” on page 322.) 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 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. (For details, “CPU Configuration Register” on page 322.) 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.

11.6.2.2 Frame Control and Status Register

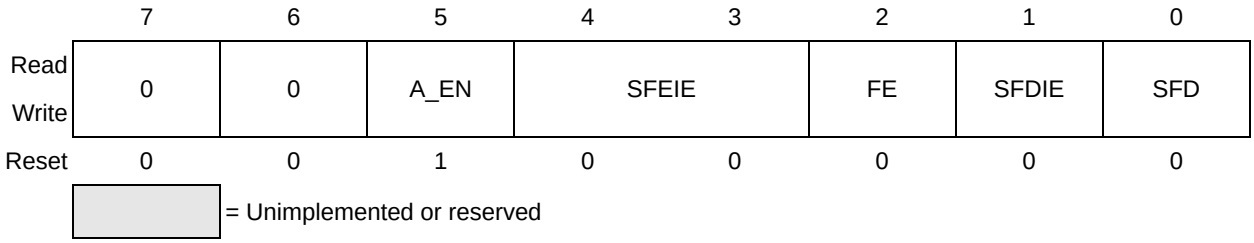


Figure 11-7. Frame Control and Status Register (FCSR)

Table 11-4. Frame Control and Status Register (FCSR) field descriptions

Bit(s)	Field	Description
7-6	—	Reserved
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.
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] 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 STOPCR[SFEE]. When set, a “start frame” signal occurring during RUN mode will set STOPCR[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 (Implement as “no error checking is performed.”) 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. STOPCR[SFEE] 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.

11.6.2.3 Reset Control and Status Register

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[4:0] 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

Power-On Reset is always set during a Power-On Reset. Power-On Reset will be cleared and external reset (PIN) will be set, however, if the external reset pin is asserted or remains asserted after the Power-On Reset has de-asserted.

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

= Unimplemented or reserved

Figure 11-8. Reset Control and Status Register (RCSR)

Table 11-5. RSCR Register Field Descriptions

Bit(s)	Field	Description
7	—	Reserved
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.
4	SW	Software Reset 0 Reset not caused by a software reset. 1 Reset initiated by writing ASR to 1

Table 11-5. RSCR Register Field Descriptions (continued)

Bit(s)	Field	Description
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. 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 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 0 Reset was not a result of a power on sequence 1 The last reset was the result of a power on sequence

11.6.2.4 Peripheral Clock Enable Register 0 for $STOP_{FC}$ Mode

The MMA955xL 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.

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.

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

= Unimplemented or reserved

Figure 11-9. Peripheral Clock Enable Register 0 for STOP FC Mode (PCESFCO)

Table 11-6. Peripheral Clock Enable Register 0 for STOP FC Mode (PCESFCO) field descriptions

Bit(s)	Field	Description
7	—	Reserved
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 General-purpose timer T1 clock is not enabled. 1 The clock to the timer/PWM 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.

Table 11-6. Peripheral Clock Enable Register 0 for STOP FC Mode (PCESFCO) field descriptions (continued)

Bit(s)	Field	Description
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.

11.6.2.5 Peripheral Clock Enable Register 1 for $STOP_{FC}$ Mode

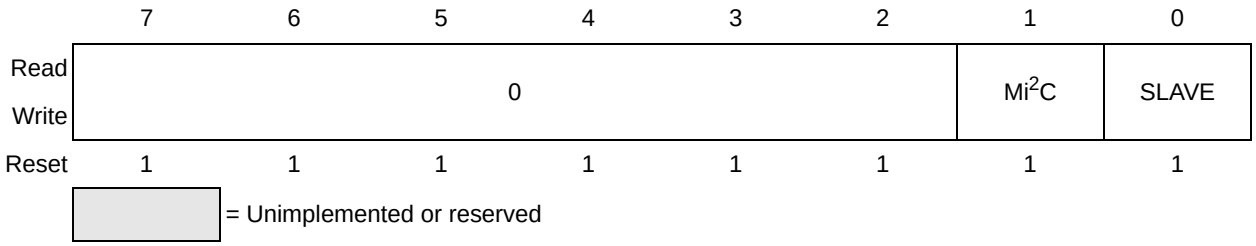


Figure 11-10. Peripheral Clock Enable Register 1 for STOP FC Mode (PCESFC1)

Table 11-7. Peripheral Clock Enable Register 1 for STOP FC Mode (PCESFC1) field descriptions

Bit(s)	Field	Description
7-2	—	Reserved
1	Mi^2C	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 ISP/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.

11.6.2.6 Peripheral Clock Enable Register 0 for $STOP_{SC}$ Mode

The MMA955xL 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.

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

= Unimplemented or reserved

Figure 11-11. Peripheral Clock Enable Register 0 for STOP SC Mode (PCESSCO)

Table 11-8. Peripheral Clock Enable Register 0 for STOP SC Mode (PCESSCO) field descriptions

Bit(s)	Field	Description
7	—	Reserved
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 General-purpose timer T1 clock is not enabled. 1 The clock to the timer/PWM 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.

Table 11-8. Peripheral Clock Enable Register 0 for STOP SC Mode (PCESSCO) field descriptions (continued)

Bit(s)	Field	Description
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.

11.6.2.7 Peripheral Clock Enable Register 1 for $STOP_{SC}$ Mode

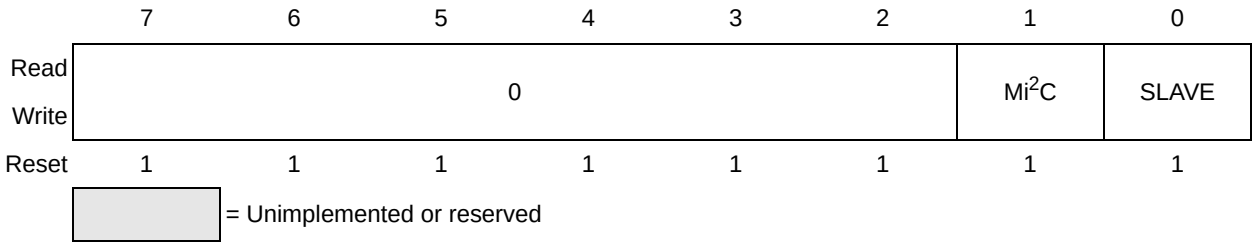


Figure 11-12. Peripheral Clock Enable Register 1 for STOP FC Mode (PCESSC1)

Table 11-9. Peripheral Clock Enable Register 1 for STOP FC Mode (PCESSC1) field descriptions

Bit(s)	Field	Description
7-2	—	Reserved
1	Mi^2C	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 ISP/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.

11.6.2.8 Peripheral Clock Enable Register 0 for RUN Mode

The MMA955xL 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.

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

= Unimplemented or reserved

Figure 11-13. Peripheral Clock Enable Register 0 for RUN Mode (PCERUNO)

Table 11-10. Peripheral Clock Enable Register 0 for RUN Mode (PCERUNO) field descriptions

Bit(s)	Field	Description
7	—	Reserved
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 General-purpose timer T1 clock is not enabled. 1 The clock to the timer/PWM 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.

Table 11-10. Peripheral Clock Enable Register 0 for RUN Mode (PCERUNO) field descriptions (continued)

Bit(s)	Field	Description
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.

11.6.2.9 Peripheral Clock Enable Register 1 for RUN Mode

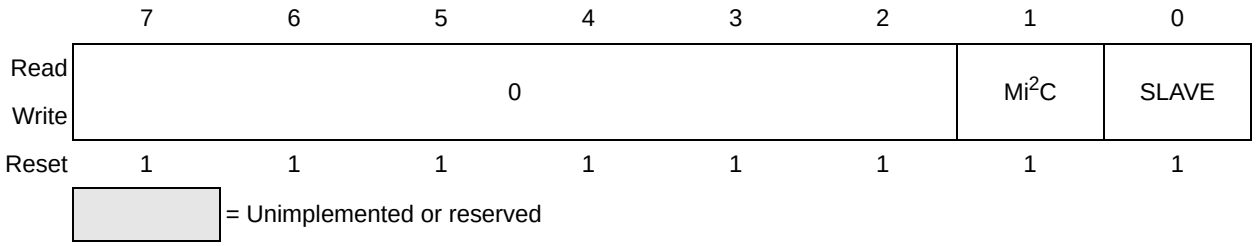


Figure 11-14. Peripheral Clock Enable register (PCERUN1)

Table 11-11. Peripheral Clock Enable register (PCERUN1) field descriptions

Bit(s)	Field	Description
7-2	—	Reserved
1	Mi^{2}C	Master I^{2}C Clock Enable 0 The clock to the master I^{2}C interface is not enabled. The module cannot be used in this mode of operation. 1 The clock to the master I^{2}C 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 ISP/I^{2}C 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.

11.6.2.10 Pin Mux Control Register0

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 on page 26.

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

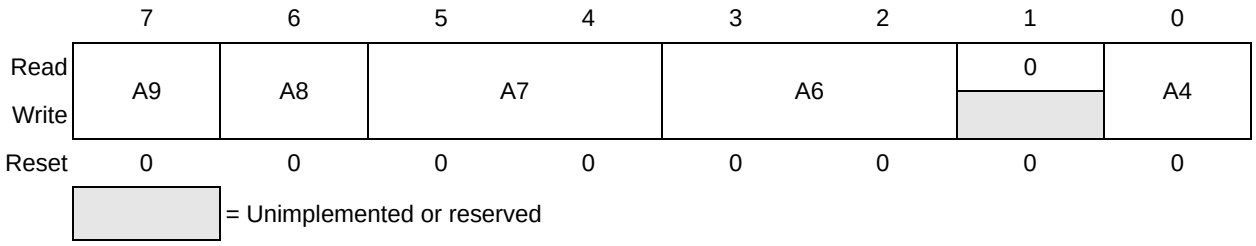


Figure 11-15. Pin Mux Control Register 0 (PMCR0)

Table 11-12. Pin Mux Control Register 0 (PMCR0) field descriptions

Bit(s)	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 00 Pin function is RGPIO Bit 7 01 Pin function is AN1 10 Pin Function is TPMCH. 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
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

11.6.2.11 Pin Mux Control Register1

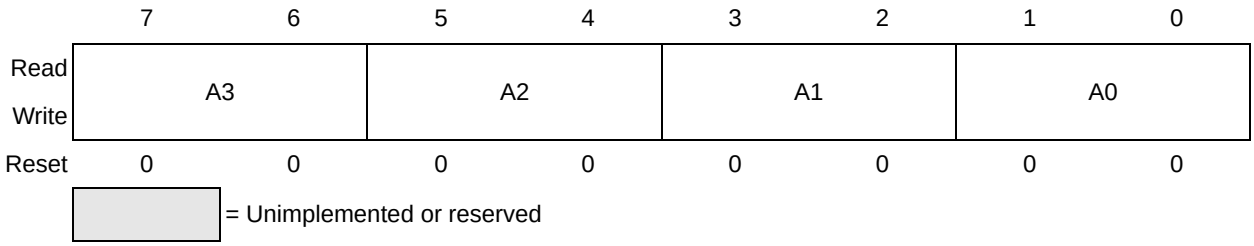


Figure 11-16. Pin Mux Control Register 1 (PMCR1)

Table 11-13. Pin Mux Control Register 1 (PMCR1) field descriptions

Bit(s)	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 SDO (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 SDI (SPI data in) 11 RESERVED
1-0	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

11.6.2.12 Pin Mux Control Register2

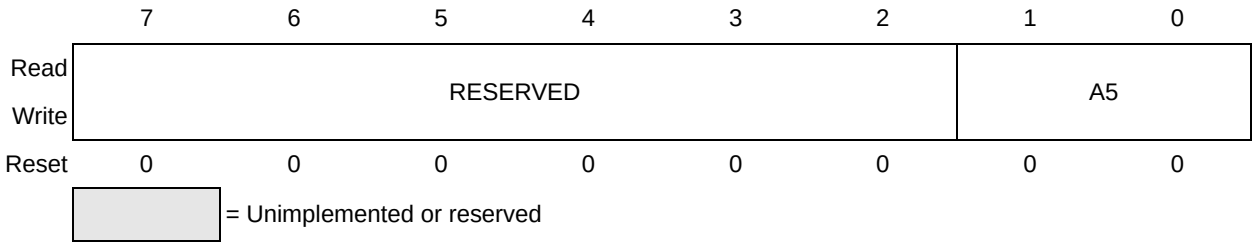


Figure 11-17. Pin Mux Control Register 2 (PMCR2)

Table 11-14. Pin Mux Control Register 2 (PMCR2) field descriptions

Bit(s)	Field	Description
7-2	—	Reserved
1-0	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 12 On-Chip Oscillator

12.1 Introduction

This device includes a single, on-chip oscillator (CLKGEN) that has several modes of operation, summarized in Table 12-1.

Table 12-1. Oscillator modes

Mode	HLb	pdwnb
High-speed mode ($F_{osc-high}$ = 8 MHz)	1	1
Low-speed mode ($F_{osc-low}$ = $F_{osc-high}$/128)	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[1], 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.

12.2 High-level overview

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 12-1 shows 128 cycles in high-speed mode, and one in low-speed mode. The second signal in the figure is running at a rate of high-speed clock/64 and is for reference purposes only.

1. Frame: Length = 1/FR. See “Frame structure” on page 35 for additional details.

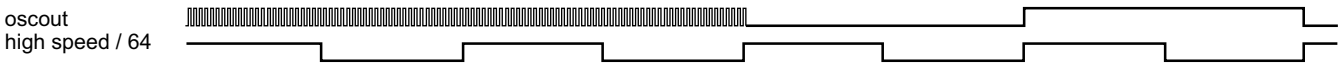


Figure 12-1. Oscillator output frequencies

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. At 128/8 MHz, that works out to be 16 µs.

Figure 12-2 illustrates the basic mechanisms required. These are:

- Two-speed oscillator. This is contained within the “oscillator wrapper” function in the figure. The faster speed is nominally 8 MHz. The lower speed is this divided by 128 (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.
- Seven-bit synchronous counter with carry. This counter only increments when the oscillator is in high-speed mode. When the counter rolls over from value $7F 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 allows 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 Chapter 11, “System Integration Module”.
- 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.

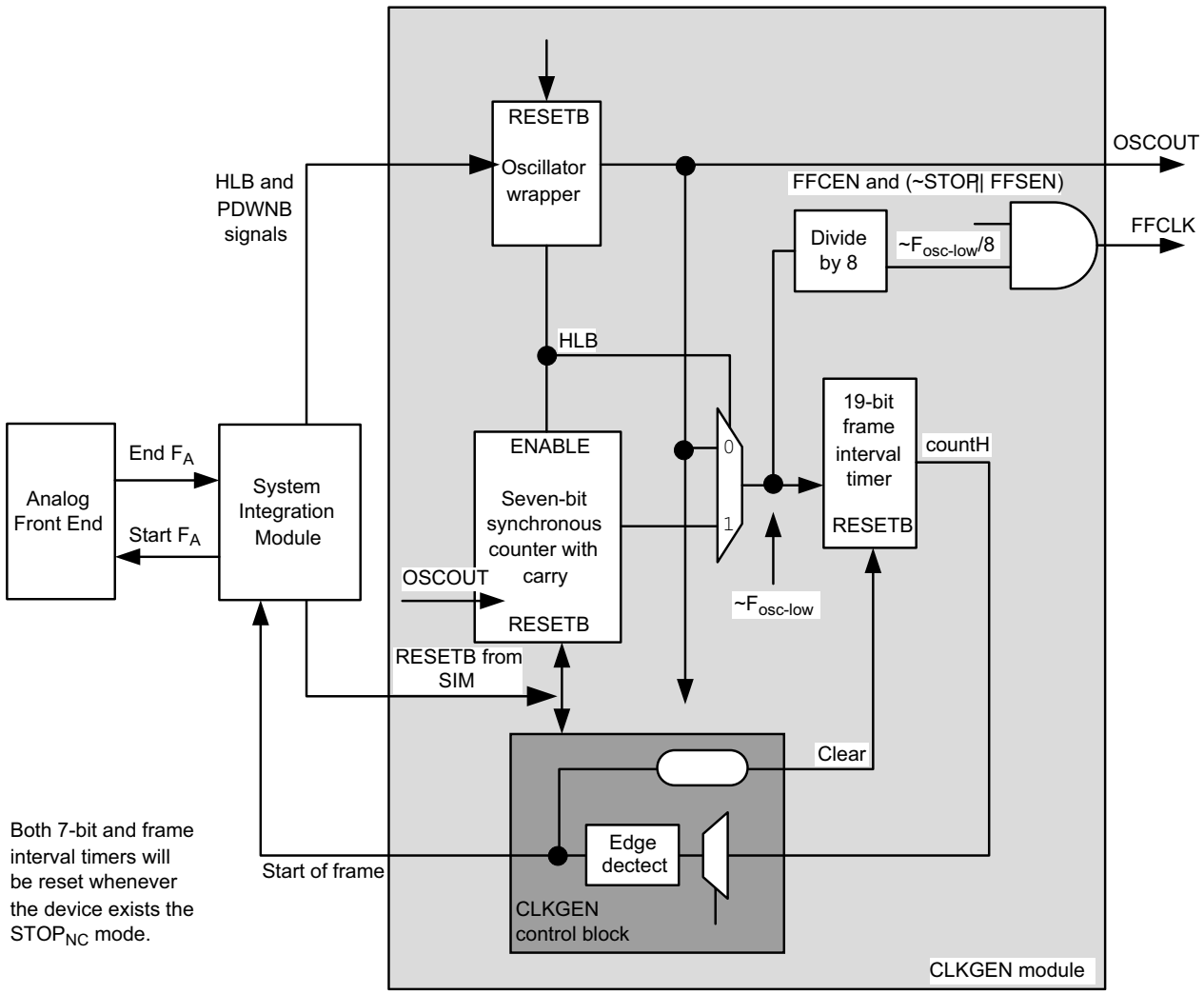


Figure 12-2. Oscillator functional block diagram

Figure 12-2 gives some insight about CLKGEN Module internal circuits as well as its interaction with the System Integration Module. This block diagram is mainly a simplified view of the oscillator that illustrates its control and operation. The image is *not* meant to fully reflect the actual hardware implementation.

12.3 CLKGEN Memory Map/Register Definition

The CLKGEN module is organized as a memory-mapped peripheral on the eight-bit IP Bus. The following table specifies the module memory map. Details of each register are provided in the following section.

Table 12-2. CLKGEN memory map

Offset address	Register	Access	Reset	Details
0x00	Oscillator Control Register (OSCTRL)	RW	0x80	Figure 12-3 on page 219

12.3.1 Oscillator Control Register

Table 12-3. Frame interval and clocks as a function of FLE (assuming 200 µs ΦA duration)

FLE	2^{FLE}	t_F (sec)	Max frames per second	Max fast clock cycles per frame	Φ_D duration (us)	Max fast clock cycles per Φ_D	% CPU available to Φ_D
4	16	1.28E-4	7812.5	2,048	Not recommended for analog frames.		
5	32	2.56E-04	3906.25	4,096	56.00	896	21.875%
6	64	5.12E-04	1953.13	8,192	312.00	4,992	60.938%
7	128	1.02E-03	976.56	16,384	824.00	13,184	80.469%
8	256	2.05E-03	488.28	32,768	1,848.00	29,568	90.234%
9	512	4.10E-03	244.14	65,536	3,896.00	62,336	95.117%
10	1024	8.19E-03	122.07	131,072	7,992.00	127,872	97.559%
11	2048	1.64E-02	61.04	262,144	16,184.00	258,944	98.779%
12	4096	3.28E-02	30.52	524,288	32,568.00	521,088	99.390%
13	8192	6.55E-02	15.26	1,048,576	65,336.00	1,045,376	99.695%
14	16384	1.31E-01	7.63	2,097,152	130,872.00	2,093,952	99.847%
15	32768	2.62E-01	3.81	4,194,304	261,944.00	4,191,104	99.924%
16	65536	5.24E-01	1.91	8,388,608	524,088.00	8,385,408	99.962%
17	131072	1.05E+00	0.95	16,777,216	1,048,376.00	16,774,016	99.981%
18	262144	2.10E+00	0.48	33,554,432	2,096,952.00	33,551,232	99.990%
19	524288	4.19E+00	0.24	67,108,864	4,194,104.00	67,105,664	99.995%

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

= Unimplemented or reserved

Figure 12-3. Oscillator Control register (OSCTRL)

Table 12-4. Oscillator Control register (OSCTRL) field descriptions

Bit(s)	Field	Description
7	FCEN	Frame Counter Enable This bit disabled 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 $F_{osc-low}$. This clock is inactive during reset. This clock is subject to a significant amount of jitter (approximately $\pm F_{osc-low}^{-1}$) 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:** Use of 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. 0 FFCLK is disabled in all STOP modes 1 FFCLK is enabled during STOPFC and STOPSC. It is disabled during STOPNC.
4-0	FLE	Frame Length Exponent The interval between frames is expressed as TF = 2^{FLE} Posc-low. FLE is any value between $04 and $12. Figure 12-3 details possible values of FLE and corresponding values of t_F. 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. Switching FLE from an inactive to an active value has the effect of starting the frame.

12.4 Interrupts

The CLKGEN module generates the start Φ_A signal which acts as a wake-up to the analog front end. Φ_A is *not* an interrupt. But the CLKGEN module can be programmed to generate a Φ_D interrupt instead of the start Φ_A signal. This will bypass the analog phase and go straight to the digital phase.

Under certain circumstances, the Φ_A signal could result in the System Integration Module (SIM) generating an interrupt to signal a start-frame error. For details, see “Peripheral Clock Enable Register 0 for $STOP_{FC}$ Mode” on page 203.

Chapter 13 Programmable Delay Block

13.1 Introduction

13.1.1 Features

- 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 has an independently controlled delay from the trigger_input.
- Digital comparator outputs can be used to schedule precise edge placement for a pulsed output.
- Continuous-trigger or single-shot mode supported.
- Each output is independently enabled.

13.1.2 Modes of operation

Modes of operation include:

- Disabled: Counter is off and both A and B outputs are low.
- Enabled One Shot: Counter is enabled and restarted at count zero upon receiving a positive edge on the input trigger. A and B will see only one output transition per input trigger.
- Enabled Continuous: Counter is enabled and restarted at count zero. The counter will be rolled over to zero again when the count reaches the value specified in the MOD register and the counting will be restarted. This enables a continuous stream of output pulses as a result of a single-trigger input.

13.1.3 Block diagram

Figure 13-1 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 point that changes in the output signals are initiated. These times are defined as:

- Trigger input to A = prescaler x (delayA + 1 peripheral bus clock cycle)
- Trigger input to B = prescaler x (delayB + 1 peripheral bus clock cycle)
- Add one additional peripheral bus-clock cycle when using both A and B comparators to schedule both edges on an output pulse.

The third digital value, modulus, is used to reset the counter back to zero at the end of the count. If CSR[CONT] is set, the counter will then resume a new count. Otherwise, the timer operation will cease until the next trigger input event occurs.

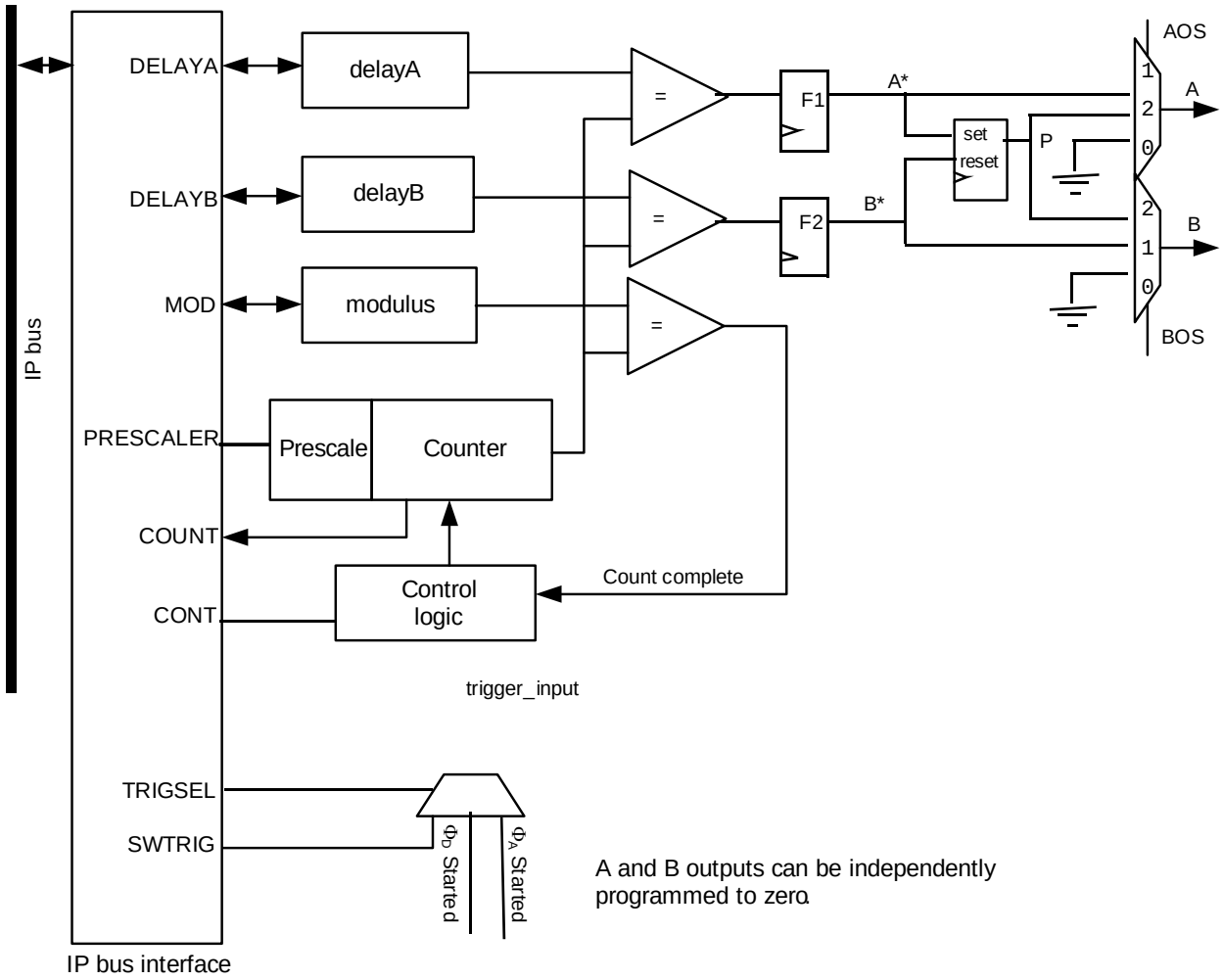


Figure 13-1. PDB block diagram

The pulsed mode is shown in Figure 13-2. In this case, A* and B* are used to precisely schedule the rising and falling edges for the output waveform.

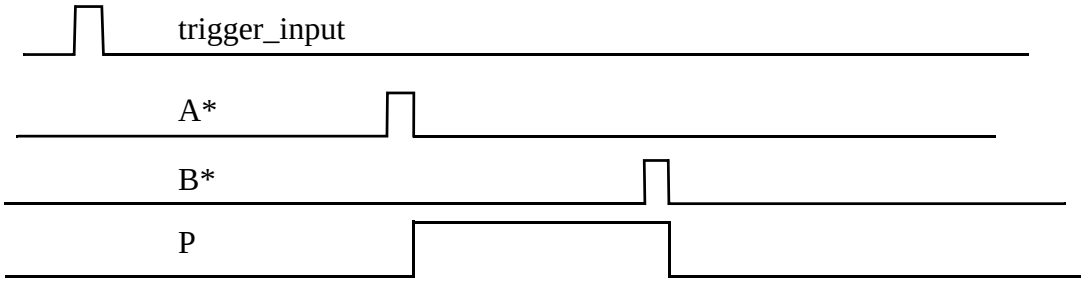


Figure 13-2. Trigger-pulsed output operation

13.2 Programmable Delay Block memory map and registers

13.2.1 Programmable Delay Block memory map

Table 13-1. PDB Memory Map

Offset address	Register	Access	Reset value	Details
0x0000	PDB Control and Status Register (CSR)	Read/Write	0x0000	page 224
0x0002	PDB Delay A Register (DELAYA)	Read/Write	0x0000	page 226
0x0004	PDB Delay B Register (DELAYB)	Read/Write	0x0000	page 226
0x0006	PDB Counter Modulus Register (MOD)	Read/Write	0xFFFF	page 227
0x0008	PDB Counter Value (COUNT)	Read	0xFFFF	page 227

13.2.2 Programmable Delay Block registers descriptions

13.2.2.1 PDB Control and Status Register

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).

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read Write	PRESCALER			SB	SA	IENB	IENA	BOS		AOS		CONT	SW TRIG	TRIGSEL		EN
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Reserved or unused

Figure 13-3. Programmable Delay Block Control and Status Register (CSR)

Table 13-2. Programmable Delay Block Control and Status Register (CSR) field descriptions

Bit(s)	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. 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 0 Interrupt B is not enabled. 1 Assert an interrupt when B* triggers and SB goes high. The interrupt is cleared by writing a 1 to SB.
9	IENA	Interrupt Enable A 0 Interrupt A is not enabled 1 Assert an interrupt when A* triggers and SA goes high. The interrupt is cleared by writing a 1 to SA.

Table 13-2. Programmable Delay Block Control and Status Register (CSR) field descriptions (continued)

Bit(s)	Field	Description
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 FCSR. It can be used even when that interrupt is not enabled. The timing is the same either way. 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.

13.2.2.2 PDB Delay A Register

These registers are used to specify the delay from assertion of TriggerIn to assertion of A and B out. The delay is in terms of peripheral clock cycles. These registers should only be changed when the module is not enabled.

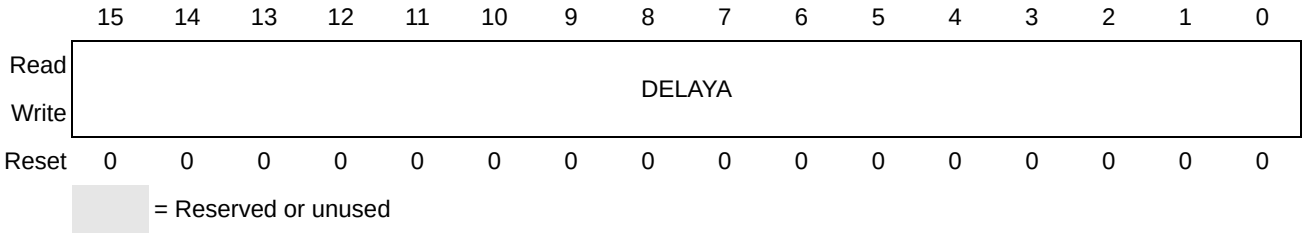


Figure 13-4. PDB Delay A register (DELAYA)

Table 13-3. PDB Delay A register (DELAYA) field descriptions

Bit(s)	Field	Description
15-0	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.

13.2.2.3 PDB Delay B Register (DELAYB)

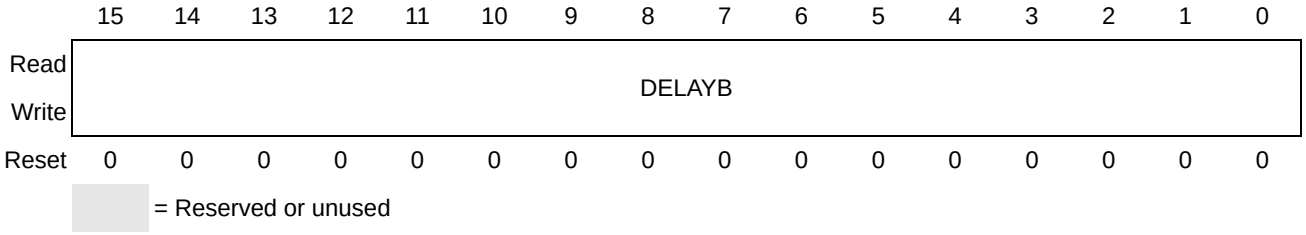


Figure 13-5. PDB Delay B register (DELAYB)

Table 13-4. PDB Delay B register (DELAYB) field descriptions

Bit(s)	Field	Description
15-0	DELAYB	Delay is in terms of prescaled peripheral clock cycles from assertion of Trigger Input to assertion of B output. A delay value of $0000 will never be reached and no output trigger will occur since the counter goes from $0001 to $FFFF.

13.2.2.4 PDB Modulus Register (MOD)

This register specifies the period of the counter in terms of peripheral-bus cycles. When the counter reaches this value, it will be reset back to all 0s. If CSR[CONT] is set to one, the count will begin anew.

This register should be changed only when the module is not enabled.

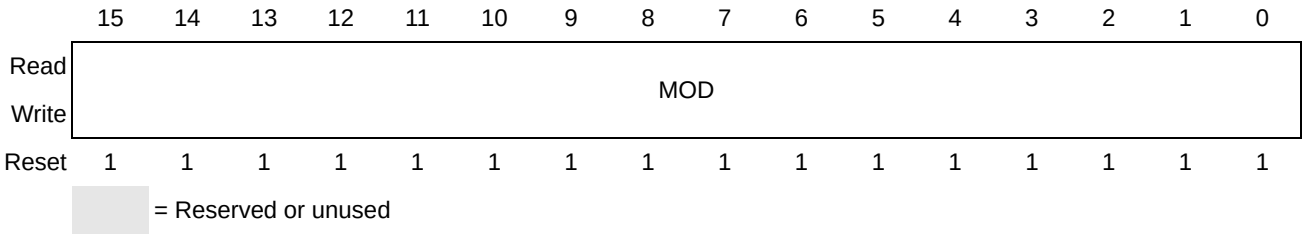


Figure 13-6. PDB Modulus register (MOD)

Table 13-5. PDB Modulus register (MOD) field descriptions

Bit(s)	Field	Description
15-0	MOD	Specifies the period of the counter in terms of prescaled peripheral bus cycles.

13.2.2.5 PDB COUNT Register (COUNT)

This register can be used to read the current value of the counter. It is READ ONLY.

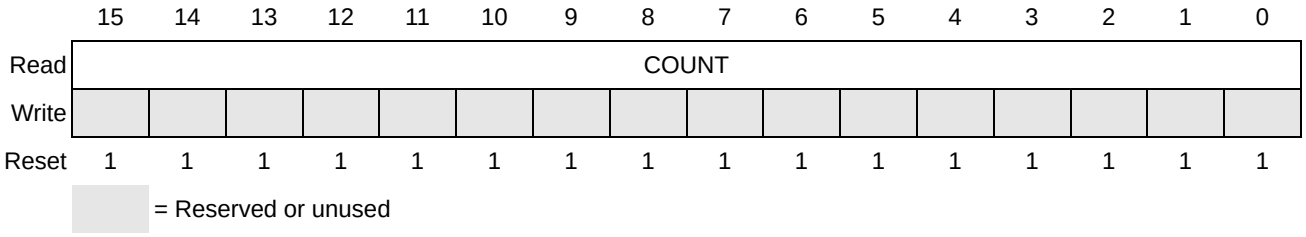


Figure 13-7. PDB COUNT register (COUNT)

Table 13-6. PDB COUNT register (COUNT) field descriptions

Bit(s)	Field	Description
15-0	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.

13.2.3 Functional description

13.2.3.1 Miscellaneous concerns and SoC integration

- A and B 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 wanted to set a really long delay value and used div 128, that person would be limited to an resolution of 128 bus clocks.

Therefore, use the lowest possible prescaler for a given application.

13.3 Resets

This module has a single reset input, corresponding to the chip-wide peripheral reset.

13.4 Clocks

This module has a single clock input, the IP Bus peripheral clock.

13.5 Interrupts

This module has two possible interrupts: One associated with the A output and one with the B output.

Chapter 14 Port Controls

14.1 Port Control customizations

There are two instances of the port-control module on the MMA955xL platform. Each port control module can control up to eight pins.

PC0 controls drive strength, slew rate and pull-up controls for RGPIO[9:8]. PC1 controls these parameters for RGPIO[7:0]. Specifically, the mapping is:

Table 14-1. RPGIO port controls

RPGIO bit number	Port control
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]

14.1.1 General rules

- PCxSE = 0 (output slew rate control disabled)
- PCxDS = 0 (low output-drive strength)
- PCxIFE = 1 (input filters enabled)
- PCxPE = 0 (pull-ups 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 “Pin Mux Control Register0” on page 212.)

14.1.2 Exceptions to the general rules

The BKGD/MS pin (RGPIO9) defaults to high drive strength when no slew rate control is enabled by pull-up resistors.

When A4 is programmed as interrupt, the pull-up/down enable is controlled by the IRQSC[IRQPDD] signal, instead of PC1PE[PE4]. At the same time, if the IRQSC[IRQPDD] signal is enabled and IRQSC[IRQEDG] is 0, a pull-up resistor is used. If IRQSC[IRQEDG] is 1, a pull-down resistor is used.

14.1.3 Pins not covered by the port control modules

The RESETB pin is not multiplexed with GPIO, so the RESET B pad cell is not configurable. The RESETB pin's fixed configuration is as follows:

- Low output drive strength
- Input filter enabled
- Pull-up resistor enabled
- The output buffer of the RESETB pin is open drain.(For details, see "RESETB" on page 28.)

14.2 Standard pin controls

14.2.1 Pin controls overview

A set of registers (shown in Figure 14-1) control pull-ups, slew-rate, drive-strength and input-filter enables for the pins. That set of registers also may be used in conjunction with the peripheral functions on these pins.

These registers are associated with the parallel I/O ports and Rapid GPIO (RGPIO) ports, but operate independently of both.

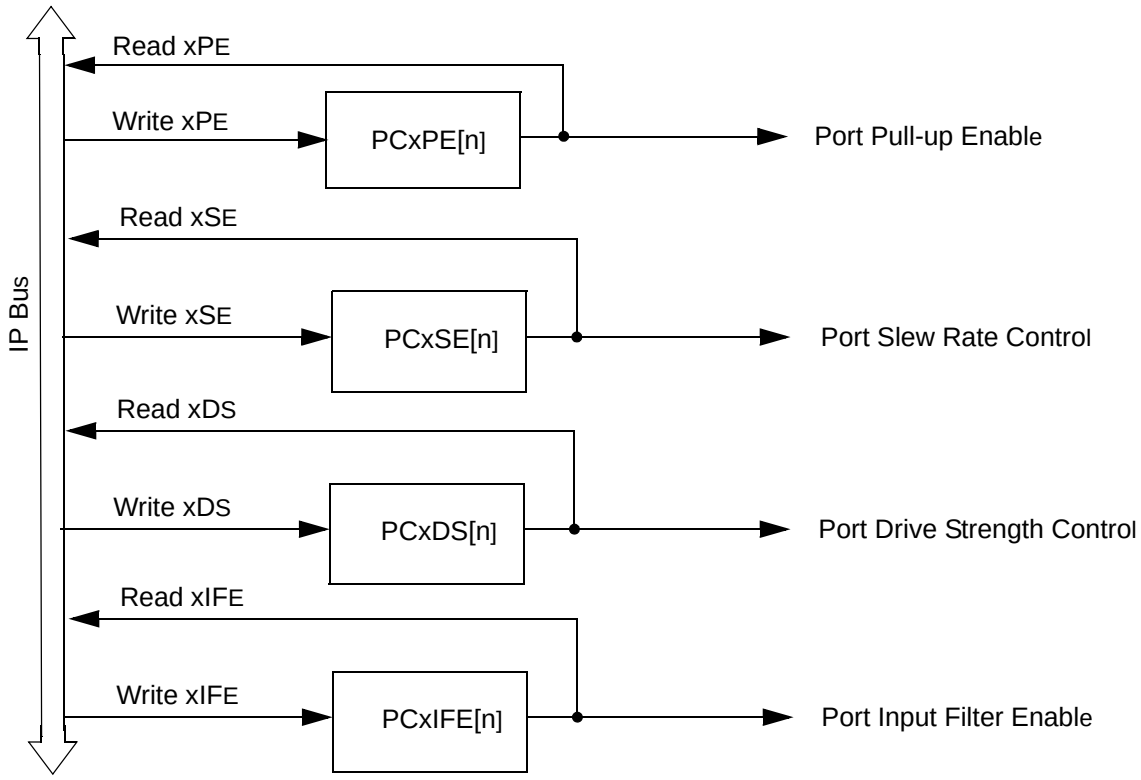


Figure 14-1. Pin control logic block diagram

14.3 Port Controls memory map and registers

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.

14.3.1 Port Controls memory map

Table 14-2. Port Controls memory map

Offset address	Register	Access	Reset	Details
0x0000	Port x Pull-Up Enable Register (PCxPE)	Read/Write	0x00	page 233
0x0001	Port x Slew Rate Enable Register (PCxSE)	Read/Write	0x00	page 235
0x0002	Port x Drive Strength Selection Register (PCxDS)	Read/Write	0x00	page 236
0x0003	Port x Input Filter Enable Register (PCxIFE)	Read/Write	0xFF	page 237

For the absolute address assignments for all registers, see the tables in Chapter 5, “Memory Maps”. 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.

14.3.2 Port Controls registers

14.3.2.1 Port x Pull-Up Enable Register

An internal pull-up device can be enabled for each port pin by setting the corresponding bit in the pull-up-enable register (PCxPE[*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 PCx pin. For Port x pins that are configured as outputs, these bits have no effect and the internal pull-up devices are disabled.

	7	6	5	4	3	2	1	0
Read Write	PCxPE7	PCxPE6	PCxPE5	PCxPE4	PCxPE3	PCxPE2	PCxPE1	PCxPE0
Reset:	0	0	0	0	0	0	0	0

= Unimplemented or reserved

Figure 14-2. Internal Pull-Up Enable for Port x register (PCxPE)

Table 14-3. Internal Pull-Up Enable for Port x register (PCxPE) field descriptions

Bit(s)	Field	Description
7	PCxPE7	Detailed description above. 0 Internal pull-up device disabled for Port x bit *n*. 1 Internal pull-up device enabled for Port x bit *n*.
6	PCxPE6	Detailed description above. 0 Internal pull-up device disabled for Port x bit *n*. 1 Internal pull-up device enabled for Port x bit *n*.
5	PCxPE5	Detailed description above. 0 Internal pull-up device disabled for Port x bit *n*. 1 Internal pull-up device enabled for Port x bit *n*.
4	PCxPE4	Detailed description above. 0 Internal pull-up device disabled for Port x bit *n*. 1 Internal pull-up device enabled for Port x bit *n*.
3	PCxPE3	Detailed description above. 0 Internal pull-up device disabled for Port x bit *n*. 1 Internal pull-up device enabled for Port x bit *n*.

Table 14-3. Internal Pull-Up Enable for Port x register (PCxPE) field descriptions (continued)

Bit(s)	Field	Description
2	PCxPE2	Detailed description above. 0 Internal pull-up device disabled for Port x bit *n*. 1 Internal pull-up device enabled for Port x bit *n*.
1	PCxPE1	Detailed description above. 0 Internal pull-up device disabled for Port x bit *n*. 1 Internal pull-up device enabled for Port x bit *n*.
0	PCxPE0	Detailed description above. 0 Internal pull-up device disabled for Port x bit *n*. 1 Internal pull-up device enabled for Port x bit *n*.

14.3.2.2 Port x Slew Rate Enable Register

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

	7	6	5	4	3	2	1	0
Read Write	PCxSE7	PCxSE6	PCxSE5	PCxSE4	PCxSE3	PCxSE2	PCxSE1	PCxSE0
Reset:	0	0	0	0	0	0	0	0

= Unimplemented or reserved

Figure 14-3. Slew-Rate Enable for Port x register (PCxSE)

Each of these control bits determines if the output slew rate control is enabled for the associated PCx pin. For Port x pins configured as inputs, these bits have no effect.

Table 14-4. Slew-Rate Enable for Port x register (PCxSE) field descriptions

Bit(s)	Field	Description
7	PCxSE7	Detailed description above. 0 Output slew rate control disabled for Port x bit *n*. 1 Output slew rate control enabled for Port x bit *n*.
6	PCxSE6	Detailed description above. 0 Output slew rate control disabled for Port x bit *n*. 1 Output slew rate control enabled for Port x bit *n*.
5	PCxSE5	Detailed description above. 0 Output slew rate control disabled for Port x bit *n*. 1 Output slew rate control enabled for Port x bit *n*.
4	PCxSE4	Detailed description above. 0 Output slew rate control disabled for Port x bit *n*. 1 Output slew rate control enabled for Port x bit *n*.
3	PCxSE3	Detailed description above. 0 Output slew rate control disabled for Port x bit *n*. 1 Output slew rate control enabled for Port x bit *n*.
2	PCxSE2	Detailed description above. 0 Output slew rate control disabled for Port x bit *n*. 1 Output slew rate control enabled for Port x bit *n*.
1	PCxSE1	Detailed description above. 0 Output slew rate control disabled for Port x bit *n*. 1 Output slew rate control enabled for Port x bit *n*.
0	PCxSE0	Detailed description above. 0 Output slew rate control disabled for Port x bit *n*. 1 Output slew rate control enabled for Port x bit *n*.

14.3.2.3 Port x Drive Strength Selection Register

An output pin can be selected to have high output drive strength by setting the corresponding bit in the drive-strength select register (PCxDS[*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 allows a pin to drive a greater load with the same switching speed as a low-drive-enabled pin into a smaller load. Because of this, the EMC emissions may be affected by enabling pins as high drive.

Each of these control bits selects between low- and high-output drive for the associated PCx pin. For Port x pins configured as inputs, these bits have no effect.

	7	6	5	4	3	2	1	0
Read Write	PCxDS7	PCxDS6	PCxDS5	PCxDS4	PCxDS3	PCxDS2	PCxDS1	PCxDS0
Reset:	0	0	0	0	0	0	0	0

= Unimplemented or reserved

Figure 14-4. Drive Strength Selection for Port x register (PCxDS)

(continued)

Table 14-5. Drive Strength Selection for Port x register (PCxDS) field descriptions

Bit(s)	Field	Description
7	PCxDS7	Detailed description above. 0 Low output drive strength selected for Port x bit *n* 1 High output drive strength selected for Port x bit *n*.
6	PCxDS6	Detailed description above. 0 Low output drive strength selected for Port x bit *n* 1 High output drive strength selected for Port x bit *n*.
5	PCxDS5	Detailed description above. 0 Low output drive strength selected for Port x bit *n* 1 High output drive strength selected for Port x bit *n*.
4	PCxDS4	Detailed description above. 0 Low output drive strength selected for Port x bit *n* 1 High output drive strength selected for Port x bit *n*.
3	PCxDS3	Detailed description above. 0 Low output drive strength selected for Port x bit *n* 1 High output drive strength selected for Port x bit *n*.
2	PCxDS2	Detailed description above. 0 Low output drive strength selected for Port x bit *n* 1 High output drive strength selected for Port x bit *n*.
1	PCxDS1	Detailed description above. 0 Low output drive strength selected for Port x bit *n* 1 High output drive strength selected for Port x bit *n*.

Table 14-5. Drive Strength Selection for Port x register (PCxDS) field descriptions

Bit(s)	Field	Description
0	PCxDS0	Detailed description above. 0 Low output drive strength selected for Port x bit *n* 1 High output drive strength selected for Port x bit *n*.

14.3.2.4 Port x Input Filter Enable Register

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 (PCxIFE[*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.

	7	6	5	4	3	2	1	0
Read Write	PCxIFE7	PCxIFE6	PCxIFE5	PCxIFE4	PCxIFE3	PCxIFE2	PCxIFE1	PCxIFE0
Reset:	1	1	1	1	1	1	1	1

= Unimplemented or reserved

Figure 14-5. Port x Input Filter Enable register (PCxIFE)

Table 14-6. Port x Input Filter Enable register (PCxIFE) field descriptions

Bit(s)	Field	Description
7	PCxIFE7	Detailed description above. 0 Input filter disabled for Port x bit *n*. 1 Input filter enabled for Port x bit *n*.
6	PCxIFE6	Detailed description above. 0 Input filter disabled for Port x bit *n*. 1 Input filter enabled for Port x bit *n*.
5	PCxIFE5	Detailed description above. 0 Input filter disabled for Port x bit *n*. 1 Input filter enabled for Port x bit *n*.
4	PCxIFE4	Detailed description above. 0 Input filter disabled for Port x bit *n*. 1 Input filter enabled for Port x bit *n*.
3	PCxIFE3	Detailed description above. 0 Input filter disabled for Port x bit *n*. 1 Input filter enabled for Port x bit *n*.

Table 14-6. Port x Input Filter Enable register (PCxIFE) field descriptions (continued)

Bit(s)	Field	Description
2	PCxIFE2	Detailed description above. 0 Input filter disabled 1 Input filter enabled
1	PCxIFE1	Detailed description above. 0 Input filter disabled 1 Input filter enabled
0	PCxIFE0	Detailed description above. 0 Input filter disabled 1 Input filter enabled

Chapter 15 Rapid GPIO

15.1 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 and 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.

15.1.1 Overview

The RGPIO module provides 16-bits of high-speed GPIO functionality, mapped to the processor's bus. The key features of this module include:

- 16 bits of high-speed GPIO functionality connected to the processor's local 32-bit bus
- Memory-mapped device connected to the ColdFire core's local bus
  - — Support for all access sizes: byte, word, and longword
  - — All reads and writes complete in a single data phase cycle for zero wait-state response
- Data bits can be accessed directly or via alternate addresses to provide set, clear, and toggle functions
  - — Alternate addresses allow set, clear, toggle functions using simple store operations without the need for read-modify-write references
- Unique data direction and pin enable control registers
- Package pin toggle rates typically 1.5–3.5x faster than comparable pin mapped onto peripheral bus

A simplified block diagram of the RGPIO module is shown in Figure 15-1 on page 240. The details of the pin muxing and pad logic are device-specific.

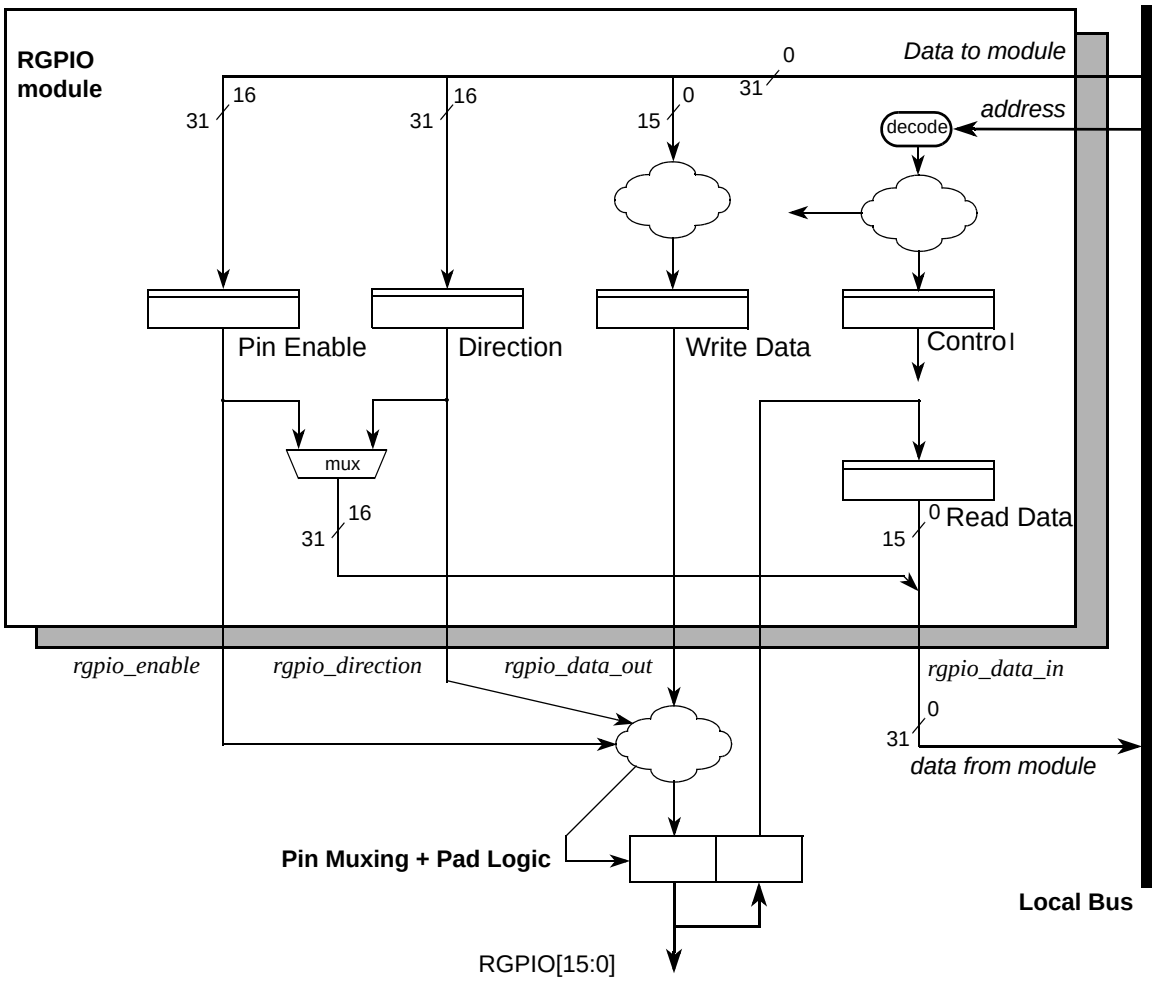


Figure 15-1. RGPIO block diagram

15.1.2 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)

15.1.3 Modes of operation

The RGPIO module does not support any special modes of operation. As a memory-mapped device located on the processor's high-speed local bus, it responds based strictly on memory address and does not consider the operating mode (supervisor, user) of its references.

15.2 External signal description

15.2.1 Overview

As shown in Figure 15-1 on page 240, the RGPIO module's interface to external logic is indirect via the device pin-muxing and pad logic. For a list of the associated RGPIO input/output signals, see Table 15-1.

Table 15-1. RGPIO Module External I/O Signals

Signal Name	Type	Description
RGPIO[15:0]	I/O	RGPIO Data Input/Output

15.2.2 Detailed Rapid GPIO signal descriptions

Table 15-2 provides descriptions of the RGPIO module's input and output signals.

Table 15-2. RGPIO signal descriptions

Signal	Type	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.

15.3 Rapid GPIO memory map/register definitions

The RGPIO module provides a compact 16-byte programming model based at a system memory address of 0x_0000 (noted as RGPIO_BASE throughout the chapter). As previously noted, the programming model views are different between reads and writes as this enables simplified software for manipulation of the RGPIO pins. Additionally, the programming model can be referenced using any operand size access (byte, word, longword). Performance is typically maximized using 32-bit accesses.

NOTE

Writes to the two-byte fields at RGPIO_BASE + 0x8 and RGPIO_BASE + 0xC are allowed, but do not affect any program-visible register within the RGPIO module.

15.3.1 Rapid GPIO memory map

Table 15-3. RGPIO *Write* memory map

Offset address	Register	Access	Reset	Section/Page
0x00	RGPIO Data Direction Register (RGPIO_DIR)	Write	0x0000	page 243
0x02	RGPIO Write Data Register (RGPIO_DATA)	Write	0x0000	page 244
0x04	RGPIO Pin Enable Register (RGPIO_ENB)	Write	0x0000	page 245
0x06	RGPIO Write Data Clear Register (RGPIO_CLR)	Write	N/A	page 246
0x0A	RGPIO Write Data Set Register (RGPIO_SET)	Write	N/A	page 247
0x0E	RGPIO Write Data Toggle Register (RGPIO_TOG)	Write	N/A	page 248

Table 15-4. RGPIO *Read* memory map

Offset address	Register	Access	Reset	Section/Page
0x00	RGPIO data direction register (RGPIO_DIR)	Read	0x0000	page 243
0x02	RGPIO write data register (RGPIO_DATA)	Read	0x0000	page 244
0x04	RGPIO pin enable register (RGPIO_ENB)	Read	0x0000	page 245
0x06	RGPIO write data register (RGPIO_DATA)	Read	0x0000	page 244
0x08	RGPIO data direction register (RGPIO_DIR)	Read	0x0000	page 243
0x0A	RGPIO write data register (RGPIO_DATA)	Read	0x0000	page 244
0x0C	RGPIO data direction register (RGPIO_DIR)	Read	0x0000	page 243
0x0E	RGPIO write data register (RGPIO_DATA)	R	0x0000	page 244

15.3.2 Rapid GPIO register descriptions

15.3.2.1 RGPIO Data Direction register

The read/write RGPIO_DIR register defines whether a properly-enabled RGPIO pin is configured as an input or output. At reset, all bits in the RGPIO_DIR are cleared. Setting any bit in the RGPIO_DIR register configures a properly-enabled RGPIO port pin as an output, while clearing configures the RGPIO port pin as an input.

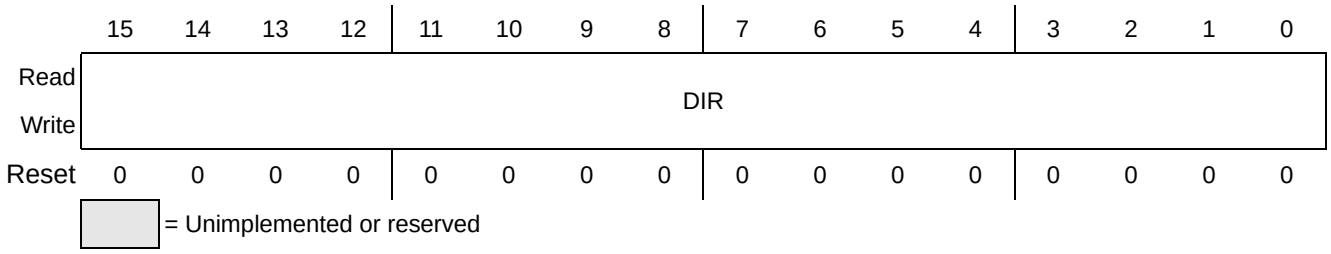


Figure 15-2. RGPIO Data Direction register (RGPIO_DIR)

Table 15-5. RGPIO Data Direction register (RGPIO_DIR) field descriptions

Bit(s)	Field	Description
15–0	DIR	RGPIO data direction. 0 A properly-enabled RGPIO pin is configured as an input 1 A properly-enabled RGPIO pin is configured as an output

15.3.2.2 RGPIO Data register

The RGPIO_DATA register specifies the write data for a properly-enabled RGPIO output pin or the sampled read data value for a properly-enabled input pin. An attempted read of the RGPIO_DATA register returns undefined data for disabled pins because the data value is dependent on the device -level pin muxing and pad implementation. The RGPIO_DATA register is read/write. At reset, all bits in the RGPIO_DATA registers are cleared.

To set bits in an 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.

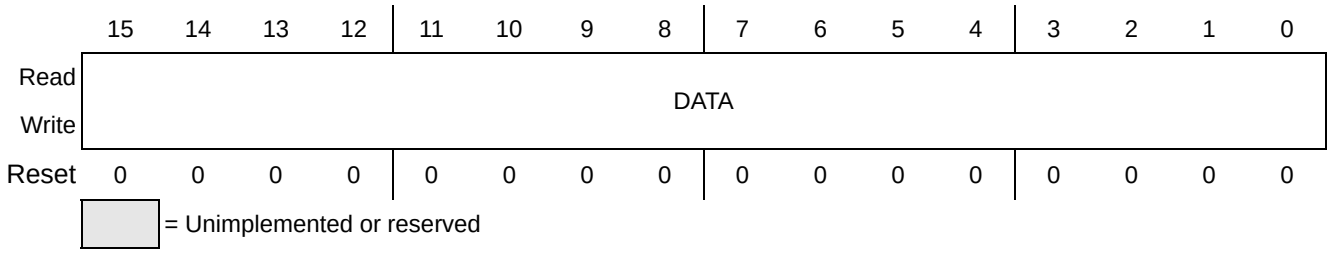


Figure 15-3. RGPIO Data register (RGPIO_DATA)

Table 15-6. RGPIO Data register (RGPIO_DATA) field descriptions

Bit(s)	Field	Description
15–0	DATA	RGPIO Data 0 A properly-enabled RGPIO output pin is driven with a logic 0, or a properly-enabled RGPIO input pin was read as a logic 0 1 A properly-enabled RGPIO output pin is driven with a logic 1, or a properly-enabled RGPIO input pin was read as a logic 1

15.3.2.3 RGPIO Pin Enable register

The RGPIO_ENB register configures the corresponding package pin as a RGPIO pin instead of the normal GPIO pin mapped onto the peripheral bus.

The RGPIO_ENB register is read/write. At reset, all bits in the RGPIO_ENB are cleared, disabling the RGPIO functionality.

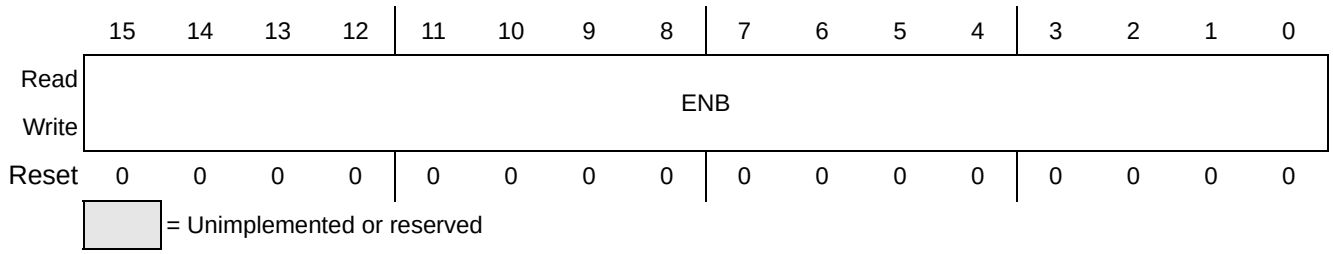


Figure 15-4. RGPIO Enable register (RGPIO_ENB)

Table 15-7. RGPIO Enable register (RGPIO_ENB) field descriptions

Bit(s)	Field	Description
15–0	ENB	RGPIO enable. 0 The corresponding package pin is configured for use as a normal GPIO pin, not a RGPIO 1 The corresponding package pin is configured for use as a RGPIO pin

15.3.2.4 RGPIO Clear Data register

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.

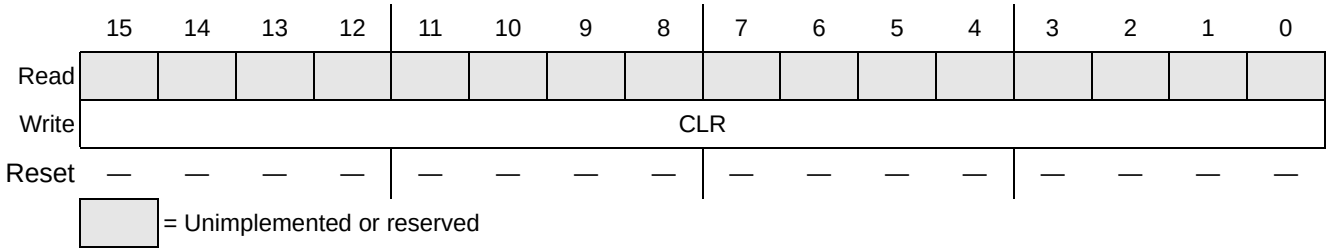


Figure 15-5. RGPIO Clear Data register (RGPIO_CLR)

Table 15-8. RGPIO Clear Data register (RGPIO_CLR) field descriptions

Bit(s)	Field	Description
15–0	CLR	RGPIO clear data. 0 Clears the corresponding bit in the RGPIO_DATA register 1 No effect

15.3.2.5 RGPIO Set Data register

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.

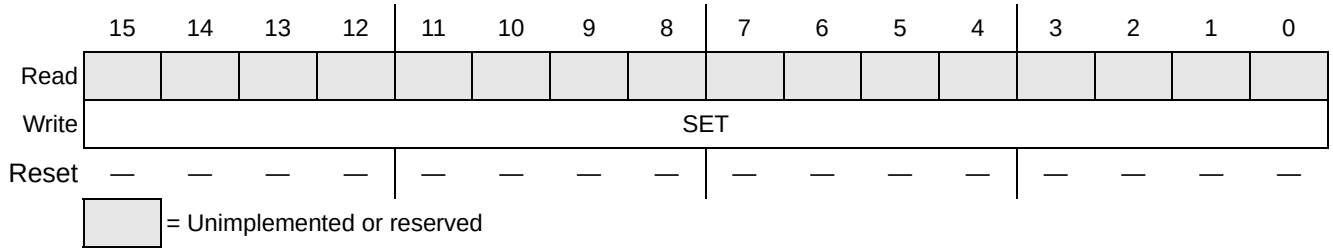


Figure 15-6. RGPIO Set Data register (RGPIO_SET)

Table 15-9. RGPIO Set Data register (RGPIO_SET) field descriptions

Bit(s)	Field	Description
15–0	SET	RGPIO set data. 0 No effect 1 Sets the corresponding bit in the RGPIO_DATA register

15.3.2.6 RGPIO Toggle Data register

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.

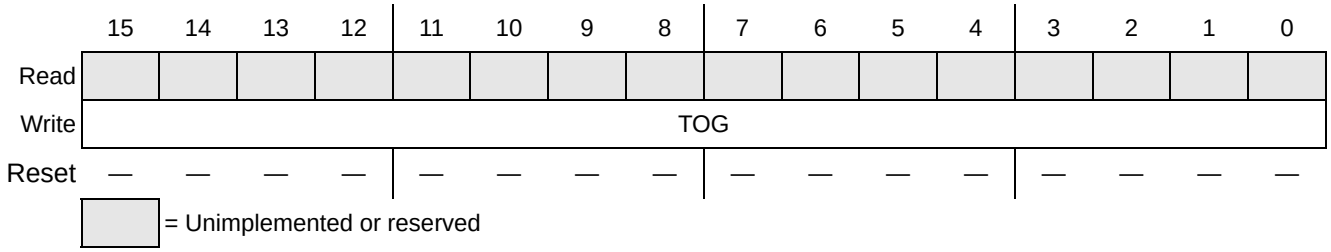


Figure 15-7. RGPIO Toggle Data register (RGPIO_TOG)

Table 15-10. RGPIO Toggle Data register (RGPIO_TOG) field descriptions

Bit(s)	Field	Description
15–0	TOG	RGPIO toggle data. 0 No effect 1 Inverts the corresponding bit in RGPIO_DATA

15.4 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).

15.5 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), configures the pin direction (RGPIO_DIR) and sets the appropriate bits in the pin-enable register (RGPIO_ENB).

15.6 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).

15.6.1 Application 1: Simple square-wave generation

In this example, several different instruction loops are executed, each generating a square-wave output with a 50 percent duty cycle. For this analysis, the executed code is mapped into the processor's RAM.

The following instruction loops were studied:

- BCHG_LOOP — In this loop, a bit change instruction was executed using the GPIO data byte as the operand. This instruction performs a read-modify-write operation and inverts the addressed bit. A pulse counter is decremented until the appropriate number of square-wave pulses have been generated.
- SET+CLR_LOOP — For this construct, two store instructions are executed: one to set the GPIO data pin and another to clear it. Single-cycle, NOP instructions (the tpf opcode) are included to maintain the 50-percent 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 with the relative performance results presented in Table 15-11. The relative performance is stated as a fraction of the processor's operating frequency, defined as *f* MHz. The performance of the BCHG loop, operating on a GPIO output, is selected as the reference.

Table 15-11. Square-wave output performance

Loop	Peripheral Bus-mapped GPIO			RGPIO		
	Square-wave frequency	Frequency at CPU *f* = 50 MHz	Relative speed	Square-wave frequency	Frequency at 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-percent duty cycle.

15.6.2 Application 2: 16-bit message transmission using SPI protocol

In this second example, a 16-bit message is transmitted using three programmable output pins. The output pins include a serial clock, an active-high chip select and the serial data bit. The software is configured to sample the serial data bit at the rising edge of the clock with the data sent in a most-significant to least-significant bit order. The resulting 3-bit output is shown in Figure 15-8.

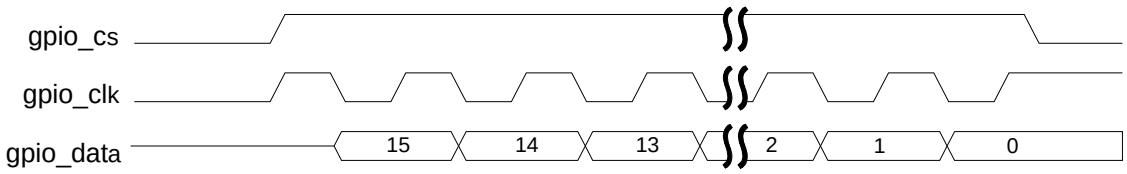


Figure 15-8. GPIO SPI example timing diagram

For this example, the processing of the SPI message is considerably more complex than the generation of a simple square wave of the previous example. The code snippet used to extract the data bit from the message and build the required GPIO data register writes is shown in Example 15-1.

Example 15-1. Sixteen-bit message using SPI protocol

```
# 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 Table 15-12.

The statistics below are applicable to a variety of ColdFire V1 devices, but not to the MMA955xL platform, which has a maximum CPU clock rate of 8 MHz.

Table 15-12. Emulated SPI performance using GPIO outputs

Peripheral, Bus-Mapped GPIO		RGPIO	
SPI speed at CPU *f* = 50 MHz	Relative speed	SPI speed at CPU *f* = 50 MHz	Relative speed
2.063 Mbps	1.00x	3.809 Mbps	1.29x

Chapter 16 Pin Interrupt Function

16.1 Overview

The IRQ (external interrupt) module provides an interrupt input.

16.2 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 interrupt sensitivity – Any of the following:
  - Programmable falling edge (or rising edge) only
  - Both falling edge and low level
  - Both rising edge and high level
- Exit from low-power modes
- Wake-up request to internal module(s) independent of interrupt enable
- Pin-level signal provided to core for BIL/BIH instruction when IRQPE is set
- Software enable/disable of on-chip, pull-up/pull-down done on IRQ pin

16.3 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.

16.4 Block diagram

The block diagram of the IRQ module is given in Figure 16-1.

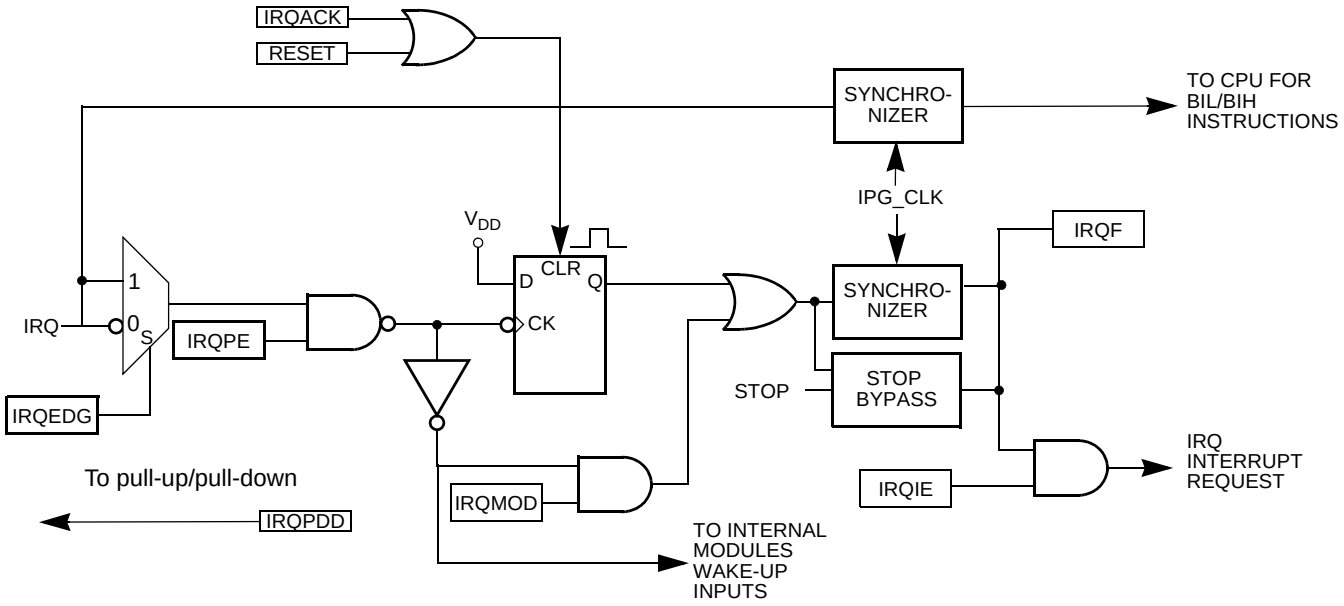


Figure 16-1. IRQ block diagram

16.5 Pin Interrupt signal description

Table 16-1 shows the single, user-accessible signal for the IRQ module.

.

Table 16-1. Signal properties

Name	Function	Reset state
IRQ	External interrupt pin	input

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.

16.6 Pin Interrupt memory map and registers

This section provides a detailed description of the IRQ register that is accessible to the end user.

16.6.1 Pin Interrupt memory map

The following table shows the register contained in the IRQ module.

Table 16-2. Pin Interrupt memory map

Offset address	Register	Access	Reset	Details
00	Interrupt Status and Control (IRQSC)	Read/Write	00h	Section 16.6.2.1

16.6.2 Register descriptions

This section consists of the IRQ register descriptions in address order.

16.6.2.1 Interrupt Status and Control register

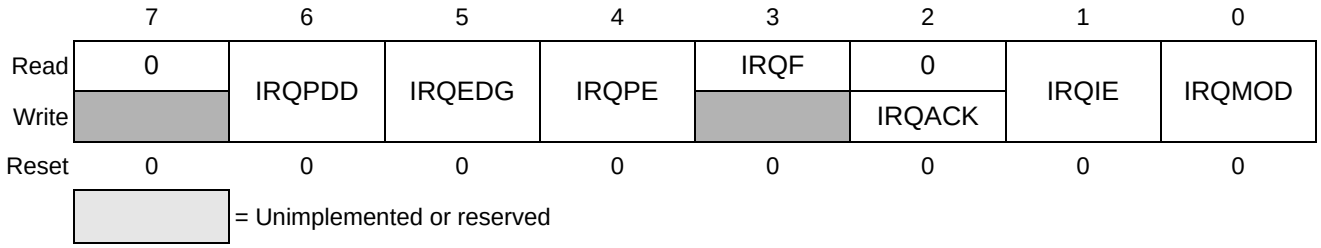


Figure 16-2. Interrupt Status and Control register (IRQSC)

Table 16-3. IRQSC Field Descriptions

Bits	Name	Description
7	—	Reserved
6	IRQPDD	IRQ Pull Device Disable Bit The IRQPDD bit is used to disable the on-chip, pull-up/pull-down device on the IRQ pin. This allows users to have an external device if required for their application. 0 On-chip pull-up/pull-down device is enabled 1 On-chip pull-up/pull-down device is disabled
5	IRQEDG	IRQ Edge Select Bit The IRQEDG 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 Bit The IRQPE bit determines whether the IRQ pin is enabled. 0 IRQ pin not enabled 1 IRQ pin enabled

Table 16-3. IRQSC Field Descriptions (continued)

Bits	Name	Description
3	IRQF	IRQ Flag Bit This IRQF bit indicates when an IRQ interrupt is detected. 0 No IRQ interrupt detected 1 IRQ interrupt detected
2	IRQACK	IRQ Acknowledge Bit 0 IRQACK always reads as logic 0. 1 Writing a logic 1 to the IRQACK bit is part of the flag-clearing mechanism. For more information on flag-clearing, see “Clearing an IRQ interrupt request” on page 257.
1	IRQIE	IRQ Interrupt Enable Bit The IRQIE bit determines whether an IRQ interrupt request is enabled. 0 IRQ interrupt requests not enabled. 1 IRQ interrupt request enabled.
0	IRQMOD	IRQ Detection Mode Bit The IRQMOD bit (along with the IRQEDG bit) controls the detection mode of the IRQ pin. **Note:** When pin 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 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.

16.7 Functional description

This section provides a complete functional description of the IRQ module.

16.7.1 External interrupt pin

Writing to the IRQPE bit in the IRQSC register, enables or disables the IRQ pin.

16.7.2 IRQ edge select

The IRQEDG bit in the IRQSC register determines if the IRQ pin is either sensitive to the falling edge and low level or the rising edge and high level.

16.7.3 IRQ sensitivity

The IRQMOD bit in the IRQSC 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 will set the IRQF bit. The IRQF bit will remain set as long as the IRQ pin remains asserted.

16.7.4 IRQ interrupts

The IRQ module can provide a source of interrupts. To cause a IRQ module interrupt request, the following must occur:

- The IRQIE bit in the IRQSC register must be set.
- The IRQF bit in the IRQSC 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.

NOTE

Changing the IRQMOD or IRQEDG bits while IRQPE bit is enabled may cause spurious interrupt and set the IRQF bit or cause an interrupt.

16.7.5 Clearing an IRQ interrupt request

If the IRQ module interrupt pin is either both falling edge and low level sensitive or rising edge and high level sensitive, both of the following actions must occur to clear a IRQ interrupt request:

- Software provides an interrupt-acknowledge by writing a logic 1 to the IRQACK bit in the IRQSC register.
- Either of the following happens:
  - The IRQ pin returns to a de-asserted logic state.
  - The IRQ pin is disabled using the IRQPE bit.

If the IRQ module interrupt pin sensitive only to the falling (or rising) edge. Writing a logic 1 to the IRQACK bit in the IRQSC register immediately clears the IRQ interrupt request even if the enabled IRQ pin remains asserted.

NOTE

The IRQ flag cannot be cleared as long as the pin is asserted during edge-sensitive mode.

16.8 Exit from low-power modes

The IRQ interrupt, if enabled, can provide a means to exit CPU low-power modes ($STOP_{FC}$, $STOP_{SC}$ and $STOP_{NC}$). If the detection mode sensitivity is set to both falling (or rising) edge and low (or high) level and the IRQ pin is enabled and low upon entering STOP, 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 STOP modes.

16.8.1 STOP

Subject to the settings of the SIM peripheral clock enable registers, the IRQ module remains active in $STOP_{FC}$ and $STOP_{SC}$ modes. Setting the IRQIE bit in the IRQSC register enables the IRQ interrupt request. Any detected IRQ interrupt will bring the CPU out of STOP mode.

16.9 Resets

The IRQ interrupt is disabled after reset. The IRQ module cannot cause a MCU reset.

16.10 Interrupts

The IRQ module generates a single interrupt.

The IRQ interrupt is listed in Table 16-4 which shows the interrupt name and the name of the local enable that can be used to disable a IRQ interrupt request.

Table 16-4. 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 17 16-Bit Modulo Timer

17.1 Introduction

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 this device, 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 MMA955xL platform. 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 structure” on page 35, MTIM activity is restricted to Φ_D.

17.2 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 the 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

17.2.1 Block diagram

The block diagram for the modulo timer module is shown Figure 17-1.

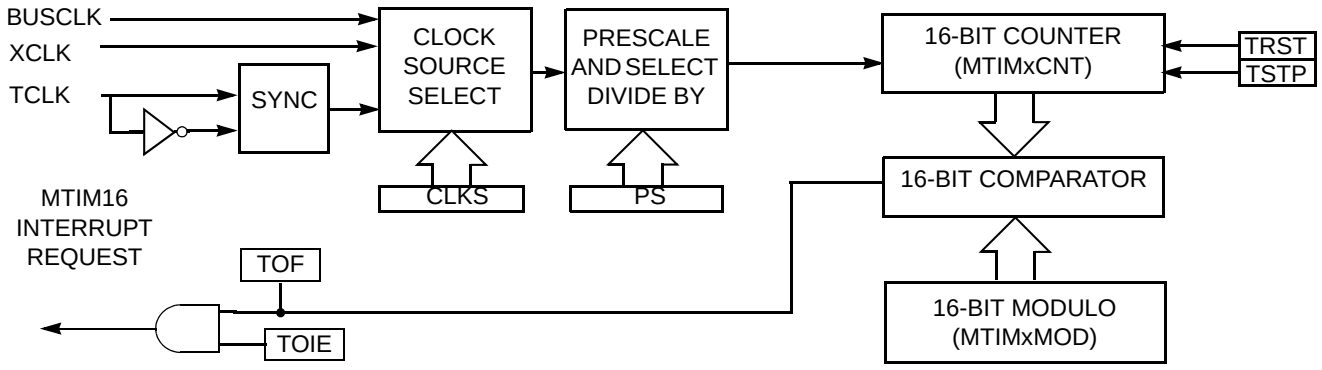


Figure 17-1. Modulo timer block diagram

17.2.2 Modes of operation

This section defines MTIM16 operation in stop and background debug modes.

17.2.2.1 MTIM16 in stop modes

All clocked modules, including MTIM16, are inactive in $STOP_{NC}$. Operation in other modes is governed by the PCESFC, PCESSC and PCERUN registers in the SIM.

17.2.2.2 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).

17.3 Model-timer memory map/registers

The MTIM16 includes four registers:

- An 8-bit, status-and-control register
- An 8-bit, clock-configuration register
- A 16-bit, counter register
- A 16-bit modulo register

For the absolute address assignments for all MTIM16 registers, see the direct-page register summary in this section. This section refers to registers and control bits only by their names and relative address offsets.

Some MCUs may have more than one MTIM16, so register names include placeholder characters to identify the specific MTIM16.

17.3.1 MTIM16 memory map

Table 17-1. MTIM16 Memory Map

Offset address	Register	Access	Reset	Details
0x0000	MTIM16 Status and Control Register (MTIMxSC)	Read/Write	0x10	page 262
0x0001	MTIM16 Clock Configuration Register (MTIMxCLK)	Read/Write	0x00	page 263
0x0002	MTIM16 Counter Register High (MTIMxCNTH)	Read	0x00	page 264
0x0003	MTIM16 Counter Register Low (MTIMxCNTL)	Read	0x00	page 264
0x0004	MTIM16 Modulo Register High (MTIMxMODH)	Read/Write	0x00	page 265
0x0005	MTIM16 Modulo Register Low (MTIMxMODL)	Read/Write	0x00	page 265

17.3.2 MTM16 registers

17.3.2.1 MTIM16 Status and Control register

MTIMxSC contains the overflow status flag and control bits. These are used to configure the interrupt enable, reset the counter and stop the counter.

	7	6	5	4	3	2	1	0
Read	TOF	TOIE	0	TSTP	0	0	0	0
Write			TRST					
Reset	0	0	0	1	0	0	0	0

= Unimplemented or reserved

Figure 17-2. MTIM16 Status and Control register (MTIMxSC)

Table 17-2. MTIM16 Status and Control register (MTIMxSC) field descriptions

Bit(s)	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 MTIMSC register while TOF is set and writing a 0 to TOF. Writing a 1 has no effect. TOF is also cleared when TRST is written to a 1. 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. Clear TOF first, then set TOIE. 0 TOF interrupts are disabled. Use software polling. 1 TOF interrupts are enabled.
5	TRST	MTIM16 Counter Reset. When a 1 is written to this write-only bit, the MTIM16 counter register resets to 0x0000 and TOF is cleared. Writing a 1 to this bit also makes 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.
3–0	—	Unused register bits. Always read 0.

17.3.2.2 MTIM16 Clock Configuration Register

MTIMxCLK contains the clock select bits (CLKS) and the prescaler select bits (PS).

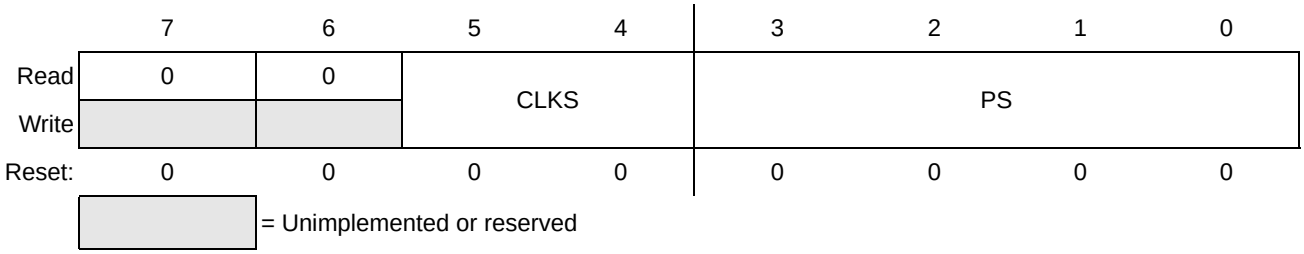


Figure 17-3. MTIM16 Clock Configuration register (MTIMxCLK)

Table 17-3. MTIM16 Clock Configuration register (MTIMxCLK) field descriptions

Bit(s)	Field	Description
7–6	—	Unused register bits. Always read 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. 00 Encoding 0. Bus clock (BUSCLK) 01 Encoding 1. Fixed-frequency clock (XCLK) 10 Encoding 3. Not available 11 Encoding 4. Not available
3–0	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. All encodings, other than the following, default to MTIM16 clock source ÷ 256. 0000 Encoding 0. MTIM16 clock source [3] 1 0001 Encoding 1. MTIM 16clock 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 1000 Encoding 8. MTIM16 clock source ÷ 256

17.3.2.3 MTIM16 Counter Register High/Low

MTIMxCNTH is the read-only value of the high byte of current MTIM16 16-bit counter.

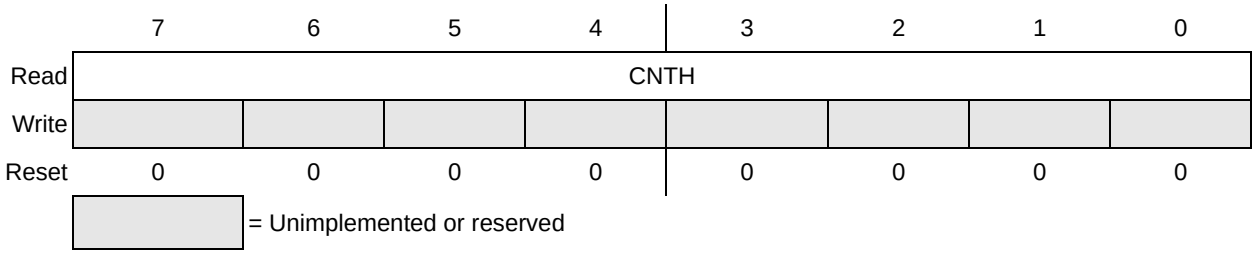


Figure 17-4. MTIM16 Counter Register High (MTIMxCNTH)

Table 17-4. MTIM16 Counter Register High (MTIMxCNTH) field descriptions

Bit(s)	Field	Description
7–0	CNTH	MTIM16 Count (High Byte). These eight read-only bits contain the current, high-byte value of the 16-bit counter. Writing has no effect to this register. Reset clears the register to 0x00.

MTIMxCNTL is the read-only value of the low byte of the current MTIM16 16-bit counter.

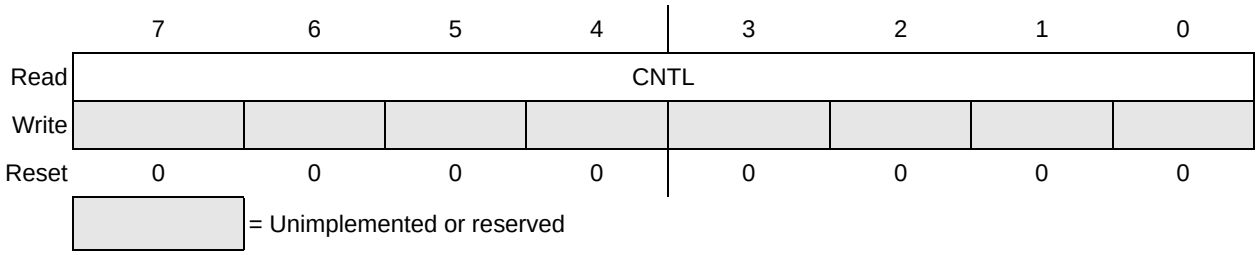


Figure 17-5. MTIM16 Counter Register Low (MTIMxCNTL)

Table 17-5. MTIM16 Counter Register Low (MTIMxCNTL) field descriptions

Bit(s)	Field	Description
7–0	CNTL	MTIM16 Count (Low Byte). These eight read-only bits contain the current, low-byte value of the 16-bit counter. Writing has no effect to this register. Reset clears the register to 0x00.

When either MTIMxCNTH or MTIMxCNTL 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 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 setting of TRST bit of MTIMxSC 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 the 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 will be read after returning to normal execution. The value read from the MTIMxCNTH and MTIMxCNTL registers in BDM mode is the value of these registers and not the value of their read buffer.

17.3.2.4 MTIM16 Modulo Register High/Low registers

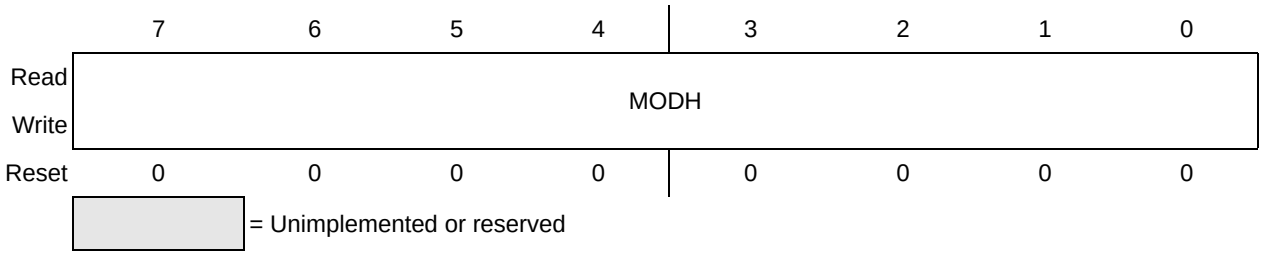


Figure 17-6. MTIM16 Modulo Register High (MTIMxMODH)

Table 17-6. MTIM16 Modulo Register High (MTIMxMODH) field descriptions

Bit(s)	Field	Description
7–0	MODH	MTIM16 Modulo (High Byte). These eight read/write bits contain the modulo high-byte value used to reset the counter and set TOF. Reset sets the register to 0x00.

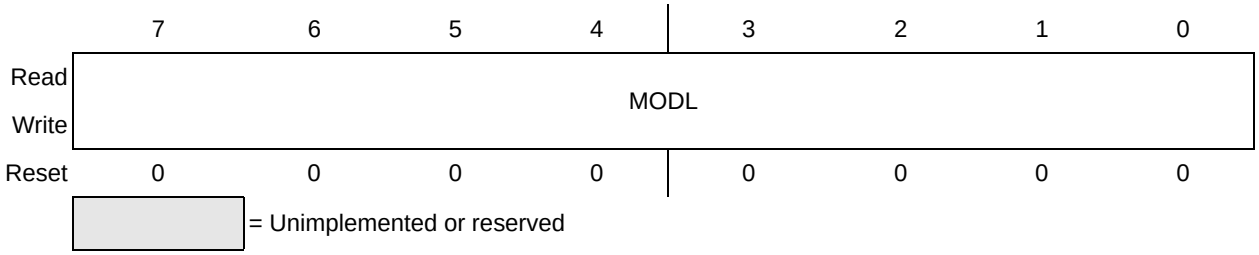


Figure 17-7. MTIM16 Modulo Register Low (MTIMxMODL)

Table 17-7. Modulo Register Low bit descriptions

Bit(s)	Field	Description
7–0	MODL	MTIM16 Modulo (Low Byte). These eight read/write bits contain the modulo low-byte value used to reset the counter and set TOF. Reset sets the register to 0x00.

A value of 0x0000 in MTIMxMODH:L puts the MTIM16 in free-running mode. Writing to either MTIMxMODH or MTIMxMODL latches the value into a buffer and the registers are updated with the value of their write buffer after the second byte-writing. The updated MTIMxMODH:L will take effect in the next MITIM16 counter cycle except for the first writing of modulo after a chip reset or in BDM mode. After a software reset, however, the MTIMxMODH:L takes effect at once, even if it did not take effect

before the reset. On the first writing of MTIMxMODH:L 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 MTIMxSC register (whether BDM is active or not).

When BDM is active, the coherency mechanism is frozen such that the buffer latches remain in the state they were in when the BDM became active, even if one or both halves of the modulo register are written while BDM is active. Any writing to the modulo registers bypasses the buffer latches and writes directly to the modulo register while BDM is active. The counter is cleared at the same time.

Reading of MTIMxMODH:L returns the modulo value which is taking effect whenever in normal run mode or in BDM mode.

17.4 Functional description

The MTIM16 is composed of a main, 16-bit up-counter with a 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 (MTIMxCNTH:L) 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. The modulo also is reset 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 register (MTIMxSC) and clear the MTIM16 stop bit (TSTP).

Two clock sources are software selectable: the internal bus clock, the fixed-frequency clock (XCLK). The MTIM16 clock-select bits (CLKS1:CLKS0) in MTIMxSC are used to select the desired clock source. If the counter is active (TSTP = 0) when a new clock source is selected, the counter continues counting from the previous value using the new clock source.

Nine prescale values are software-selectable: the clock source divided by 1, 2, 4, 8, 16, 32, 64, 128 or 256. The prescaler select bits (PS[3:0]) in MTIMxSC 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 (MTIMxMODH:L) 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 (TSTP = 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. First, the MTIMxSC register is read and the TOF set. The second step writes 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 allows for an optional interrupt to be generated whenever TOF is set. To enable the MTIM16 overflow interrupt, set the MTIM16 overflow interrupt enable bit (TOIE) in MTIMxSC. TOIE should never be written to a 1 while TOF = 1. Instead, TOF should be cleared first, then the TOIE can be set to 1.

17.4.1 MTIM16 operation example

This section shows an example of the MTIM16 operation as the counter reaches a matching value from the modulo register.

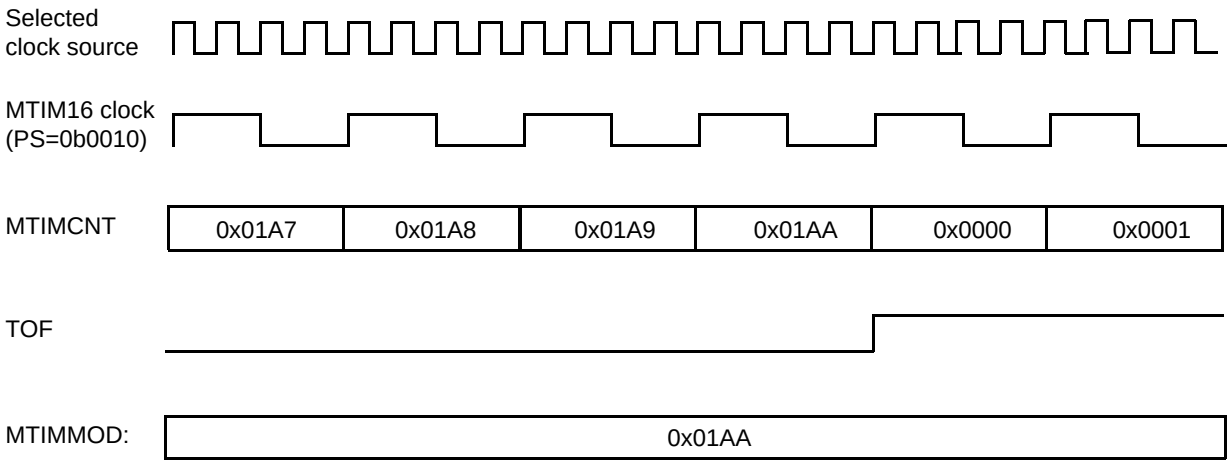


Figure 17-8. MTIM16 counter overflow example

Figure 17-8, the selected clock source could be any of the two possible choices. The prescaler is set to PS =0b0010 or divide-by-4. The modulo value in the MTIMxMODH:L register is set to 0x01AA. When the counter reaches the modulo value of 0x01AA, it overflows to 0x0000 and continues counting. The timer overflow flag, TOF, sets when the counter value changes from 0x01AA to 0x0000. An MTIM16 overflow interrupt is generated when TOF is set, if TOIE = 1.

Chapter 18 Timer/PWM Module

The MMA955xL platform 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 also can 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 MMA955xL device. 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. Use of the fixed system clock is discouraged for applications that require utmost sensor accuracy. 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 structure” on page 35, TPM activity is restricted to Φ_D.

18.1 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.

18.1.1 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

18.1.2 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.

18.1.2.1 Input-capture mode

When a selected edge event occurs on the associated MCU pin, the current value of the 16-bit, timer counter is captured into the channel value register and an interrupt flag bit is set. Rising edges, falling edges, any edge or no edge (disable channel) are selected as the active edge that triggers the input capture.

18.1.2.2 Output-compare mode

When the value in the timer-counter register matches the channel value register, an interrupt flag bit is set and a selected output action is forced on the associated MCU pin. The output-compare action is selected to force the pin to 0, force the pin to 1, toggle the pin or ignore the pin (used for software-timing functions).

18.1.2.3 Edge-aligned PWM mode

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.

18.1.2.4 Center-aligned PWM mode

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.

18.1.3 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 “Pins and Connections” on page 25.”)

Figure 18-1 on page 272 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.

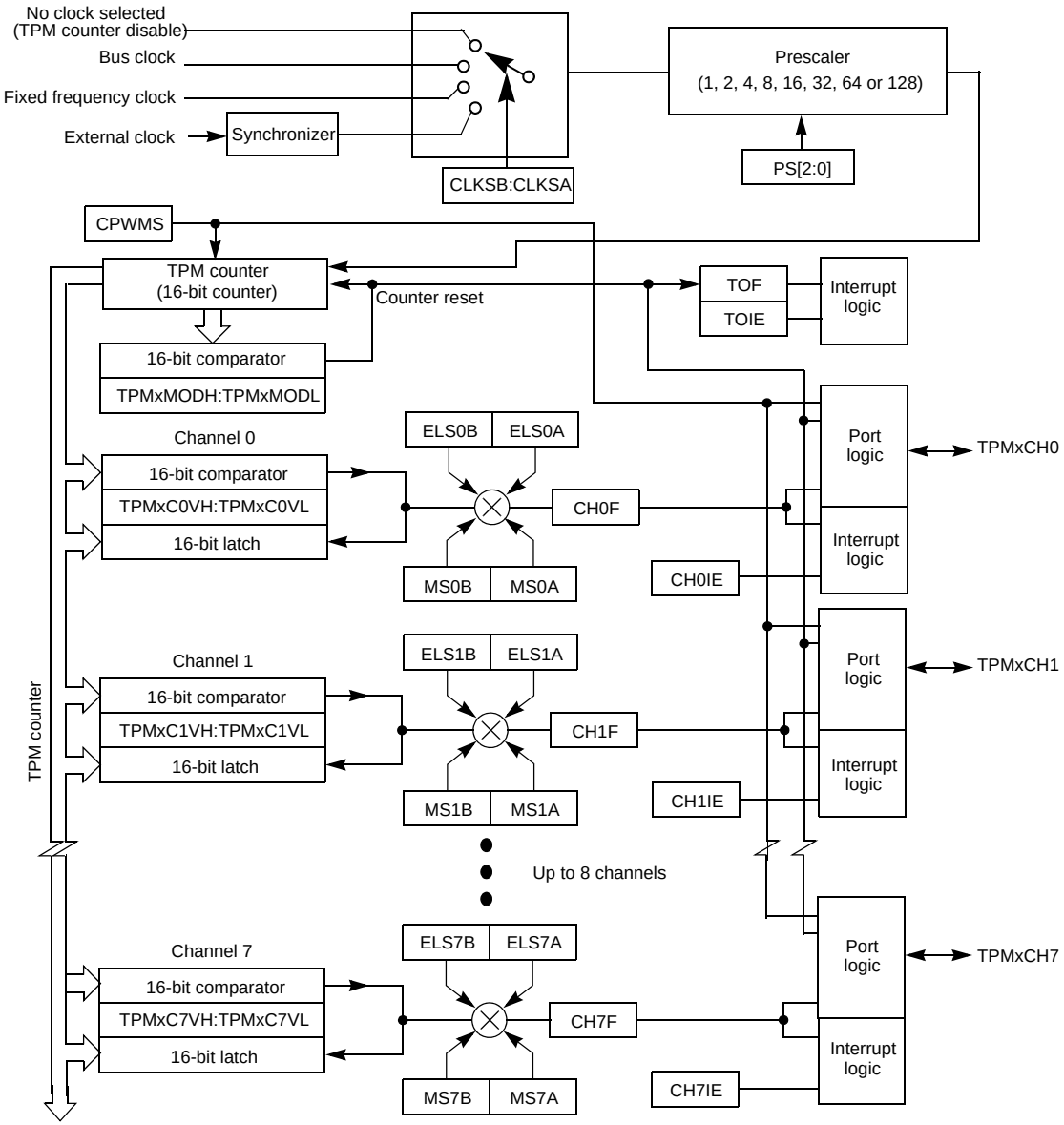


Figure 18-1. Timer/PWM block diagram

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.

18.2 Timer/PWM signal description

Table 18-1 shows the user-accessible signals for the TPM. The number of channels are varied from one to eight. When an external clock is included, it can be shared with the same pin as any TPM channel; however, it could be connected to a separate input pin. For the specific chip implementation, see Table 3-1 on page 26.

Table 18-1. Signal properties

Name	Function
TPMxCH*n*[1]	I/O pin associated with TPM channel *n*.

1 *n* = channel number (1–8).

18.2.1 Timer/PWM detailed signal descriptions

18.2.1.1 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.

TPMxMODH:TPMxMODL = 0x0008
TPMxC*n*VH:TPMxC*n*VL = 0x0005

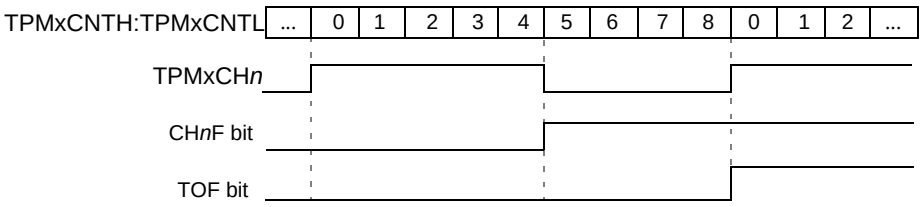


Figure 18-2. High-true pulse of an edge-aligned PWM

TPMxMODH:TPMxMODL = 0x0008
TPMxC*n*VH:TPMxC*n*VL = 0x0005

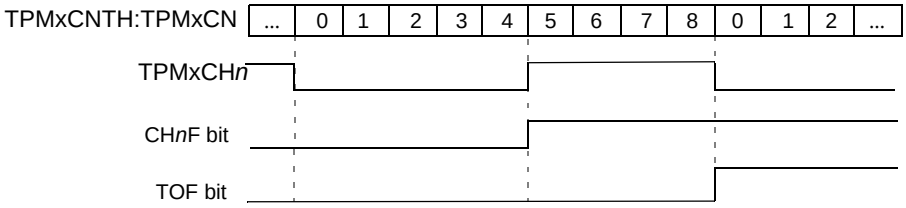


Figure 18-3. Low-true pulse of an edge-aligned PWM

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

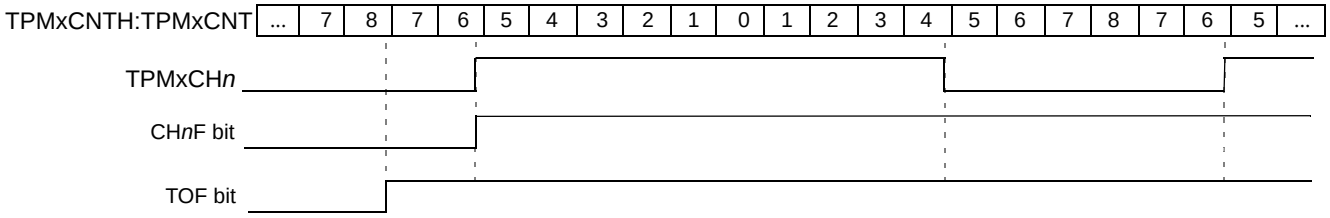


Figure 18-4. High-true pulse of a center-aligned PWM

TPMxMODH:TPMxMODL = 0x0008
TPMxC*n*VH:TPMxC*n*VL = 0x0005

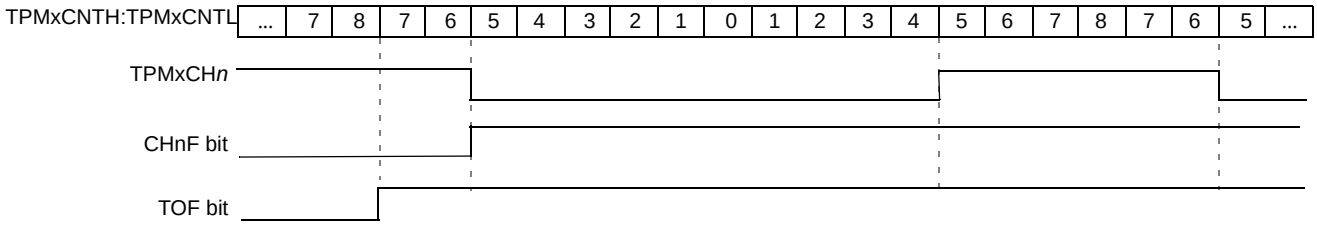


Figure 18-5. Low-True Pulse of a Center-Aligned PWM

18.3 Timer/PWM memory map/register descriptions

18.3.1 Timer/PWM memory map

Table 18-2. Timer/PWM memory map

Offset address	Register	Access	Reset	Details
0x0000	TPM Status and Control Register (TPMxSC)	Read/Write	0x00	page 277
0x0001	TPM Counter Register High (TPMxCNTH)	Read	0x00	page 280
0x0002	TPM Counter Register Low (TPMxCNTL)	Read	0x00	page 281
0x0003	TPM Counter Modulo Register High (TPMxMODH)	Read/Write	0x00	page 282
0x0004	TPM Counter Modulo Register Low (TPMxMODHL)	Read/Write	0x00	page 283
0x0005	TPM Channel *n* Status and Control Register (TPMxC*n*SC)	Read/Write	0x00	page 284
0x0008	TPM Channel Value Register High (TPMxC*n*VH)	Read/Write	0x00	page 286
0x000B	TPM Channel Value Register Low (TPMxC*n*VL)	Read/Write	0x00	page 287

18.3.2 Timer/PWM register descriptions

18.3.2.1 TPM Status and Control Register

TPMxSC contains the overflow status flag and control bits used to configure the interrupt enable, TPM configuration, clock source, and prescale factor. These controls relate to all channels within this timer module.

	7	6	5	4	3	2	1	0
Read Write	TOF	TOIE	CPWMS	CLKS1	CLKS0	PS2	PS1	PS0
Reset	0	0	0	0	0	0	0	0

= Unimplemented or reserved

Figure 18-6. TPM Status and Control Register (TPMxSC)

Table 18-3. TPM Status and Control Register (TPMxSC) field descriptions

Bit(s)	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 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	CLKS1	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

Table 18-3. TPM Status and Control Register (TPMxSC) field descriptions (continued)

Bit(s)	Field	Description
3	CLKS0	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	PS	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 2 001 TPM clock divided by 4 001 TPM clock divided by 8 001 TPM clock divided by 16 001 TPM clock divided by 32 001 TPM clock divided by 64 001 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 2 001 TPM clock divided by 4 001 TPM clock divided by 8 001 TPM clock divided by 16 001 TPM clock divided by 32 001 TPM clock divided by 64 001 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. 000 TPM clock divided by 1 001 TPM clock divided by 2 001 TPM clock divided by 4 001 TPM clock divided by 8 001 TPM clock divided by 16 001 TPM clock divided by 32 001 TPM clock divided by 64 001 TPM clock divided by 128

Table 18-4. Timer/PWM clock selection

CLKSB:CLKSA	TPM clock to prescaler input
00	No clock selected (TPM counter disable)
01	Bus clock
10	Fixed frequency clock
11	UNAVAILABLE

Table 18-5. Prescale factor selection

PS[2:0]	TPM clock divided-by
000	1
001	2
010	4
011	8
100	16
101	32
110	64
111	128

18.3.2.2 TPM Counter Register High

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.

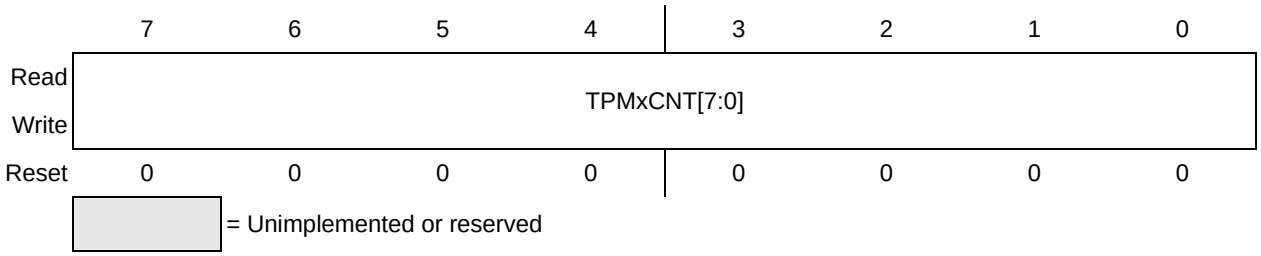


Figure 18-7. TPM Counter Register High (TPMxCNTH)

Table 18-6. TPM Counter Register High (TPMxCNTH) field descriptions

Bit(s)	Field	Description
7-0	TPMxCNT	See full description above.

18.3.2.3 TPM Counter Register Low

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.

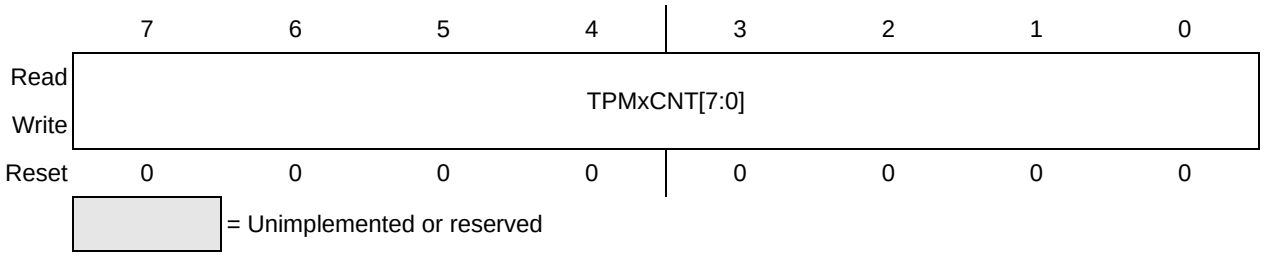


Figure 18-8. Timer/PWM Counter Register Low (TPMxCNTL)

Table 18-7. Timer/PWM Counter Register Low (TPMxCNTL) field descriptions

Bit(s)	Field	Description
7-0	TPMxCNTL	See full description above.

18.3.2.4 TPM Counter Modulo Register High

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.

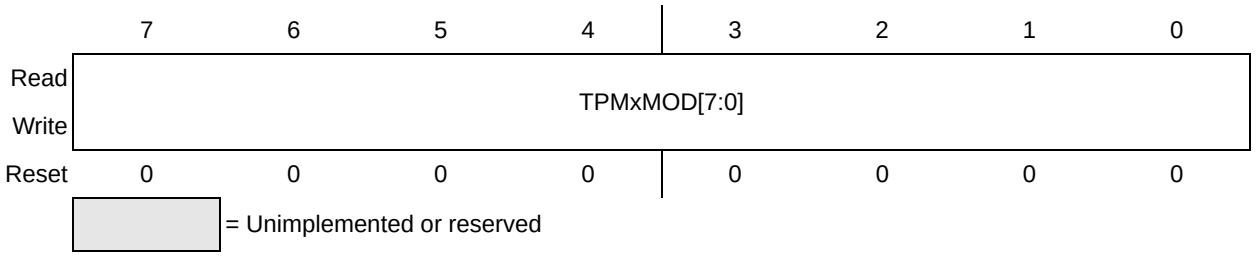


Figure 18-9. Timer/PWM Counter Modulo Register High (TPMxMODH)

Table 18-8. Timer/PWM Counter Modulo Register High (TPMxMODH) field descriptions

Bit(s)	Field	Description
7-0	TPMxMODH	See full description above.

18.3.2.5 TPM Counter Modulo Register Low

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.

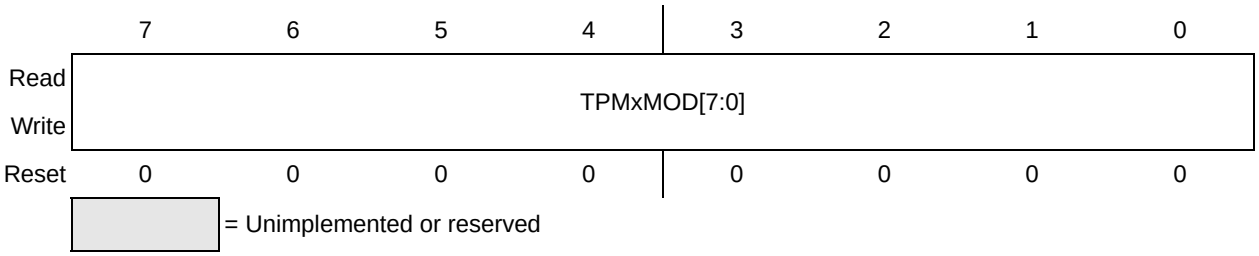


Figure 18-10. Timer/PWM Counter Modulo Register Low (TPMxMODL)

Table 18-9. Timer/PWM Counter Modulo Register Low (TPMxMODL) field descriptions

Bit(s)	Field	Description
7-0	TPMxMODL	See full description above.

18.3.2.6 TPM Channel *n* Status and Control Register

TPMxC*n*SC contains the channel-interrupt-status flag and control bits that configure the interrupt enable, channel configuration and pin function.

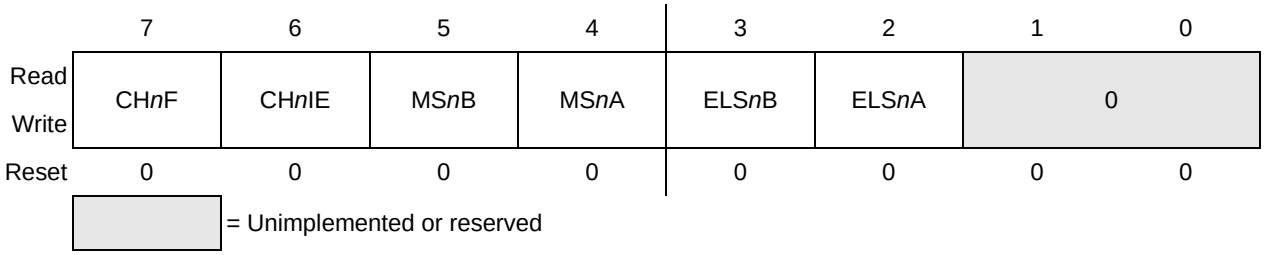


Figure 18-11. Timer/PWM Channel *n* Status and Control Register (TPMxCnSC)

Table 18-10. Timer/PWM Channel *n* Status and Control Register (TPMxCnSC) field descriptions

Bit(s)	Field	Description
7	CH*n*F	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. 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	CH*n*IE	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	MS*n*B	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	MS*n*A	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.

Table 18-10. Timer/PWM Channel *n* Status and Control Register (TPMxCnSC) field descriptions (continued)

Bit(s)	Field	Description
3	ELS*n*B	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	ELS*n*A	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.
1-0	—	Reserved or Unimplemented

Table 18-11. 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)

18.3.2.7 TPM Channel Value Register High

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).

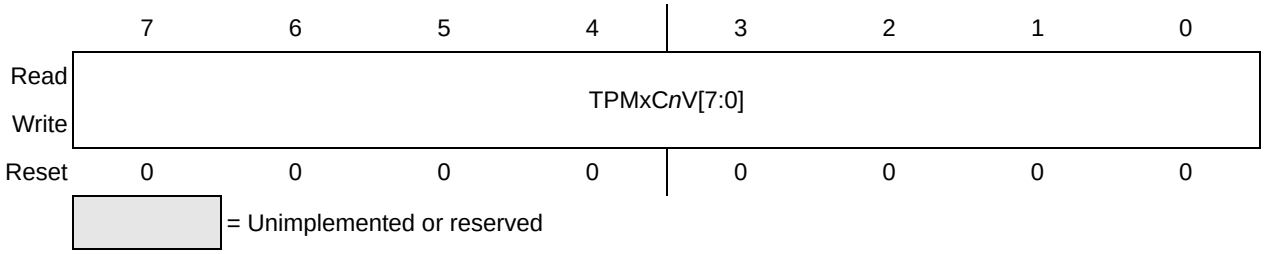


Figure 18-12. Timer/PWM Channel Value Register High (TPMxCnVH)

Table 18-12. Timer/PWM Channel Value Register High (TPMxCnVH) field descriptions

Bit(s)	Field	Description
7-0	TPMxC*n*VH	See full description above.

18.3.2.8 TPM Channel Value Register Low

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).

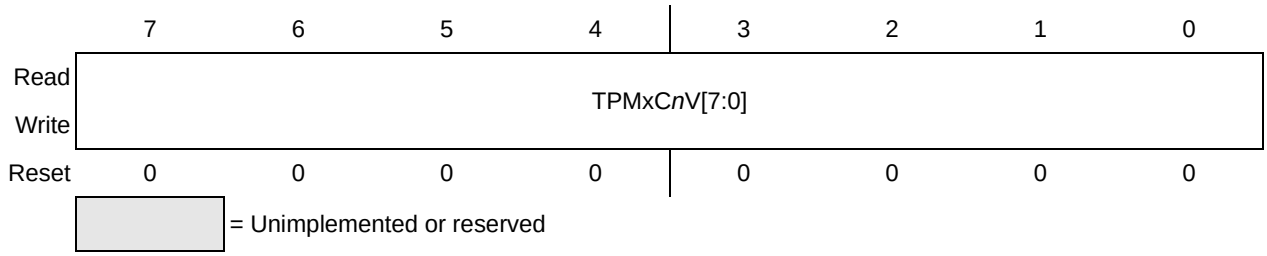


Figure 18-13. Timer/PWM Channel Value Register Low (TPMxCnVL)

Table 18-13. Timer/PWM Channel Value Register Low (TPMxCnVL) field descriptions

Bit(s)	Field	Description
7-0	TPMxC*n*VL	See full description above.

18.4 Functional description

All TPM functions are associated with a central, 16-bit counter that allows flexible selection of the clock and prescale factor. There is also a 16-bit, modulo register associated with this counter.

The CPWMS control bit chooses between center-aligned PWM operation for all channels in the TPM (CPWMS = 1) or general-purpose timing functions (CPWMS = 0) where each channel can independently be configured to operate in input capture, output compare or edge-aligned PWM mode. The CPWMS control bit is located in the TPM status and control register because it affects all channels within the TPM and influences the way the main counter operates. (In CPWM mode, the counter changes to an up/down mode rather than the up-counting mode used for general-purpose timer functions.)

The following sections describe TPM counter and each of the timer operating modes (input capture, output compare, edge-aligned PWM and center-aligned PWM). Because details of pin operation and interrupt activity depend upon the operating mode, these topics are covered in the associated mode explanation sections.

18.4.1 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.

18.4.1.1 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 (Table 18-4). 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 Chapter 12, “On-Chip Oscillator”. 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.

18.4.1.2 Counter overflow and modulo reset

An interrupt flag and enable are associated with the 16-bit, main counter. The flag (TOF) is a software-accessible indication that the timer counter has overflowed. The enable signal selects between software polling (TOIE = 0)—where no interrupt is generated—or interrupt-driven operation (TOIE = 1)—where the interrupt is generated whenever the TOF is set.

The conditions causing TOF to become set depend on whether the TPM is configured for center-aligned PWM (CPWMS = 1). If CPWMS is cleared and there is no modulus limit, the 16-bit timer counter counts

from 0x0000 through 0xFFFF and overflows to 0x0000 on the next counting clock. TOF is set at the transition from 0xFFFF to 0x0000. When a modulus limit is set, TOF is set at the transition from the value set in the modulus register to 0x0000.

When the TPM is in center-aligned PWM mode (CPWMS = 1), the TOF flag is set as the counter changes direction at the end of the count value set in the modulus register (at the transition from the value set in the modulus register to the next-lower count value). This corresponds to the end of a PWM period. (The 0x0000 count value corresponds to the center of a period.)

18.4.1.3 Counting modes

The main timer counter has two counting modes. When center-aligned PWM is selected (CPWMS = 1), the counter operates in up/down counting mode. Otherwise, the counter operates as a simple up-counter. As an up counter, the timer counter counts from 0x0000 through its terminal count and continues with 0x0000. The terminal count is 0xFFFF or a modulus value in TPMxMODH:TPMxMODL.

When center-aligned PWM operation is specified, the counter counts up from 0x0000 through its terminal count and then down to 0x0000 where it changes back to up-counting. The terminal count value and 0x0000 are normal length counts (one timer clock period long). In this mode, the Timer Overflow Flag (TOF) is set at the end of the terminal-count period (as the count changes to the next-lower count value).

18.4.1.4 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.

18.4.2 Channel-mode selection

If CPWMS is cleared, the MS*n*B and MS*n*A bits determine the basic mode of operation for the corresponding channel. Choices include input capture, output compare and edge-aligned PWM.

18.4.2.1 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.

18.4.2.2 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.

18.4.2.3 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). The polarity of this PWM signal is determined by ELS*n*A bit. Zero-percent and 100-percent duty-cycle cases are possible.

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 18-14). If ELS*n*A is cleared, the counter overflow forces the PWM signal high and the channel match forces the PWM signal low. If ELS*n*A is set, the counter overflow forces the PWM signal low and the channel-match forces the PWM signal high.

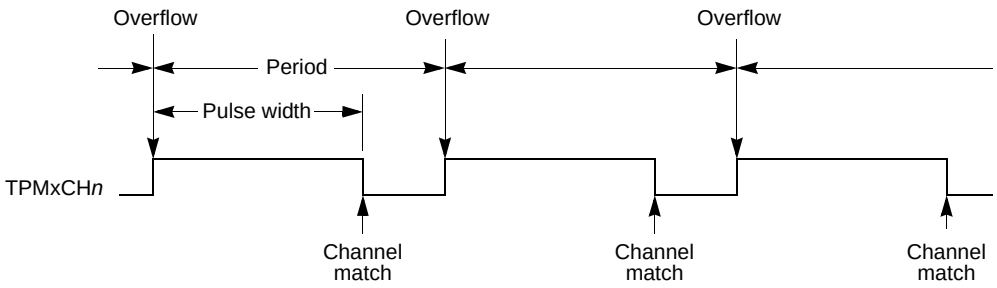


Figure 18-14. EPWM Period and Pulse Width (ELSnA = 0)

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 the modulus setting must be less than 0xFFFF in order to get a 100-percent duty cycle.

The timer-channel registers are buffered to ensure coherent, 16-bit updates and to avoid unexpected PWM pulse widths. Writes to any of the registers 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 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.

18.4.2.4 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 while the period is determined by the value in TPMxMODH:TPMxMODL.

TPMxMODH:TPMxMODL must be kept in the range of 0x0001 to 0x7FFF because values outside this range can produce ambiguous results. 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), 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, 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).

This is not a significant limitation. 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 in order to change directions from up-counting to down-counting.

The channel-match value in the TPM channel registers (times two) determines the pulse width (duty cycle) of the CPWM signal (Figure 18-15 on page 293). If ELS*n*A is cleared, a channel match occurring while counting up clears the CPWM output signal and a channel match occurring while counting down sets the output. The counter counts up until it reaches the modulo setting in TPMxMODH:TPMxMODL, then counts down until it reaches zero. This sets the period equal to two times TPMxMODH:TPMxMODL.

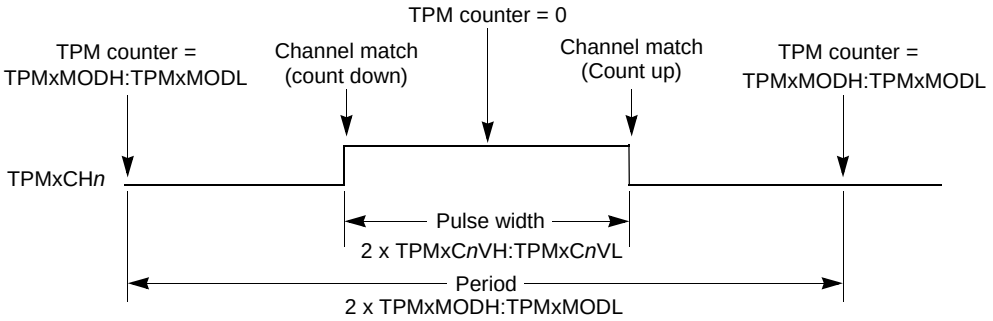


Figure 18-15. CPWM period and pulse width (ELSnA = 0)

Center-aligned PWM outputs typically produce less noise than edge-aligned PWMs because fewer I/O pin transitions are lined up at the same system clock edge. This type of PWM is 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 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.

When TPMxCNTH:TPMxCNTL equals TPMxMODH:TPMxMODL, the TPM can optionally generate a TOF interrupt (at the end of this count).

18.5 Reset overview

18.5.1 General

The TPM is reset whenever any MCU reset occurs.

18.5.2 Description of reset operation

Reset clears TPMxSC that disables TPM counter clock and overflow interrupt (TOIE=0). CPWMS, MS*n*B, MS*n*A, ELS*n*B and ELS*n*A are all cleared. This configures all TPM channels for input capture operation and the associated pins are not controlled by TPM.

18.6 Interrupts

18.6.1 General

The TPM generates an optional interrupt for the main counter overflow and an interrupt for each channel. The meaning of channel interrupts depends on each channel's mode of operation. If the channel is configured for input capture, the interrupt flag is set each time the selected input capture edge is recognized. If the channel is configured for output compare or PWM modes, the interrupt flag is set each time the main timer counter matches the value in the 16-bit, channel-value register.

All TPM interrupts are listed in Table 18-14.

Table 18-14. 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.

18.6.2 Description of interrupt operation

For each interrupt source in the TPM, a flag bit is set upon recognition of the interrupt condition such as timer overflow, channel input-capture or output-compare events. This flag is read (polled) by software to determine that the action has occurred or an associated enable bit (TOIE or 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. Software must perform a sequence of steps to clear the interrupt flag before returning from the interrupt-service routine.

TPM interrupt flags are cleared by a two-step process including a read of the flag bit, while it is set, followed by a write of 0 to the bit. If a new event is detected between these two steps, the sequence is reset and the interrupt flag remains set after the second step to avoid the possibility of missing the new event.

18.6.2.1 Timer overflow interrupt (TOF) description

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 above.

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.

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).

18.6.2.2 Channel event interrupt description

The meaning of channel interrupts depends on the channel's current mode (input capture, output compare, edge-aligned PWM or center-aligned PWM).

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 "Description of interrupt operation" on page 294.

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 "Description of interrupt operation" on page 294.

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 "Description of interrupt operation" on page 294.

Chapter 19 Interrupt Controller

19.1 Introduction

The interrupt controller is intended for use in low-cost micro controller 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, yet 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 (via software convention with hardware assists) equivalent to the S08's interrupt processing with only one level of nesting.
- Leverages the current ColdFire interrupt controller programming model and functionality, but with a minimal hardware implementation and cost.

Table 19-1 provides a high-level architectural comparison between HCS08 and ColdFire exception processing as these differences are important in the definition of the CF1_INTC module. Throughout this document, the term IRQ refers to an interrupt request, and ISR refers to an interrupt service routine to process an interrupt exception.

Table 19-1. Exception processing comparison

Attribute	HCS08	V1 ColdFire
Exception vector table	32 two-byte entries, fixed location at upper end of memory	103 four-byte entries, located at lower end of memory at reset, relocatable with the VBR
More on vectors	Two for CPU + 30 for IRQs, reset at upper address	64 for CPU + 39 for IRQs, reset at lowest address
Exception stack frame	Five-byte frame: CCR, A, X, PC	Eight-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 19-1. Exception processing comparison (continued)

Attribute	HCS08	V1 ColdFire
INTC vectoring	Fixed priorities and vector assignments	Fixed priorities and vector assignments, plus any two IRQs can be remapped as the highest priority Level-6 requests
Software IACK	No	Yes
Exit instruction from ISR	RTI	RTE

19.2 Overview

Interrupt exception processing includes interrupt recognition, aborting the current instruction execution stream, storing of an 8-byte exception stack frame in 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, which cannot be masked.

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

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]. Occurrence of an interrupt exception also forces the master mode (M) bit to be cleared and the interrupt priority mask (I) to be 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 1-Mbyte 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-Mbyte 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, 0x(00)30_0000 in ROM, 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 user-defined interrupt vectors. For the V1 ColdFire core, the table is partially populated with the first 64 reserved for internal processor exceptions, while vectors 64 and above are reserved for the peripheral I/O requests and the seven software interrupts. The IRQ assignments are device-specific as they depend on the exact set of peripherals for any given device.

A simplified V1 ColdFire exception vector table is shown in Table 19-2. **This is a generic table for illustration purposes only. It is NOT necessarily the exception table for this device. See the memory map chapter of the device specification for that detail**.

Table 19-2. Sample V1 ColdFire exception vector table

Vector number(s)	Vector offset (hexadecimal)	Stacked program counter	Assignment
0	0x000	—	Initial supervisor stack pointer
1	0x004	—	Initial program counter
2–63	0x008–0x0FC	—	Reserved for internal CPU exceptions.
64	0x100	Next	IRQ_pin
65	0x104	Next	Low_voltage
66	0x108	Next	TPM1_ch0
67	0x10C	Next	TPM1_ch1
68	0x110	Next	TPM1_ch2
69	0x114	Next	TPM1_ovfl
70	0x118	Next	TPM2_ch0
71	0x11C	Next	TPM2_ch1

Table 19-2. Sample V1 ColdFire exception vector table (continued)

Vector number(s)	Vector offset (hexadecimal)	Stacked program counter	Assignment
72	0x120	Next	TPM2_ch2
73	0x124	Next	TPM2_ovfl
74	0x128	Next	SPI2
75	0x12C	Next	SPI1
76	0x130	Next	SCI1_err
77	0x134	Next	SCI1_rx
78	0x138	Next	SCI1_tx
79	0x13C	Next	IIC
80	0x140	Next	KBIx
81	0x144	Next	Reserved
82	0x148	Next	ACMPx
83	0x14C	Next	SCI2_err
84	0x150	Next	SCI2_rx
85	0x154	Next	SCI2_tx
86	0x158	Next	RTC
87	0x15C	Next	TPM3_ch0
88	0x160	Next	TPM3_ch1
89	0x164	Next	TPM3_ch2
90	0x168	Next	TPM3_ch3
91	0x16C	Next	TPM3_ch4
92	0x170	Next	TPM3_ch5
93	0x174	Next	TPM3_ovfl
94–95	0x178–0x17C	—	Reserved; unused for V1
96	0x180	Next	Level 7 Software Interrupt
97	0x184	Next	Level 6 Software Interrupt
98	0x188	Next	Level 5 Software Interrupt
99	0x18C	Next	Level 4 Software Interrupt
100	0x190	Next	Level 3 Software Interrupt
101	0x194	Next	Level 2 Software Interrupt
102	0x198	Next	Level 1 Software Interrupt

Table 19-2. Sample V1 ColdFire exception vector table (continued)

Vector number(s)	Vector offset (hexadecimal)	Stacked program counter	Assignment
103–255	0x19C–0x3FC	—	Reserved; unused for V1

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 sometimes 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 30 are available for I/O interrupt requests. The requirement for an exact match between the interrupt requests and priorities across two architectures means the 30 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 peripheral bus interface to access its programming model. These interfaces are shown in the simplified block diagram of Figure 19-1.

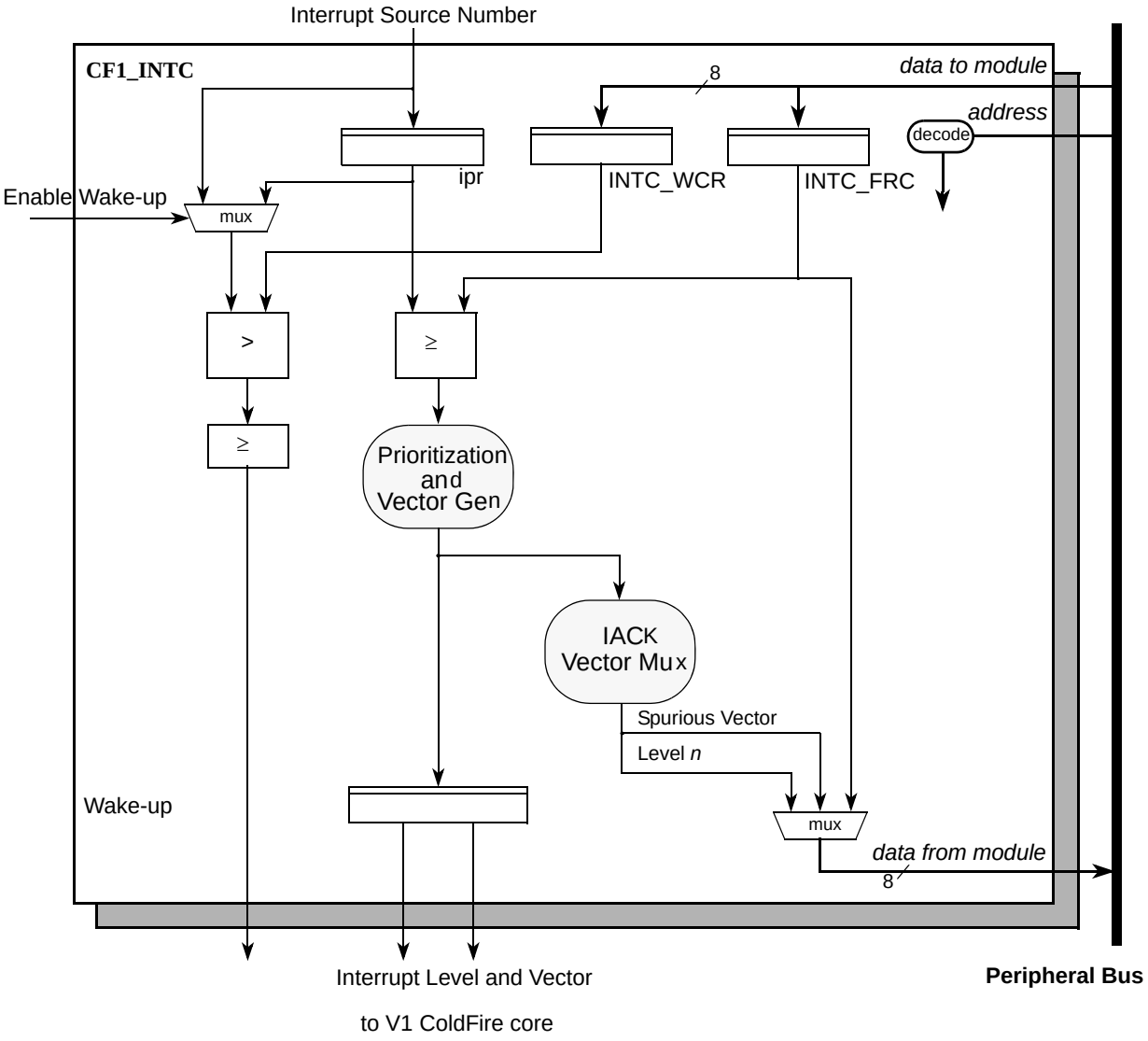


Figure 19-1. CF1_INTC block diagram

19.2.1 Features

The Version 1 ColdFire interrupt controller includes:

- Memory-mapped off-platform slave module
  - 64-byte space located at top end of memory: 0x(FF)FF_FFC0–0x(FF)FF_FFFF
  - Programming model accessed via the peripheral bus
  - Encoded interrupt level and vector sent directly to processor core
- Support of 35 peripheral I/O interrupt requests plus seven software (one per level) interrupt requests
- Fixed association between interrupt request source and level plus priority

  - 35 I/O requests assigned across seven available levels and nine priorities per level
  - Exactly matches HCS08 interrupt request priorities
  - Up to two requests can be remapped to the highest maskable level + priority
- Unique vector number for each interrupt source
  - ColdFire vector number = 62 + HCS08 vector number
  - Details on IRQ and vector assignments are device-specific
- Support for service routine interrupt acknowledge (software IACK) read cycles for improved system performance
- Combinatorial path provides wake-up signal from wait and sleep modes

19.2.2 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 IPS peripheral slave bus, it responds based strictly on the memory addresses of the connected bus.

The wake-up mode of the CF1_INTC deserves mention. When the device enters a wait or stop mode of operation 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 sent to the clock generation logic to re-enable the internal device clocks to exit the low-power mode.

19.2.3 Device-specific exception and interrupt vector tables

For this information, see "Interrupt vector table" on page 57.

19.2.4 External signal description

The CF1_INTC module does not include any external interfaces.

19.3 Interrupt Controller memory map and register definition

The CF1_INTC module provides a 64-byte programming model mapped to the upper region of the 16 Mbyte address space. All the register names are prefixed with INTC_ as an abbreviation for the full module name.

The programming model is referenced using 8-bit accesses. Attempted references to undefined (reserved) addresses or with a non-supported access type (for example, a write to a read-only register) generate a bus error termination.

The programming model follows the definition from previous ColdFire interrupt controllers. This compatibility accounts for the various memory holes in this module's memory map.

19.3.1 Interrupt Controller memory map

Memory space defined by the V1 ColdFire core uses a 24-bit address, providing support for a 16-MByte definition. Table 19-3 shows the resulting system memory map.

The CF1_INTC module is based at address 0x(FF)FF_FFC0 (referred to as CF1_INTC_BASE) and occupies the upper 64 bytes of the peripheral space. The module memory map is shown in Table 19-3

(continued)

Table 19-3. CF1_INTC memory map

Offset address	Register	Width (bits)	Access	Reset	Details
0x10	CF1_INTC Force Interrupt (INTC_FRC)	8	Read/Write	0x00	page 304
0x18	CF1_INTC Programmable Level 6, Priority 7 (INTC_PL6P7)	8	Read/Write	0x00	page 306
0x19	CF1_INTC Programmable Level 6, Priority 6 (INTC_PL6P6)	8	Read/Write	0x00	page 306
0x1B	CF1_INTC Wake-up Control (INTC_WCR)	8	Read/Write	0x00	page 307
0x1E	CF1_INTC Set Interrupt Force (INTC_SFRC)	8	Write	—	page 308
0x1F	CF1_INTC Clear Interrupt Force (INTC_CFRC)	8	Write	—	page 309
0x20	CF1_INTC Software Interrupt Acknowledge (INTC_SWIACK)	8	Read	0x00	page 310
0x24	CF1_INTC Level 1 Interrupt Acknowledge (INTC_LVL1IACK)	8	Read	0x18	page 310
0x28	CF1_INTC Level 2 Interrupt Acknowledge (INTC_LVL2IACK)	8	Read	0x18	page 310
0x2C	CF1_INTC Level 3 Interrupt Acknowledge (INTC_LVL3IACK)	8	Read	0x18	page 310
0x30	CF1_INTC Level 4 Interrupt Acknowledge (INTC_LVL4IACK)	8	Read	0x18	page 310
0x34	CF1_INTC Level 5 Interrupt Acknowledge (INTC_LVL15ACK)	8	Read	0x18	page 310
0x38	CF1_INTC Level 6 Interrupt Acknowledge (INTC_LVL6IACK)	8	Read	0x18	page 310
0x3C	CF1_INTC Level 7 Interrupt Acknowledge (INTC_LVL7IACK)	8	Read	0x18	page 310

19.3.2 Interrupt Controller register descriptions

The following sections detail the individual registers within the CF1_INTC's programming model.

19.3.2.1 INTC Force Interrupt register

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).

	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

= Unimplemented or reserved

Figure 19-2. INTC Force Interrupt register (INTC_FRC)

Table 19-4. INTC_FRC field descriptions

Bit(s)	Field	Description
7	—	Reserved. Must be cleared.
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.
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.

19.3.2.2 INTC Programmable Level 6, Priority {7,6} registers

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 is not changed. 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, which 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” on page 312.

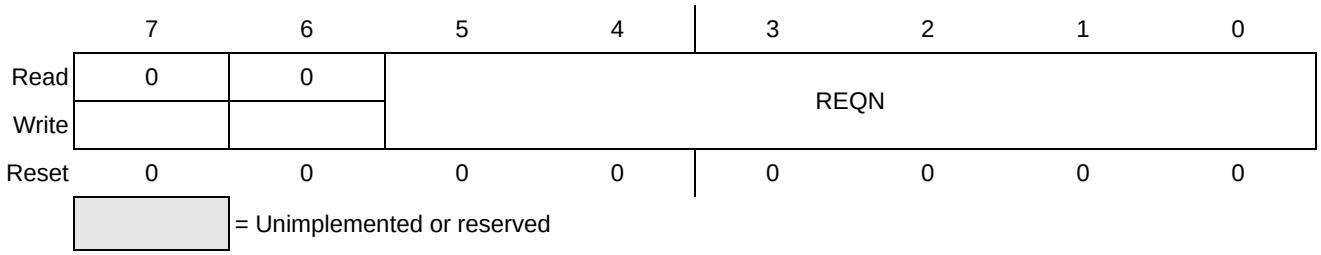


Figure 19-3. INTC Programmable Level, Priority {7, 6} registers (INTC_PL6P{7,6})

Table 19-5. INTC Programmable Level, Priority {7, 6} registers (INTC_PL6P{7,6}) field descriptions

Bit(s)	Field	Description
7–5	—	Reserved. Must be cleared.
4–0	REQN	Request number. Defines the peripheral IRQ number to be remapped as the level 6, priority 7 (for INTC_PL6P7) request (priority 6 for INTC_PL6P6). This is should be the vector number - 64. The value must be in the valid range of interrupts; all other values are ignored.

19.3.2.3 INTC Wake-up Control register

The interrupt controller provides a combinatorial logic path to generate a special wake-up signal to exit from the wait or stop modes. The INTC_WCR register defines wake-up 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 (INTC_WCR[ENB]) and define the interrupt mask level needed to force the core to exit the wait or stop mode (INTC_WCR[MASK]). The maximum value of INTC_WCR[MASK] is 0x6 (0b110).
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 wake-up output signal. This signal is routed to the clock generation logic to exit the low-power mode and resume processing.

Typically, the interrupt mask level loaded into the processor's status register field (SR[I]) during the execution of the STOP instruction matches the INTC_WCR[MASK] value.

The interrupt controller's wake-up signal is defined as:

```
wake-up = INTC_WCR[ENB] & (level of any asserted_int_request > INTC_WCR[MASK])
```

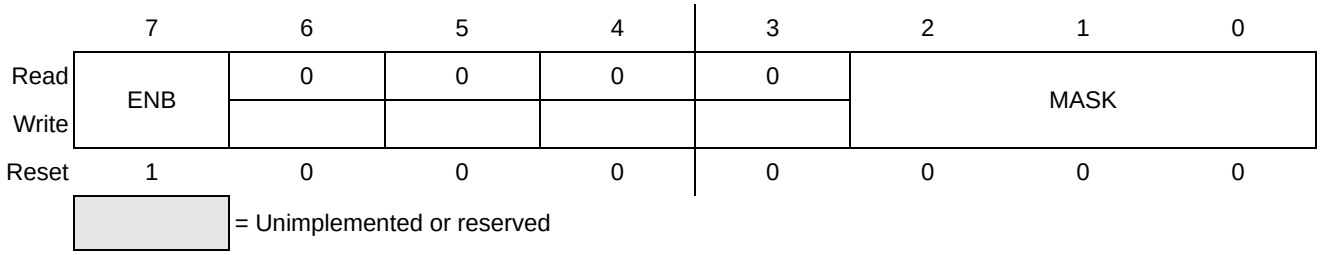


Figure 19-4. INTC Wake-up Control register (INTC_WCR)

Table 19-6. INTC Wake-up Control register (INTC_WCR) field descriptions

Bit(s)	Field	Description
7	ENB	Enable. 0 Wake-up signal not enabled. 1 Wake-up signal enabled.
6–3	—	Reserved, must be cleared.
2–0	MASK	Interrupt mask level. Defines the interrupt mask level during wait or stop mode and is enforced by the hardware to be within the range 0–6. If INTC_WCR[ENB] is set, after an interrupt request of a level higher than MASK is asserted, the wake-up signal to the clock generation logic is asserted.

19.3.2.4 INTC Set Interrupt Force register

The INTC_SFRC register provides a simple memory-mapped mechanism to set a given bit in the INTC_FRC register to assert a specific level interrupt request. The data value written causes the appropriate bit in the INTC_FRC register to be set. Attempted reads of this register generate an error termination.

This register is provided so interrupt service routines can generate a forced interrupt request without the need to perform a read-modify-write sequence on the INTC_FRC register.

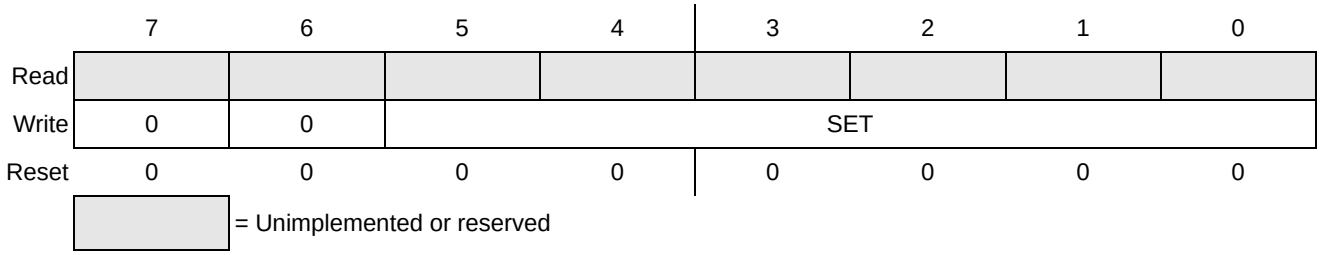


Figure 19-5. INTC Set Interrupt Force register (INTC_SFRC)

Table 19-7. INTC Set Interrupt Force register (INTC_SFRC) field descriptions

Bit(s)	Field	Description
7–6	—	Reserved. Must be cleared.
5–0	SET	For data values within the 56–62 range, the corresponding bit in the INTC_FRC register is set, as defined below. **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. 0x38 INTC_FRC[LVL7] is set. 0x39 INTC_FRC[LVL6] is set. 0x3A INTC_FRC[LVL5] is set. 0x3B INTC_FRC[LVL4] is set. 0x3C INTC_FRC[LVL3] is set. 0x3D INTC_FRC[LVL2] is set. 0x3E INTC_FRC[LVL1] is set.

19.3.2.5 INTC Clear Interrupt Force register

The INTC_CFRC register provides a simple memory-mapped mechanism to clear a given bit in the INTC_FRC register to negate a specific level interrupt request. The data value on the register write causes the appropriate bit in the INTC_FRC register to be cleared. Attempted reads of this register generate an error termination.

This register is provided so interrupt service routines can negate a forced interrupt request without the need to perform a read-modify-write sequence on the INTC_FRC register.

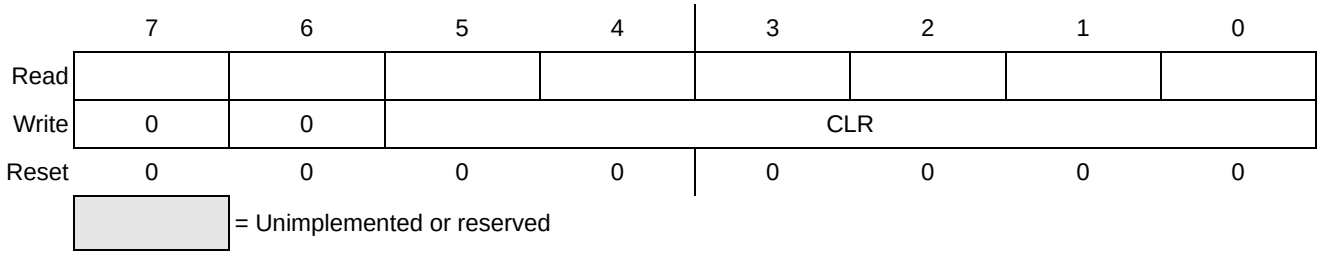


Figure 19-6. INTC Clear Interrupt Force register (INTC_CFRC)

Table 19-8. INTC Clear Interrupt Force register (INTC_CFRC) field descriptions

Bit(s)	Field	Description
7–6		Reserved. Must be cleared.
5–0	CLR	For data values within the 56–62 range, the corresponding bit in the INTC_FRC register is cleared, as defined below. Data values outside the following ranges 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. 0x38 INTC_FRC[LVL7] is cleared. 0x39 INTC_FRC[LVL6] is cleared. 0x3A INTC_FRC[LVL5] is cleared. 0x3B INTC_FRC[LVL4] is cleared. 0x3C INTC_FRC[LVL3] is cleared. 0x3D INTC_FRC[LVL2] is cleared. 0x3E INTC_FRC[LVL1] is cleared.

19.3.2.6 INTC Software and Level-*n* IACK registers (*n* = 1,2,3,...,7)

The eight read-only interrupt acknowledge (IACK) registers can be explicitly addressed via memory-mapped accesses or implicitly addressed via a processor-generated interrupt acknowledge cycle during exception processing when CPUCR[IAE] is set. In either case, the interrupt controller's actions are similar.

First, consider an IACK cycle to a specific level, a level-*n* IACK. When this type of IACK arrives in the interrupt controller, the controller examines all currently-active level-*n* interrupt requests, determines the highest priority within the level, and then responds with the unique vector number corresponding to that specific interrupt source. The vector number is supplied as the data for the byte-sized IACK read cycle.

If there is no active interrupt source at the time of the level-*n* IACK, a special spurious interrupt vector (vector number 24 (0x18)) is returned. It is the responsibility of the service routine to manage this error situation.

This protocol implies the interrupting peripheral is not accessed during the acknowledge cycle because the interrupt controller completely services the acknowledge. This means the interrupt source must be explicitly disabled in the peripheral device by the interrupt service routine. This approach provides unique vector capability for all interrupt requests, regardless of the complexity of the peripheral device.

Second, the interrupt controller also supports the concept of a software IACK. This is the ability to query the interrupt controller near the end of an interrupt service routine (after the current interrupt request has been negated) to determine if there are any pending (but currently masked) interrupt requests. If the response to the software IACK's byte operand read is non-zero, the service routine uses the returned value as the vector number of the highest pending interrupt request and passes control to the appropriate new handler. If the returned value is zero, there is no pending interrupt request.

This process avoids the overhead of a context restore and RTE instruction execution, followed immediately by another interrupt exception and context save. In system environments with high rates of interrupt activity, this mechanism can noticeably improve overall performance. For additional details on software IACKs, see “More on software IACKs” on page 313.

	7	6	5	4	3	2	1	0
Read	0	VECN						
Write								
SWIACK reset	0	0	0	0	0	0	0	0
LVL*n*IACK reset	0	0	0	1	1	0	0	0

= Unimplemented or reserved

Figure 19-7. INTC Software and Level-*n* IACK registers (INTC_SWIACK, INTC_LVLnIACK)

Table 19-9. INTC Software and Level-*n* IACK registers (INTC_SWIACK, INTC_LVLnIACK) field descriptions

Bit(s)	Field	Description
7	—	Reserved. Must be cleared.
6–0	VECN	Vector number. Indicates the appropriate vector number. • For the SWIACK register, it is the highest-level, highest-priority request currently being asserted in the CF1_INTC module. If there are no pending requests, VECN is zero. • For the LVL*n*IACK register, 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.

19.4 Functional description

The basic operation of the CF1_INTC has been detailed in the preceding sections. This section describes special rules applicable to non-maskable level seven interrupt requests and the module's interfaces.

19.4.1 Handling of non-maskable, level-7 interrupt requests

Level seven interrupts are treated as non-maskable, edge-sensitive requests while levels one through six are maskable, level-sensitive requests. As a result of this definition, level seven interrupt requests are a special case. The edge-sensitive nature of these requests means the encoded 3-bit level input from the CF1_INTC to the V1 ColdFire core must change state before the CPU detects an interrupt. A non-maskable interrupt (NMI) is generated each time the encoded interrupt level changes to level seven (regardless of the SR[I] field) and each time the SR[I] mask changes from seven to a lower value while the encoded request level remains at seven.

19.5 Initialization information

The reset state of the CF1_INTC module enables the default IRQ mappings and clears any software-forced interrupt requests (INTC_FRC is cleared). The wake-up control register (INTC_WCR) is also disabled, so it must be written before the processor executes any STOP instructions to properly exit from any wait or stop mode. 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 processor core.

19.6 Application information

This section discusses three application topics: emulation of the HCS08's one level interrupt nesting structure, elevating the priority of two IRQs, and more details on the operation of the software interrupt acknowledge (SWIACK) mechanism.

19.6.1 Emulation of the HCS08's one-level, IRQ handling

As noted in Table 19-1, the HCS08 architecture specifies a 1-level IRQ nesting capability. Interrupt masking is controlled by CCR[I], the interrupt mask flag: clearing CCR[I] enables interrupts, while setting CCR[I] disables interrupts. The ColdFire architecture defines seven interrupt levels, controlled by the 3-bit interrupt priority mask field in the status register, SR[I], and the hardware automatically supports nesting of interrupts.

To emulate the HCS08's 1-level IRQ capabilities on V1 ColdFire, only two SR[I] settings are used:

- Writing 0 to SR[I] enables interrupts.
- Writing 7 to SR[I] disables interrupts.

ColdFire 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 setting SR[I] = 0 (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], which 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 slightly improves system performance.

19.6.2 Using INTC_PL6P{7,6} registers

"INTC Programmable Level 6, Priority {7,6} registers" on page 306 describes control registers that provide the ability to dynamically alter the request level and priority of two IRQs. Specifically, these registers provide the ability to reassign two IRQs to be the highest level 6 (maskable) requests. Consider the following example.

Suppose the system operation desires to remap the slave port wake-up interrupt as the highest maskable interrupt. The default assignment for the slave port wake-up is:

slave port wake-up = vector 82 at level 4, priority 5

To remap this requests, the INTC_PL6P7 register is programmed with 0x12.

The reset state of the INTC_PL6P{7,6} registers disables any request remapping.

19.6.3 More on software IACKs

As previously mentioned, the notion of a software IACK refers to the ability to query the interrupt controller near the end of an interrupt service routine (after the current interrupt request has been cleared) to determine if there are any pending (but currently masked) interrupt requests. If the response to the software IACK's byte operand read is non-zero, the service routine uses the value as the vector number of the highest pending interrupt request and passes control to the appropriate new handler. This process avoids the overhead of a context restore and RTE instruction execution, followed immediately by another interrupt exception and context save. In system environments with high rates of interrupt activity, this mechanism can improve overall system performance noticeably.

To illustrate this concept, consider the following ISR code snippet shown in Example 19-1.

Example 19-1. ISR code snippet with SWIACK

```
     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
```

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 these registers have been saved, 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 has been 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, ISR is interrupted to service one of these requests. To avoid any race conditions, this check ignores the two 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 20 ColdFire Core

20.1 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*.

20.2 Overview

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

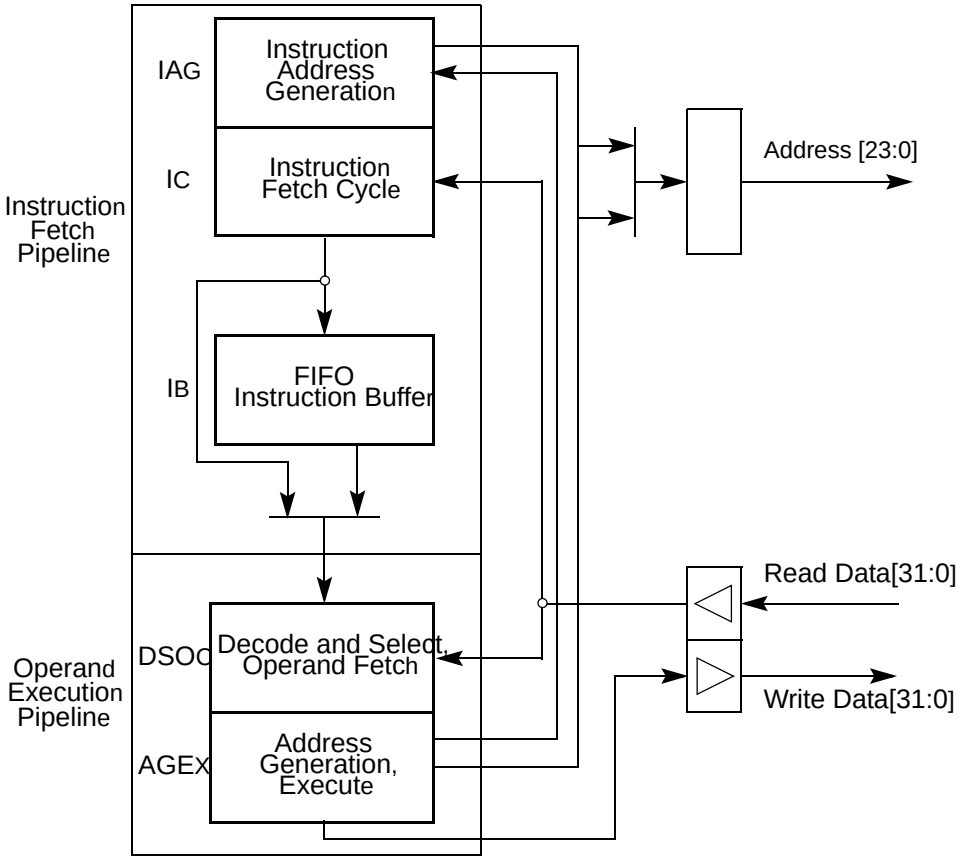


Figure 20-1. V1 ColdFire core pipelines

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), which 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 long words 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.

20.3 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 processor's registers are listed in Table 20-1.)

NOTE

The trace buffer and supervisor mode are not enabled for the MMA955xL platform.

Table 20-1 lists the processor registers.

The user-programming model consists of the following registers:

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

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

- 16-bit Status Register (SR)
- 32-bit Supervisor Stack Pointer (SSP)
- 32-bit Vector Base Register (VBR)
- 32-bit CPU Configuration Register (CPUCR)

3

Table 20-1. ColdFire core programming model

BDM command[1]	Register	Width (bits)	Access	Reset value	Written with MOVEC[2]	Section/Page
Supervisor/User Access Registers						
Load: 0x60 Store: 0x40	Data Register 0 (D0)	32	R/W	0xCF10_029	No	page 319
Load: 0x61 Store: 0x41	Data Register 1 (D1)	32	R/W		No	page 319
Load: 0x6–7 Store: 0x4–7	Data Register –7 (D–D7)	32	R/W	POR: Undefined Else: Unaffected	No	page 319
Load: 0x68–E Store: 0x48–E	Address Register 0–6 (A0–A6)	32	R/W	POR: Undefined Else: Unaffected	No	page 319
Load: 0x6F Store: 0x4F	User A7 Stack Pointer (A7)	32	R/W	POR: Undefined Else: Unaffected	No	page 320

Table 20-1. ColdFire core programming model (continued)

BDM command[1]	Register	Width (bits)	Access	Reset value	Written with MOVEC[2]	Section/Page
Load: 0xEE Store: 0xCE	Condition Code Register (CCR)	8	R/W	POR: Undefined Else: Unaffected	No	page 321
Load: 0xEF Store: 0xCF	Program Counter (PC)	32	R/W	Contents of location 0x(00)00_0004	No	page 321
Supervisor Access Only Registers						
Load: 0xE0 Store: 0xC0	Supervisor A7 Stack Pointer (OTHER_A7)	32	R/W	Contents of location 0x(00)00_0000	No	page 320
Load: 0xE1 Store: 0xC1	Vector Base Register (VBR)	32	R/W	0x0000_0000	Yes; Rc = 0x801	page 322
Load: 0xE2 Store: 0xC2	CPU Configuration Register (CPUCR)	32	W	0x0000_0000	Yes; Rc = 0x802	page 322
Load: 0xEE Store: 0xCE	Status Register (SR)	16	R/W	0x27--	No	page 324

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 more information see Figure 21 on page 347. (These BDM commands are not similar to other ColdFire processors.)

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

20.3.1 Data registers

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

NOTE

Registers D0 and D1 contain hardware configuration details after reset. See "Reset exception" on page 335 for more details.

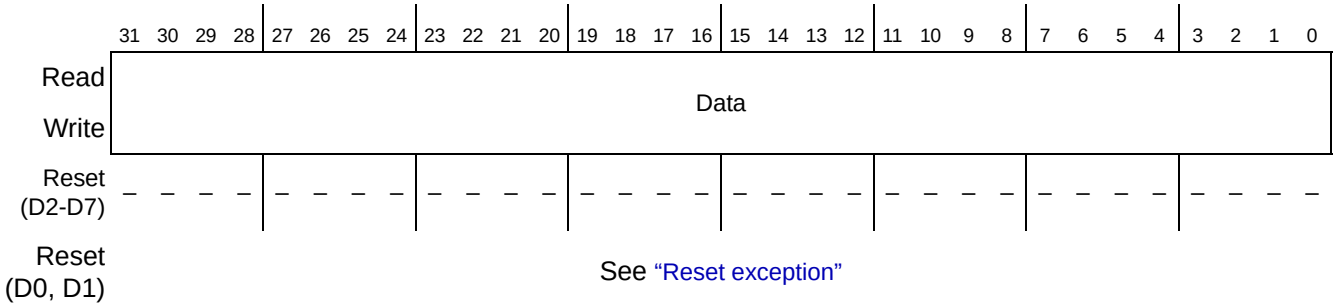


Figure 20-2. Data registers (D0–D7)

20.3.2 Address Registers

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

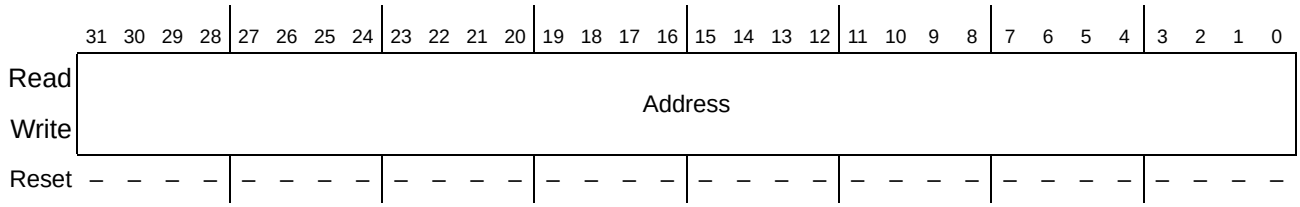


Figure 20-3. Address registers (A0–A6)

20.3.3 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 example.

Example 20-1. Supervisor/user stack pointers

```
if SR[S] = 1
   then        A7 = Supervisor Stack Pointer
     OTHER_A7 = User Stack Pointer
   else      A 7 = 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:

Example 20-2. Supervisor instructions

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

These instructions are described in the *ColdFire Family Programmer's Reference Manual*. All other instruction references to the stack pointer, explicit or implicit, access the active A7 register.

NOTE

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

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

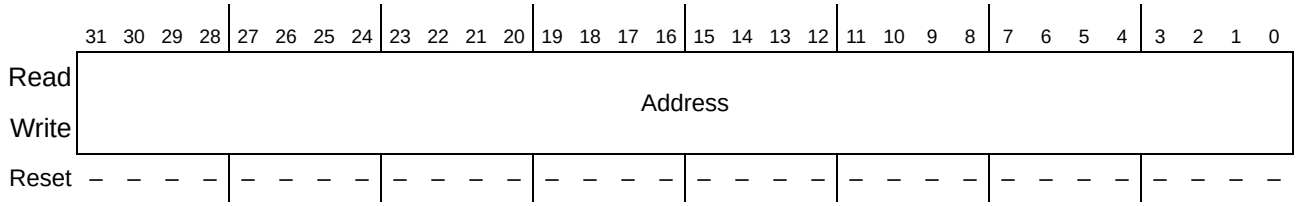


Figure 20-4. Stack Pointer Registers (A7 and OTHER_A7)

20.3.3.1 Condition Code Register

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 multi precision arithmetic computations. The CCR register must be explicitly loaded after reset and before any compare (CMP), Bcc, or Scc instructions are executed.

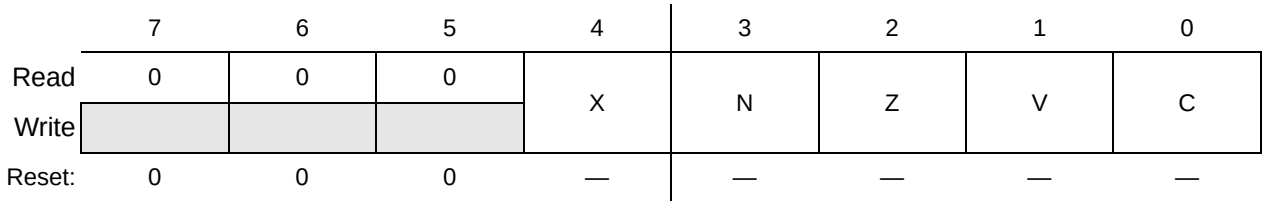


Figure 20-5. Condition Code Register (CCR)

Table 20-2. Condition Code Register (CCR) field descriptions

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

20.3.4 Program Counter (PC)

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

The PC is initially loaded during reset exception processing with the contents of location 0x(00)00_0004.

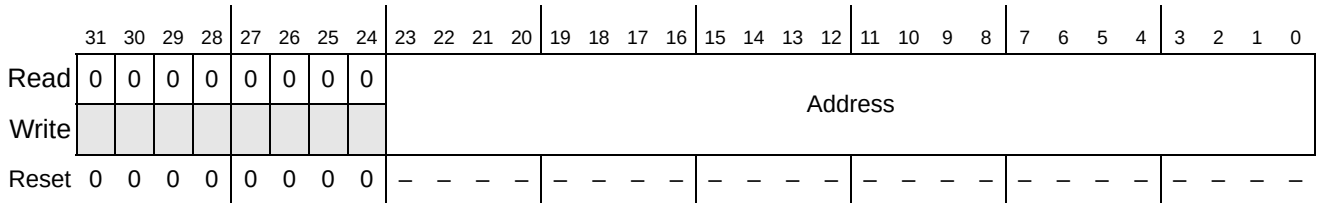


Figure 20-6. Program Counter Register (PC)

20.3.5 Vector Base Register

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

In addition, because the V1 ColdFire core supports a 16-Mbyte 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.

	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
Read	0	0	0	0	0	0	0	0	Base Address				0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Write																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 20-7. Vector Base Register (VBR)

20.3.6 CPU Configuration Register

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

	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
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 20-8. CPU Configuration Register (CPUCR)

Table 20-3. CPUCR field descriptions

Bit(s)	Field	Description
31	ARD	Address-related reset disable. Used to disable the generation of a reset event in response to a processor exception caused by an address error, a bus error, an RTE format error, or a fault-on-fault halt condition. 0 The detection of these types of exception conditions or the fault-on-fault halt condition generate a reset event. 1 No reset is generated in response to these exception conditions.
30	IRD	Instruction-related reset disable. Used to disable the generation of a reset event in response to a processor exception caused by the attempted execution of an illegal instruction (except for the ILLEGAL opcode), illegal line A, illegal line F instructions, or a privilege violation. 0 The detection of these types of exception conditions generate a reset event. 1 No reset is generated in response to these exception conditions.
29	IAE	Interrupt acknowledge (IACK) enable. Forces the processor to generate an IACK read cycle from the interrupt controller during exception processing to retrieve the vector number of the interrupt request being acknowledged. The processor's execution time for an interrupt exception is slightly improved when this bit is cleared. 0 The processor uses the vector number provided by the interrupt controller at the time the request is signaled. 1 IACK read cycle from the interrupt controller is generated.
28	IME	Interrupt mask enable. Forces the processor to raise the interrupt level mask (SR[I]) to 7 during every interrupt exception. 0 As part of an interrupt exception, the processor sets SR[I] to the level of the interrupt being serviced. 1 As part of an interrupt exception, the processor sets SR[I] to 7. This disables all level 1-6 interrupt requests but allows recognition of the edge-sensitive level 7 requests.
27	BWD	If buffered writes are enabled (BWD = 0), any error status is lost as the immediate termination of the data transfer assumes an error-free completion. **Note:** Buffered write disable. The ColdFire core is capable of marking processor memory writes as bufferable or non-bufferable. 0 Writes are buffered and the bus cycle is terminated immediately with zero wait states. 1 Disable the buffering of writes. In this configuration, the write transfer is terminated based on the response time of the addressed destination memory device.
26	—	Reserved, must be cleared.
25	FSD	Flash speculation disabled. Disables certain performance-enhancing features related to address speculation in the flash memory controller. 0 The flash controller tries to speculate on read accesses to improve processor performance by minimizing the exposed flash memory access time. Recall the basic flash access time is two processor cycles. 1 Certain flash address speculation is disabled.
24–0	—	Reserved, must be cleared.

20.3.7 Status Register (SR)

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

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

Figure 20-9. Status Register (SR)

Table 20-4. SR field descriptions

Bit(s)	Field	Description
15	T	Trace enable. When set, the processor performs a trace exception after every instruction.
14	—4	Reserved, must be cleared.
13	S	Supervisor/user state 0 User mode 1 Supervisor mode
12	M	Master/interrupt state. Bit is cleared by an interrupt exception and software can set it during execution of the RTE or move to SR instructions.
11	—	Reserved, must be cleared.
10–8	I	Interrupt level mask. Defines current interrupt level. Interrupt requests are inhibited for all priority levels less than or equal to current level, except edge-sensitive level 7 requests, which cannot be masked.
7–0	CCR	Refer to “Condition Code Register” on page 321.

20.4 Functional description

20.4.1 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 the following areas:

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

Table 20-5 summarizes the instructions added to revision ISA_A to form revision ISA_C. For more details see the *ColdFire Family Programmer's Reference Manual*.

Table 20-5. 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.
INTOUCH	Loads blocks of instructions to be locked in the instruction cache.
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.

20.4.2 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 20-10 on page 329, 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 Mbyte boundary. This instruction address is generated by fetching an exception vector from the table located at the address defined in the vector base register. The index into the exception table is calculated as (4 × 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 Mbyte address boundary (see Table 20-6). For the V1 ColdFire core, the only practical locations for the vector table are based at 0x(00)00_0000 in the flash or 0x(00)80_0000 in the RAM.

The table contains 256 exception vectors; the first 64 are defined for the core and the remaining 192 are device-specific peripheral interrupt vectors. "Interrupt Controller" on page 297 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 20-6. Exception vector assignments

Vector number(s)	Vector offset (hexadecimal)	Stacked program counter	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	0x01–0x01C	—	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, unused for V1

[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*.

20.4.2.1 Exception stack frame definition

Figure 20-10 on page 329 shows exception stack frame. The first long-word contains the 16-bit format/vector word (F/V) and the 16-bit status register, and the second long-word contains the 32-bit program counter address.

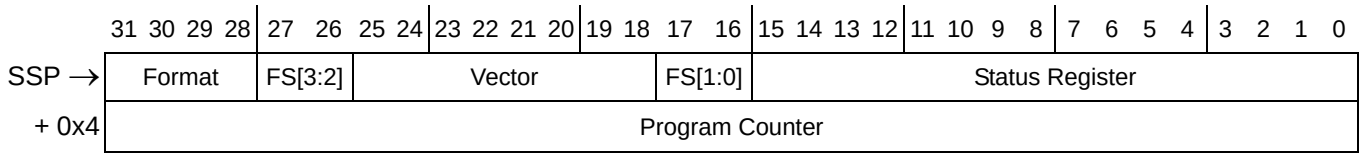


Figure 20-10. Exception Stack Frame Form

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

- A four-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.

Table 20-7. Format field encodings

Original SSP @ time of exception, Bits 1:0	SSP at first 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.

Table 20-8. Fault status encodings

FS[3:0]	Definition
00*xx*	Reserved
0100	Error on instruction fetch
0101	Reserved

Table 20-8. Fault status encodings (continued)

FS[3:0]	Definition
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 20-6.

20.4.3 Processor exceptions

20.4.3.1 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 op word and/or extension words, the access error is signaled and the instruction aborted. For this type of exception, the programming model has not been altered by the instruction generating the access error.

If the access error occurs on an operand read, the processor immediately aborts the current instruction's execution and initiates exception processing. In this situation, any address register updates attributable to the auto-addressing modes, (for example, (An)+,-(An)), have already been performed, so the programming model contains the updated An value. In addition, if an access error occurs during 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.

20.4.3.2 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.

20.4.3.3 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 op word), while the optional words are known as extension word 1 and extension word 2. The op word 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 (op mode), and the low-order 6 bits define the effective address. See Figure 20-11. The op word line definition is shown in Table 20-9.

Figure 20-11. ColdFire instruction operation word (opword) format

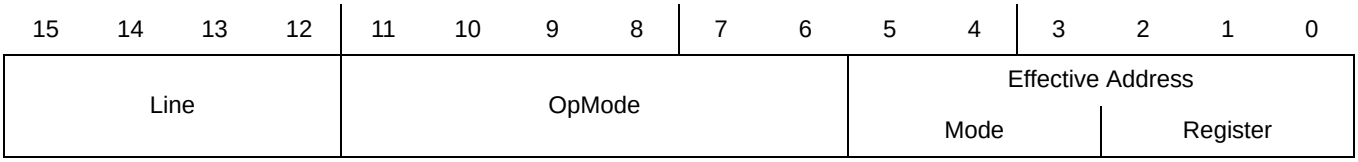


Table 20-9. ColdFire op word line definition

Opword[Line]	Instruction class
0x0	Bit manipulation, Arithmetic and Logical Immediate
0x1	Move Byte
0x2	Move Long

Table 20-9. ColdFire op word line definition (continued)

Opword[Line]	Instruction class
0x3	Move Word
0x4	Miscellaneous
0x5	Add (ADDQ) and Subtract Quick (SUBQ), Set according to Condition Codes (Scc)
0x6	PC-relative change-of-flow instructions Conditional (Bcc) and unconditional (BRA) branches, subroutine calls (BSR)
0x7	Move Quick (MOVEQ), Move with sign extension (MVS) and zero fill (MVZ)
0x8	Logical OR (OR)
0x9	Subtract (SUB), Subtract Extended (SUBX)
0xA	Move 3-bit Quick (MOV3Q)
0xB	Compare (CMP), Exclusive-OR (EOR)
0xC	Logical AND (AND), Multiply Word (MUL)
0xD	Add (ADD), Add Extended (ADDX)
0xE	Arithmetic and logical shifts (ASL, ASR, LSL, LSR)
0xF	Write DDATA (WDDATA), Write Debug (WDEBUG)

In the original M68000 ISA definition, lines A and F were effectively reserved for user-defined operations (line A) and co-processor instructions (line F). Accordingly, there are two unique exception vectors associated with illegal opwords in these two lines.

Any attempted execution of an illegal 16-bit opcode (except for line-A and line-F opcodes) generates an illegal instruction exception (vector 4). Additionally, any attempted execution of any line-A and most line-F opcodes generate their unique exception types, vector numbers 10 and 11, respectively. ColdFire cores do not provide illegal instruction detection on the extension words on any instruction, including MOVEC.

The V1 ColdFire processor also detects two special cases involving illegal instruction conditions:

1. If execution of the stop instruction is attempted and neither low-power stop nor wait modes are enabled, the processor signals an illegal instruction.
2. If execution of the halt instruction is attempted and BDM is not enabled (XCSR[ENBDM] equals 0), the processor signals an illegal instruction.

In both cases, the processor response is then dependent on the state of CPUCR[IRD]— a reset event or a processor exception.

20.4.3.4 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.

20.4.3.5 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.

20.4.3.6 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.

20.4.3.7 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.

20.4.3.8 Debug interrupt

See Figure 21 on page 347, 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.

20.4.3.9 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.

20.4.3.10 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. These instructions and their functionality should not be confused 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.

20.4.3.11 Unsupported-instruction exception

If execution of a valid instruction is attempted but the required hardware is not present in the processor, an unsupported instruction exception is generated. The instruction functionality can then be emulated in the exception handler, if desired.

All ColdFire cores record the processor hardware configuration in the D0 register immediately after the negation of $\overline{\text{RESET}}$. See “Reset exception”, for details.

20.4.3.12 Interrupt exception

Interrupt exception processing includes interrupt recognition and the fetch of the appropriate vector from the interrupt controller using an IACK cycle. See “Interrupt Controller” on page 297, for details on the interrupt controller.

20.4.3.13 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 force the processor to exit this halted state.

20.4.3.14 Reset exception

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

The reset exception places the processor in the supervisor mode by setting the SR[S] bit and disables tracing by clearing the SR[T] bit. This exception also clears the SR[M] bit and sets the processor’s SR[I] field to the highest level (level 7, 0b111). Next, the VBR is initialized to zero (0x0000_0000). The control registers specifying the operation of any memories (e.g., cache and/or RAM modules) connected directly to the processor are disabled.

NOTE

Other implementation-specific registers are also affected. Refer to each module in this reference manual for details on these registers.

After the processor is granted the bus, it performs two longword read-bus cycles. The first longword at address 0x(00)00_0000 is loaded into the supervisor stack pointer and the second longword at address 0x(00)00_0004 is loaded into the program counter. After the initial instruction is fetched from memory, program execution begins at the address in the PC. If an access error or address error occurs before the first instruction is executed, the processor enters the fault-on-fault state.

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

Information loaded into D0 defines the processor hardware configuration as shown in Figure 20-12.

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
Read	PF								VER				REV			
Write																
Reset	1	1	0	0	1	1	1	1	0	0	0	1	0	0		0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	MAC	DIV	EMAC	FPU			0	0	ISA				DEBUG			
Write																
Reset				0			0	0	0	0	1	0	1	0	0	1

Figure 20-12. D0 Hardware configuration information

Table 20-10. D0 Hardware configuration information field description

Bit(s)	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 Else Reserved for future use
19-16	REV	Processor revision number. The default is 0b000.
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. (This is the value used for this device.) 1 Divide execute engine is present in core.
13	EMAC	EMAC present. This bit signals if the optional enhanced multiply-accumulate (EMAC) execution engine is present in processor core. 0 EMAC execute engine not present in core. (This is the value used for this device.) 1 EMAC execute engine is present in core.
12	FPU	FPU present. This bit signals if the optional floating-point (FPU) execution engine is present in processor core. 0 FPU execute engine not present in core. (This is the value used for this device.) 1 FPU execute engine is present in core.

Table 20-10. D0 Hardware configuration information field description (continued)

Bit(s)	Field	Description
11	—	Reserved.
10-8	—	Reserved.
7-4	ISA	ISA revision. Defines the instruction-set architecture (ISA) revision level implemented in ColdFire processor core. 0010 ISA_C Else Reserved
3-0	DEBUG	Debug module revision number. Defines revision level of the debug module used in the ColdFire processor core. 1001 DEBUG_B+ Else Reserved

Information loaded into D1 defines the local memory hardware configuration as shown in the figure below.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Read	0	0	0	1	0	0	0	0	FLASHSZ					0	0	0
Write																

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

Figure 20-13. D1 hardware configuration information

Table 20-11. D1 hardware configuration information field description

Bit(s)	Field	Description
31-24	—	Reserved.
23-19	FLASHSZ	Flash bank size. 0000-0111 No flash 1000 64-Kbyte flash 1001 128-Kbyte flash 1010 256-Kbyte flash 1011 512-Kbyte flash Else Reserved for future use.
18-16	—	Reserved

Table 20-11. D1 hardware configuration information field description (continued)

Bit(s)	Field	Description
15-8	—	Reserved**, resets to 0b010000**
7-3	SRAMSZ	SRAM bank size. 00000 No SRAM 00010 512 bytes 00100 1 KB 00110 2 KB 01000 4 KB 01010 8 KB 01100 16 KB 01110 32 KB 10010 128 KB Else Reserved for future use
2-0	—	Reserved.

20.4.4 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.

20.4.4.1 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 20-12.

Table 20-12. Misaligned operand references

address[1:0]	Size	Bus operations	Additional C(R/W)
01 or 11	Word	Byte, Byte	2(1/0) if read 1(0/1) if write
01 or 11	Long	Byte, Word, Byte	3(2/0) if read 2(0/2) if write
10	Long	Word, Word	2(1/0) if read 1(0/1) if write

20.4.4.2 MOVE instruction execution times

Table 20-14 lists execution times for MOVE.{B,W} instructions; Table 20-15 lists timings for MOVE.L.

NOTE

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

Table 20-13.

ET with {<ea> = (d16,PC)}	equals ET with {<ea> = (d16,An)}
ET with {<ea> = (d8,PC,Xi*SF)}	equals ET with {<ea> = (d8,An,Xi*SF)}

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

Table 20-14. MOVE byte and word execution times

Source	Destination						
	Rx	(Ax)	(Ax)+	-(Ax)	(d16,Ax)	(d8,Ax,Xi*SF)	xxx.wl
Dy	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)
Ay	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)
(Ay)	2(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1))	3(1/1)
(Ay)+	2(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1))	3(1/1)
-(Ay)	2(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1))	3(1/1)

Table 20-14. MOVE byte and word execution times (continued)

Source	Destination						
	Rx	(Ax)	(Ax)+	-(Ax)	(d16,Ax)	(d8,Ax,Xi*SF)	xxx.wl
(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 20-15. MOVE long-execution times

Source	Destination						
	Rx	(Ax)	(Ax)+	-(Ax)	(d16,Ax)	(d8,Ax,Xi*SF)	xxx.wl
Dy	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)
Ay	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)
(Ay)	2(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	3(1/1)	2(1/1)
(Ay)+	2(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	3(1/1)	2(1/1)
-(Ay)	2(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	3(1/1)	2(1/1)
(d16,Ay)	2(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	—	—
(d8,Ay,Xi*SF)	3(1/0)	3(1/1)	3(1/1)	3(1/1)	—	—	—
xxx.w	2(1/0)	2(1/1)	2(1/1)	2(1/1)	—	—	—
xxx.l	2(1/0)	2(1/1)	2(1/1)	2(1/1)	—	—	—
(d16,PC)	2(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	—	—
(d8,PC,Xi*SF)	3(1/0)	3(1/1)	3(1/1)	3(1/1)	—	—	—
#xxx	1(0/0)	2(0/1)	2(0/1)	2(0/1)	—	—	—

20.4.4.3 Standard One operand instruction execution times

Table 20-16. One operand instruction execution times

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An)	(d8,An,Xn*SF)	xxx.wl	#xxx
BITREV	Dx	1(0/0)	—	—	—	—	—	—	—
BYTEREV	Dx	1(0/0)	—	—	—	—	—	—	—
CLR.B	<ea>	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)	—
CLR.W	<ea>	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)	—
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)

20.4.4.4 Standard Two operand instruction execution times

Table 20-17. Two operand instruction execution times

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An) (d16,PC)	(d8,An,Xn*SF) (d8,PC,Xn*SF)	xxx.wl	#xxx
ADD.L	<ea>,Rx	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
ADD.L	Dy,<ea>	—	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
ADDI.L	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
ADDQ.L	#imm,<ea>	1(0/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
ADDX.L	Dy,Dx	1(0/0)	—	—	—	—	—	—	—
AND.L	<ea>,Rx	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
AND.L	Dy,<ea>	—	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
ANDI.L	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
ASL.L	<ea>,Dx	1(0/0)	—	—	—	—	—	—	1(0/0)
ASR.L	<ea>,Dx	1(0/0)	—	—	—	—	—	—	1(0/0)
BCHG	Dy,<ea>	2(0/0)	4(1/1)	4(1/1)	4(1/1)	4(1/1)	5(1/1)	4(1/1)	—
BCHG	#imm,<ea>	2(0/0)	4(1/1)	4(1/1)	4(1/1)	4(1/1)	—	—	—
BCLR	Dy,<ea>	2(0/0)	4(1/1)	4(1/1)	4(1/1)	4(1/1)	5(1/1)	4(1/1)	—
BCLR	#imm,<ea>	2(0/0)	4(1/1)	4(1/1)	4(1/1)	4(1/1)	—	—	—
BSET	Dy,<ea>	2(0/0)	4(1/1)	4(1/1)	4(1/1)	4(1/1)	5(1/1)	4(1/1)	—
BSET	#imm,<ea>	2(0/0)	4(1/1)	4(1/1)	4(1/1)	4(1/1)	—	—	—
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)

Table 20-17. Two operand instruction execution times (continued)

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

20.4.4.5 Miscellaneous instruction execution times

Table 20-18. Miscellaneous instruction execution times

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An)	(d8,An,Xn*SF)	xxx.wl	#xxx
LINK.W	Ay,#imm	2(0/1)	—	—	—	—	—	—	—
MOV3Q.L	#imm,<ea>	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)	—
MOVE.L	Ay,USP	3(0/0)	—	—	—	—	—	—	—
MOVE.L	USP,Ax	3(0/0)	—	—	—	—	—	—	—
MOVE.W	CCR,Dx	1(0/0)	—	—	—	—	—	—	—
MOVE.W	<ea>,CCR	1(0/0)	—	—	—	—	—	—	1(0/0)
MOVE.W	SR,Dx	1(0/0)	—	—	—	—	—	—	—
MOVE.W	<ea>,SR	7(0/0)	—	—	—	—	—	—	7(0/0) [2]
MOVEC	Ry,Rc	9(0/1)	—	—	—	—	—	—	—
MOVEM.L	<ea>,and list	—	1+n(n/0)	—	—	1+n(n/0)	—	—	—
MOVEM.L	and list,<ea>	—	1+n(0/n)	—	—	1+n(0/n)	—	—	—
MVS	<ea>,Dx	1(0/0)	2(1/0)	2(1/0)	2(1/0)	2(1/0)	3(1/0)	2(1/0)	1(0/0)
MVZ	<ea>,Dx	1(0/0)	2(1/0)	2(1/0)	2(1/0)	2(1/0)	3(1/0)	2(1/0)	1(0/0)

Table 20-18. Miscellaneous instruction execution times (continued)

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An)	(d8,An,Xn*SF)	xxx.wl	#xxx
NOP		3(0/0)	—	—	—	—	—	—	—
PEA	<ea>	—	2(0/1)	—	—	2(0/1)[4]	3(0/1)[5]	2(0/1)	—
PULSE		1(0/0)	—	—	—	—	—	—	—
STLDSR	#imm	—	—	—	—	—	—	—	5(0/1)
STOP	#imm	—	—	—	—	—	—	—	3(0/0)[3]
TRAP	#imm	—	—	—	—	—	—	—	12(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]The n is the number of registers moved by the MOVEM opcode.
[2]If a MOVE.W #imm,SR instruction is executed and imm[13] equals 1, the execution time is 1(0/0).
[3]The execution time for STOP is the time required until the processor begins sampling continuously for interrupts.
[4]PEA execution times are the same for (d16,PC).
[5]PEA execution times are the same for (d8,PC,Xn*SF).

20.4.4.6 Branch-instruction execution times

Table 20-19. General branch-instruction execution times

Opcode	<EA>	Effective address							
		Rn	(An)	(An)+	-(An)	(d16,An) (d16,PC)	(d8,An,Xi*SF) (d8,PC,Xi*SF)	xxx.wl	#xxx
BRA		—	—	—	—	2(0/1)	—	—	—
BSR		—	—	—	—	3(0/1)	—	—	—
JMP	<ea>	—	3(0/0)	—	—	3(0/0)	4(0/0)	3(0/0)	—
JSR	<ea>	—	3(0/1)	—	—	3(0/1)	4(0/1)	3(0/1)	—
RTE		—	—	7(2/0)	—	—	—	—	—
RTS		—	—	5(1/0)	—	—	—	—	—

Table 20-20. Bcc instruction execution times

Opcode	Forward Taken	Forward Not Taken	Backward Taken	Backward Not Taken
Bcc	3(0/0)	1(0/0)	2(0/0)	3(0/0)

Chapter 21 Version 1 ColdFire Debug

21.1 Chip-specific information about CF1_DEBUG

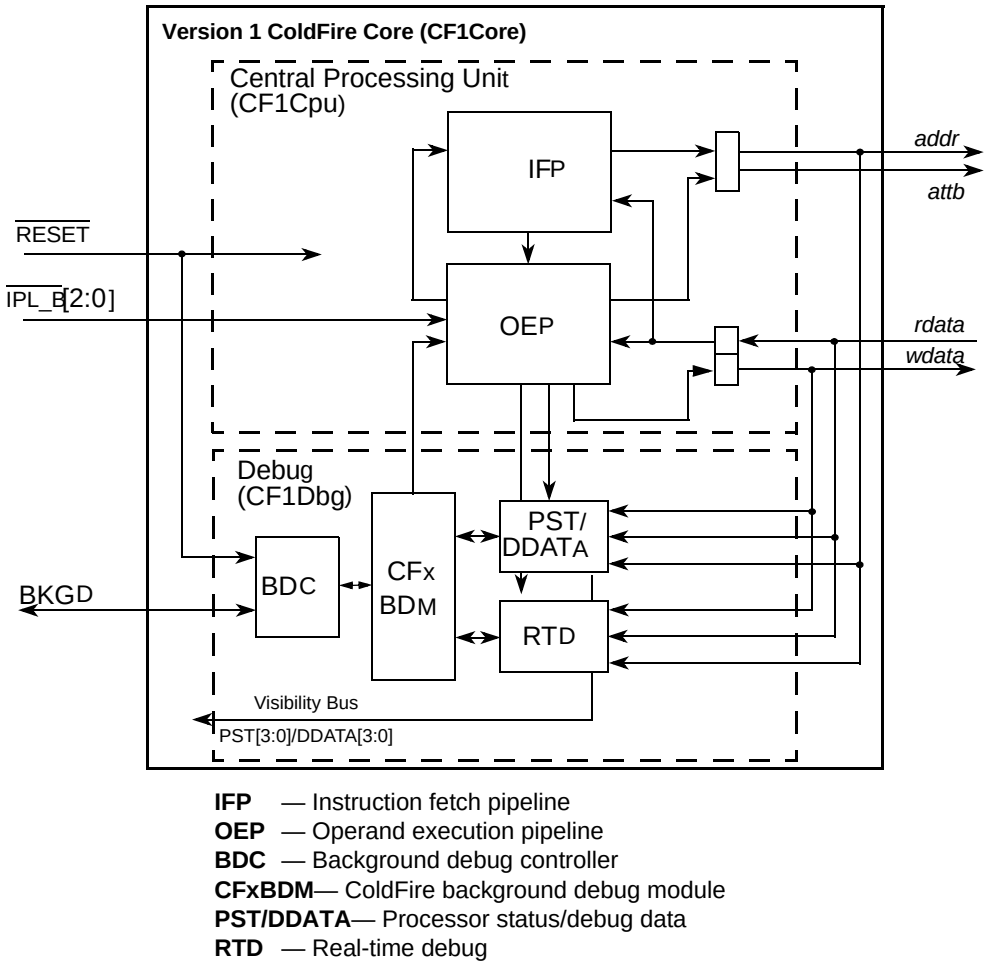
The built-in PST trace buffer (PSTB) feature is not implemented on MMA955xL. All references to this feature should be ignored.

21.2 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 communications are based on an 8-bit data packet using a single package pin (BKGD).

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 capabilities. The V1 definition supports revision B+ (DEBUG_B+) of the ColdFire debug architecture.

A simplified block diagram of the V1 core including the processor and debug module is shown in Figure 21-1 on page 348.



IFP — Instruction fetch pipeline
OEP — Operand execution pipeline
BDC — Background debug controller
CFxBDM— ColdFire background debug module
PST/DDATA— Processor status/debug data
RTD — Real-time debug

Figure 21-1. Simplified Version 1 ColdFire core block diagram

21.2.1 Overview

Debug support is divided into these 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)” on page 377.
- 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” on page 409.

There are two fields in debug registers which provide revision information: the hardware revision level in CSR and the 1-pin debug hardware revision level in CSR2. Table 21-1 summarizes the various debug revisions.

Table 21-1. Debug revision summary

Revision	CSR[HRL]	CSR2[D1HRL][1]	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

1 CSR2 is only available in Version 1 ColdFire devices.

21.2.2 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)
- Debug resources are accessible via single-pin BDM interface or the privileged WDEBUG instruction from the core

21.2.3 Modes of operations

V1 ColdFire devices typically implement a number of modes of operation, including run, wait, and stop modes. Additionally, the operation of the core's debug module is highly dependent on a number of chip configurations which determine its operating state.

When operating in secure mode, as defined by a 2-bit field in the flash memory examined at reset, BDM access to debug resources is extremely restricted. It is possible to tell that the device has been secured, and to clear security, which involves mass erasing the on-chip flash memory. No other debug access is allowed. Secure mode can be used in conjunction with each of the wait and stop low-power modes.

If the BDM interface is not enabled, access to the debug resources is limited in the same manner as a secure device.

If the device is not secure and the BDM interface is enabled (XCSR[ENBDM] is set), the device is operating in debug mode and additional resources are available via the BDM interface. In this mode, the mode of the processor (running, stopped, or halted) determines which BDM commands may be used.

Debug mode functions are managed through the background debug controller (BDC) in the Version 1 ColdFire core. The BDC provides the means for analyzing MCU operation during software development.

BDM commands can be classified into three types as shown in Table 21-2.

Table 21-2. 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 • 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” on page 386.

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 “Configuration/Status Register 2 (CSR2)” on page 361 for details)
- A background debug force reset occurs (CSR2[BDFR] is set) and CSR2[BFHBR] is set
- A computer operating properly reset occurs and CSR2[COPHR] is set
- An illegal operand reset occurs and CSR2[IOPHR] is set
- An illegal address reset occurs and CSR2[IADHR] is set
- A BACKGROUND command is received through the BKGD pin. If necessary, this wakes the device from STOP/WAIT modes.
- A properly-enabled (XCSR[ENBDM] is set) HALT instruction is executed
- Encountering a BDM breakpoint and the trigger response is programmed to generate a halt

While in halt mode, the core waits for serial background commands rather than executing instructions from the application program.

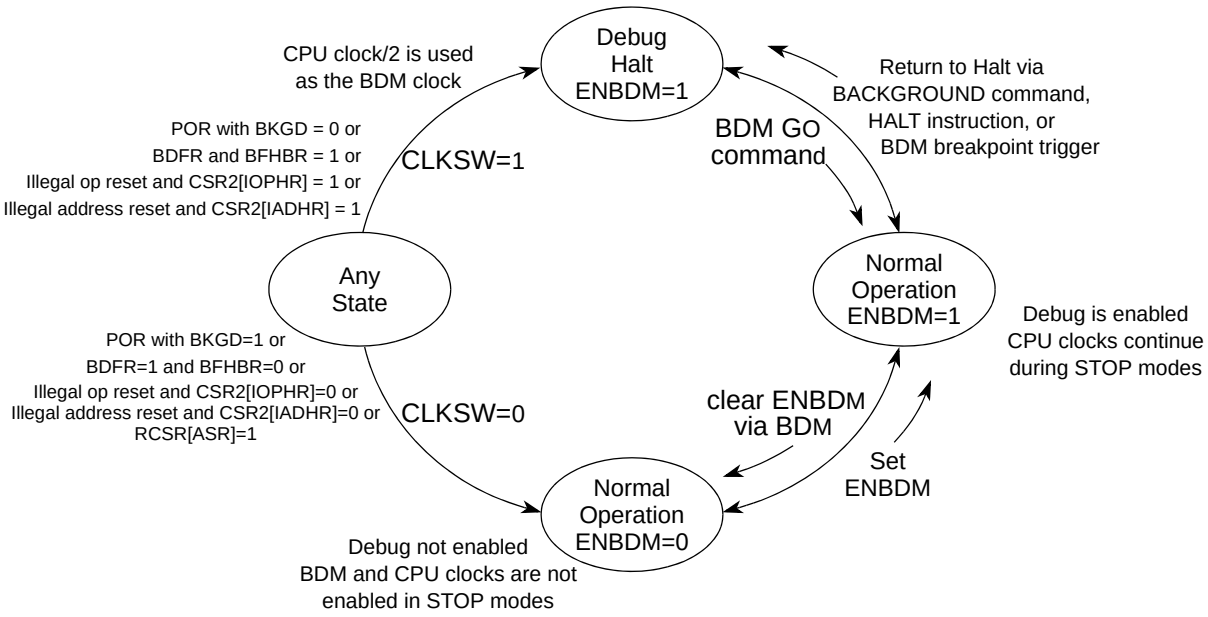


Figure 21-2. Debug modes state transition diagram

Figure 21-2 on page 351 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 is half the CPU clock. When CLKSW is cleared, the BDC serial clock is supplied from an alternate clock source.

The ENBDM bit determines if the device can be placed in halt mode, if the core and BDC serial clocks continue to run in STOP modes, and if the regulator can be placed into standby mode. Again, if booting to halt mode, XCSR[ENBDM, CLKSW] are automatically set.

If ENBDM is cleared, the ColdFire core treats the HALT instruction as an illegal instruction and generates a reset (if CPUCR[IRD] is cleared) or an exception (if CPUCR[IRD] is set) if execution is attempted.

If XCSR[ENBDM] is set, the device can be restarted from STOP/WAIT via the BDM interface.

21.3 External signal descriptions

Table 21-3 describes the debug module's 1-pin external signal (BKGD). A standard 6-pin debug connector is shown in "Freescale-recommended BDM pinout" on page 409.

Table 21-3. 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.

21.4 Memory map/register definition

In addition to the BDM commands that provide access to the processor's registers and the memory subsystem, the debug module contains a number of registers. Most of these registers are also accessible (write-only) from the processor's supervisor programming model by executing the WDEBUG instruction. Thus, the breakpoint hardware in the debug module can be read (certain registers) or written by the external development system using the serial debug interface or written by the operating system running on the processor core. Software is responsible for guaranteeing that accesses to these resources are serialized and logically consistent. The hardware provides a locking mechanism in the CSR to allow the external development system to disable any attempted writes by the processor to the breakpoint registers (setting CSR[IPW]). BDM commands must not be issued during the processor's execution of the WDEBUG instruction to configure debug module registers or the resulting behavior is undefined.

These registers, shown in Table 21-4, are treated as 32-bit quantities regardless of the number of implemented bits and unimplemented bits are reserved and must be cleared. These registers are also accessed through the BDM port by the commands, WRITE_DREG and READ_DREG, described in "BDM command set summary" on page 386. These commands contain a 5-bit field, DRc, that specifies the register, as shown in Table 21-4.

NOTE

Accessing unspecified BDM registers (either through the WDEBUG instruction or BDM commands) has "undefined" behavior.

Table 21-4. Debug module memory map

DRc	Register name	Width (bits)	Access	Reset value	Section/ Page
0x00	Configuration/status register (CSR)	32	Read/Write (BDM), Write (CPU)	0x0090_0000	Section 21.4.1, “Configuration/Status Register”
0x01	Extended Configuration/Status Register (XCSR)	32	Read/Write[1] (BDM), Write (CPU)	0x0000_0000	Section 21.4.2, “Extended Configuration/Status Register”
0x02	Configuration/Status Register 2 (CSR2)	32	Read/Write[1] (BDM), Write (CPU)	See Section	Section 21.4.3, “Configuration/Status Register 2 (CSR2)”
0x03	Configuration/Status Register 3 (CSR3)	32	Read/Write[1] (BDM), Write (CPU)	0x0000_0000	Section 21.4.4, “Configuration/Status Register 3 (CSR3)”
0x05	BDM address attribute register (BAAR)	32[2]	Write	0x0000_0005	Section 21.4.5, “BDM Address Attribute Register (BAAR)”
0x06	Address attribute trigger register (AATR)	32[2]	Write	0x0000_0005	Section 21.4.6, “Address Attribute Trigger Register (AATR)”
0x07	Trigger definition register (TDR)	32	Write	0x0000_0000	Section 21.4.7, “Trigger Definition Register”
0x08	PC breakpoint register 0 (PBR0)	32	Write	Undefined, Unaffected	Section 21.4.8, “Program Counter Breakpoint/Mask Registers”
0x09	PC breakpoint mask register (PBMR)	32	Write	Undefined, Unaffected	Section 21.4.8, “Program Counter Breakpoint/Mask Registers”
0x0C	Address breakpoint high register (ABHR)	32	Write	Undefined, Unaffected	Section 21.4.9, “Address Breakpoint Registers”
0x0D	Address breakpoint low register (ABLR)	32	Write	0x0000_0000	Section 21.4.9, “Address Breakpoint Registers”
0x0E	Data breakpoint register (DBR)	32	Write	0x0000_0000	Section 21.4.10, “Data Breakpoint and Mask Registers”
0x0F	Data breakpoint mask register (DBMR)	32	Write	0x0000_0000	Section 21.4.10, “Data Breakpoint and Mask Registers”
0x18	PC breakpoint register 1 (PBR1)	32	Write	PBR1[0] = 0	Section 21.4.8, “Program Counter Breakpoint/Mask Registers”
0x1A	PC breakpoint register 2 (PBR2)	32	Write	PBR2[0] = 0	Section 21.4.8, “Program Counter Breakpoint/Mask Registers”
0x1B	PC breakpoint register 3 (PBR3)	32	Write	PBR3[0] = 0	Section 21.4.8, “Program Counter Breakpoint/Mask Registers”

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.

NOTE

Debug control registers can be written by the external development system or the CPU through the WDEBUG instruction. These control registers are write-only from the programming model and they can be written through the BDM port using the WRITE_DREG command. In addition, the four configuration/status registers (CSR, XCSR and CSR2) 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.

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.

21.4.1 Configuration/Status Register

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.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Read	BSTAT				FOF	TRG	HALT	BKPT	HRL				0	BKD	0	IPW
Write																
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
Read	0	TRC	0	DDC		UHE	BTB		0	NPL	IPI	SSM	0	0	FID	DDH
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 21-3. Configuration/Status Register (CSR)

Table 21-5. CSR field descriptions

Bit(s)	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 or by a CSR read when a level-2 breakpoint is triggered or a level-1 breakpoint is triggered and the level-2 breakpoint is disabled. 0000 No breakpoints enabled 0001 Waiting for level-1 breakpoint 0010 Level-1 breakpoint triggered 0101 Waiting for level-2 breakpoint 0110 Level-2 breakpoint triggered
27	FOF	Fault-on-fault. Indicates a catastrophic halt occurred and forced entry into BDM. FOF is cleared by reset or when CSR is read (from the BDM port only).
26	TRG	Hardware breakpoint trigger. Indicates a hardware breakpoint halted the processor core and forced entry into BDM. Reset, the debug GO command, or reading CSR (from the BDM port only) clears TRG.
25	HALT	Processor halt. Indicates the processor executed a HALT and forced entry into BDM. Reset, the debug GO command, or reading CSR (from the BDM port only) clears HALT.
24	BKPT	Breakpoint assert. Indicates the $\overline{\text{BKPT}}$ input was asserted or a BDM BACKGROUND command received, forcing the processor into a BDM halt. Reset, the debug GO command, or reading CSR (from the BDM port only) clears BKPT.
23–20	HRL	Hardware revision level. Indicates, from the BDM port only, the level of debug module functionality. An emulator can use this information to identify the level of functionality supported. 0000 Revision A 0001 Revision B 0010 Revision C 0011 Revision D 1001 Revision B+ (The value used for this device) 1011 Revision D+
19	—	Reserved. Must be cleared.
18	BKD	Breakpoint disable. Disables the BACKGROUND command functionality, and allows the execution of the BACKGROUND command to generate a debug interrupt. 0 Normal operation 1 The receipt of a BDM BACKGROUND command signals a debug interrupt to the ColdFire core. The processor makes this interrupt request pending until the next sample point occurs, when the exception is initiated. In the ColdFire architecture, the interrupt sample point occurs once per instruction. There is no support for nesting debug interrupts.
17	—	Reserved. Must be cleared.
16	IPW	Inhibit processor writes. Inhibits processor-initiated writes to the debug module's programming model registers. IPW can be modified only by commands from the BDM interface.
15	—	Reserved. Must be cleared.
14	TRC	Force emulation mode on trace exception. 0 Processor enters supervisor mode. 1 Processor enters emulator mode when a trace exception occurs.
13	—	Reserved. Must be cleared.

Table 21-5. CSR field descriptions (continued)

Bit(s)	Field	Description
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). A non-zero value enables partial data trace capabilities. 00 No operand data is displayed. 01 Capture all write data. 10 Capture all read data. 11 Capture all read and write data.
10	UHE	User halt enable. Selects the CPU privilege level required to execute the HALT instruction. The core must be operating with XCSR[ENBDM] set to execute any HALT instruction, else the instruction is treated as an illegal opcode. 0 HALT is a supervisor-only instruction. 1 HALT is a supervisor/user instruction.
9–8	BTB	Branch target bytes. Defines the number of bytes of branch target address DDATA displays. 00 No target address capture 01 Lower 2 bytes of the target address 1*x* 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. 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. 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.
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

21.4.2 Extended Configuration/Status Register

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 (not implemented in this device).

There are multiple ways to reference the XCSR. They are summarized in Table 21-6.

Table 21-6. XCSR reference summary

Method	Reference Details
READ_XCSR_BYTE	Reads XCSR[31–24] from the BDM interface. Available in all modes.
WRITE_XCSR_BYTE	Writes XCSR[31–24] from the BDM interface. Available in all modes.
READ_DREG	Reads XCSR[31–0] from the BDM interface. Classified as a non-intrusive BDM command.
WRITE_DREG	Writes XCSR[31–0] from the BDM interface. Classified as a non-intrusive BDM command.
WDEBUG instruction	Writes XCSR[23–0] during the core's execution of WDEBUG instruction. This instruction is a privileged supervisor-mode instruction.

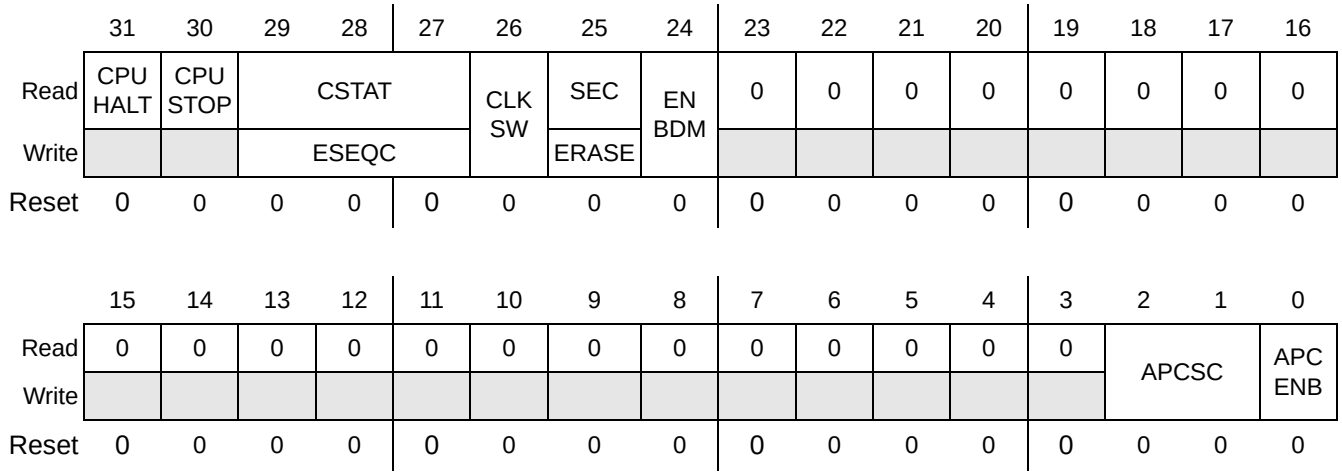


Figure 21-4. Extended Configuration/Status Register (XCSR)

Table 21-7. XCSR field descriptions

<table>
<tr><th>Bit(s)</th><th>Field</th><th>Description</th></tr>
<tr><td>31</td><td>CPUHALT</td><td>Indicates that the CPU is in the halt state. The CPU state may be running, stopped, or halted, which is determined by the CPUHALT and CPUSTOP bits as shown below.

<table>
<tr><th>XCSR [CPUHALT]</th><th>XCSR [CPUSTOP]</th><th>CPU State</th></tr>
<tr><td>0</td><td>0</td><td>Running</td></tr>
<tr><td>0</td><td>1</td><td>Stopped</td></tr>
<tr><td>1</td><td>0</td><td>Halted</td></tr>
</table>
</td></tr>
<tr><td>30</td><td>CPUSTOP</td><td>Indicates that the CPU is in the stop state. The CPU state may be running, stopped, or halted, which is determined by the CPUHALT and CPUSTOP bits as shown in the CPUHALT bit description.</td></tr>
<tr><td>29-27</td><td>CSTAT (R)
ESEQC (W)</td><td>During reads, indicates the BDM command status.
000b Command done, no errors
001b Command done, data invalid
01xb Command done, illegal
1xxb 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 XCSR[CSTAT] = 000b
<pre>
then status is okay; proceed
else
        Halt the CPU with a BDM BACKGROUND command
        Repeat steps 1,2,3
        If XCSR[CSTAT] ≠ 000, then reset device
</pre>
During writes, the ESEQC field is used for the erase sequence control during flash programming. ERASE must also be set for this bit to have an effect.
Note: See the Memory chapter for a detailed description of the algorithm for clearing security.
0000 User mass erase
Else Reserved</td></tr>
<tr><td>26</td><td>CLKSW</td><td>Select source for serial BDC communication clock.

The initial state of the XCSR[CLKSW] bit is loaded by the hardware in response to certain reset events and the state of the BKGD pin as described in Figure 21-2 on page 351.

On the MMA955xL device, the BDC clock is equal to the CPU clock. This is also equal to the synchronous bus clock rate. Therefore setting CLKSW to one has the effect of halving the BDC clock rate.
0 Alternate, asynchronous BDC clock, typically 10 MHz
1 CPU clock divided by 2</td></tr>
</table>

Table 21-7. XCSR field descriptions (continued)

Bit(s)	Field	Description
25	SEC (R) ERASE (W)	The read value of this bit typically defines the status of the flash security field. 0 Flash security is disabled 1 Flash security is enabled In addition, the SEC bit is context-sensitive during reads. After a mass-erase sequence has been initiated by BDM, it acts as a flash busy flag. When the erase operation is complete and the bit is cleared, it returns to reflect the status of the chip security. 0 Flash is not busy performing a BDM mass-erase sequence 1 Flash is busy performing a BDM mass-erase sequence During writes, this bit qualifies XCSR[ESEQC] for the write modes shown in the ESEQC field description. 0 Do not perform a mass-erase of the flash. 1 Perform a mass-erase of the flash, using the sequence specified in the XCSR[ESEQC] field.
24	ENBDM	Enable BDM. 0 BDM mode is disabled 1 Active background mode is enabled (assuming the flash is not secure)

Table 21-7. XCSR field descriptions (continued)

<table>
<tr><th>Bit(s)</th><th>Field</th><th>Description</th></tr>
<tr><td>23-3</td><td>—</td><td>Reserved.
For future use by the debug module. Must be cleared.</td></tr>
<tr><td>2-1</td><td>APCSC</td><td>Automatic PC synchronization control. Determines the periodic interval of PC address captures, if XCSR[APCENB] is set. When the selected interval is reached, a SYNC_PC command is sent to the ColdFire CPU. For more information on the SYNC_PC operation, see the APCENB description.

The chosen frequency depends on CSR2[APCDIV16] as shown in the equation and table below:
$$\text{PC address capture period} = \frac{2^{(\text{APCSC}+1)} \times 1024}{16^{\text{APCDIV16}}}$$ Eqn. 21-1

<table>
<tr><th>XCSR [APCENB]</th><th>CSR2 [APCDIV16]</th><th>XCSR [APCSC]</th><th>SYNC_PC Interval</th></tr>
<tr><td>1</td><td>0</td><td>00</td><td>2048 cycles</td></tr>
<tr><td>1</td><td>0</td><td>01</td><td>4096 cycles</td></tr>
<tr><td>1</td><td>0</td><td>10</td><td>8192 cycles</td></tr>
<tr><td>1</td><td>0</td><td>11</td><td>16384 cycles</td></tr>
<tr><td>1</td><td>1</td><td>00</td><td>128 cycles</td></tr>
<tr><td>1</td><td>1</td><td>01</td><td>256 cycles</td></tr>
<tr><td>1</td><td>1</td><td>10</td><td>512 cycles</td></tr>
<tr><td>1</td><td>1</td><td>11</td><td>1024 cycles</td></tr>
</table>
</td></tr>
<tr><td>0</td><td>APCENB</td><td>Automatic PC synchronization enable. Enables the periodic output of the PC which can be used for PST/DDATA trace synchronization.
As described in XCSR[APCSC], when the enabled periodic timer expires, a SYNC_PC command is sent to the ColdFire CPU which generates a forced instruction fetch of the next instruction. The PST/DDATA module captures the target address as defined by CSR[9] (two bytes if CSR[9] is cleared, three bytes if CSR[9] is set). This produces a PST sequence of the PST marker indicating a 2- or 3-byte address, followed by the captured instruction address.
0 Automatic PC synchronization disabled
1 Automatic PC synchronization enabled</td></tr>
</table>

21.4.3 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) which is not implemented in this device.

There are multiple ways to reference CSR2. They are summarized in Table 21-8.

Table 21-8. CSR2 Reference Summary

Method	Reference Details
READ_CSR2_BYTE	Reads CSR2[31–24] from the BDM interface. Available in all modes.
WRITE_CSR2_BYTE	Writes CSR2[31–24] from the BDM interface. Available in all modes.
READ_DREG	Reads CSR2[31–0] from the BDM interface. Classified as a non-intrusive BDM command.
WRITE_DREG	Writes CSR2[31–0] from the BDM interface. Classified as a non-intrusive BDM command.
WDEBUG Instruction	Writes CSR2[23–0] during the core's execution of WDEBUG instruction. This instruction is a privileged supervisor-mode instruction.

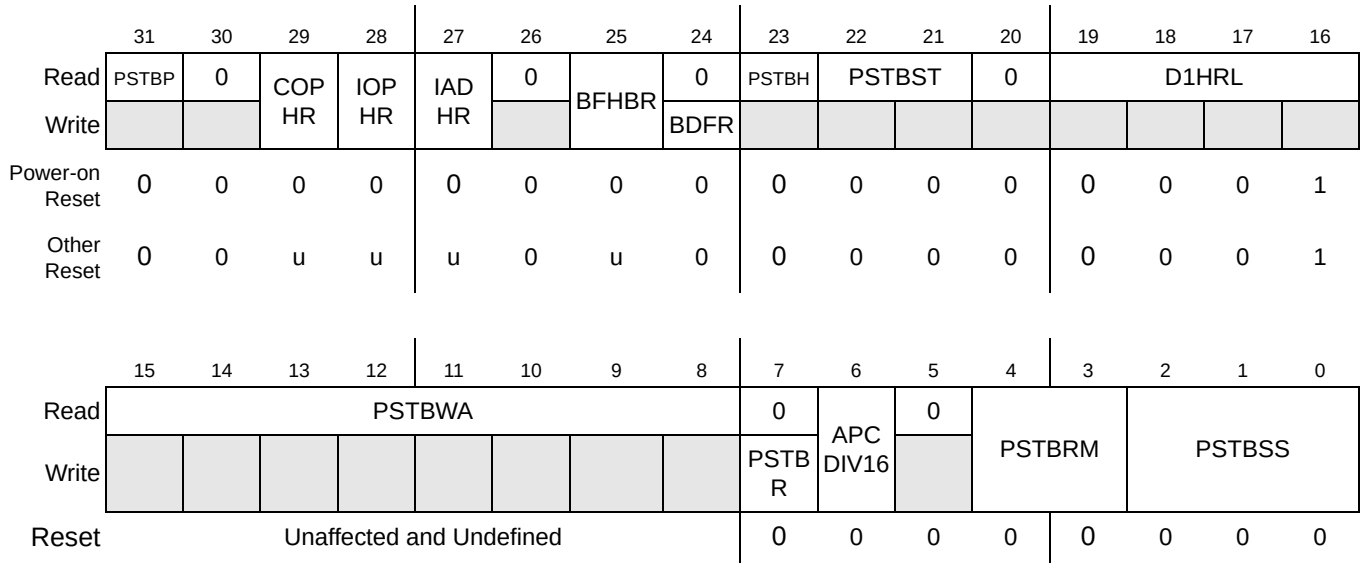


Figure 21-5. Configuration/Status Register 2 (CSR2)

Table 21-9. CSR2 Field Descriptions

Bit(s)	Field	Description
31	PSTBP	PST buffer stop. Signals if a PST buffer stop condition has been reached. 0 A PST trace buffer stop condition has not been reached 1 A PST trace buffer stop condition has been reached
30	—	Reserved. Must be cleared.
29	COPHR	Computer operating properly halt after reset. Determines operation of the device after a COP reset. This bit is cleared after a power-on reset and is unaffected by any other reset. **Note:** This bit may only be changed if XCSR[ENBDM] is set and the flash is unsecure. **Note:** The MMA955xL platform does not include COP functionality. Therefore the COPHR bit will always be inactive. 0 After a computer-operating-properly reset, the device immediately enters normal operation mode. 1 A computer-operating-properly reset immediately halts the device (as if the BKGD pin was held low after a power-on reset).
28	IOPHR	Illegal operation halt after reset. Determines operation of the device after an illegal operation reset. This bit is cleared after a power-on reset and is unaffected by any other reset. **Note:** This bit may only be changed if XCSR[ENBDM] is set and the flash is unsecure. 0 After the device has an illegal operation reset, the device immediately enters normal operation mode. 1 An illegal operation reset immediately halts the device (as if the BKGD pin was held low after a power-on reset).
27	IADHR	Illegal address halt after reset. Determines operation of the device after an illegal address reset. This bit is cleared after a power-on reset and is unaffected by any other reset. **Note:** This bit may only be changed if XCSR[ENBDM] is set and the flash is unsecure. 0 After the device has an illegal address reset, the device immediately enters normal operation mode. 1 An illegal address reset immediately halts the device (as if the BKGD pin was held low after a power-on reset).
26	—	Reserved. Must be cleared.
25	BFHBR	BDM force halt on BDM reset. Determines operation of the device after a BDM reset. This bit is cleared after a power-on reset and is unaffected by any other reset. **Note:** This bit can only change state if XCSR[ENBDM] = 1 and the flash is unsecure. 0 The device enters normal operation mode following a BDM reset. 1 The device enters in halt mode following a BDM reset, as if the BKGD pin was held low after a power-on-reset or standard BDM-initiated reset.
24	BDFR	Background debug force reset. Forces a BDM reset to the device. This bit always reads as 0 after the reset has been initiated. 0 No reset initiated. 1 Force a BDM reset.
23	—	Reserved. Must be cleared.
22–21	PSTBST	PST trace buffer state. Indicates the current state of the PST trace buffer recording. 00 PSTB disabled 01 PSTB enabled and waiting for the start condition 10 PSTB enabled, recording and waiting for the stop condition 11 PSTB enabled, completed recording after the stop condition was reached

Table 21-9. CSR2 Field Descriptions (continued)

<table>
<tr><th>Bit(s)</th><th>Field</th><th>Description</th></tr>
<tr><td>20</td><td>—</td><td>Reserved.
Must be cleared.</td></tr>
<tr><td>19–16</td><td>D1HRL</td><td>Debug 1-pin hardware revision level. Indicates the hardware revision level of the 1-pin debug module implemented in the ColdFire core. For this device, this field is 0x1.</td></tr>
<tr><td>15–8</td><td>PSTBWA</td><td>PST trace buffer write address. Indicates the current write address of the PST trace buffer. The most-significant-bit of this field is sticky; if set, it remains set until a PST/DDATA reset event occurs. As the ColdFire core inserts PST and DDATA packets into the trace buffer, this field is incremented. The value of the write address defines the next location in the PST trace buffer to be loaded. In other words, the contents of PSTB[PSTBWA-1] is the last valid entry in the trace buffer.
The msb of this field can be used to determine if the entire PST trace buffer has been loaded with valid data.
<table>
<tr><th>PSTBWA[7]</th><th>PSTB Valid Data Locations (Oldest to Newest)</th></tr>
<tr><td>0</td><td>0, 1, ... PSTBWA-1</td></tr>
<tr><td>1</td><td>PSTBWA, PSTBWA+1,..., 0, 1, PSTBWA-1</td></tr>
</table>
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.</td></tr>
<tr><td>7</td><td>PSTBR</td><td>PST trace buffer reset. Generates a reset of the PST trace buffer logic, which clears PSTBWA and PSTBST. The same resources are reset when a disabled trace buffer becomes enabled. These reset events also clear any accumulation of PSTs. This bit always reads as a zero.
0 Do not force a PST trace buffer reset
1 Force a PST trace buffer reset</td></tr>
<tr><td>6</td><td>APCDIV16</td><td>Automatic PC synchronization divide cycle counts by 16. This bit divides the cycle counts for automatic SYNC_PC command insertion by 16. See the APCSC and APCENB field descriptions.</td></tr>
</table>

Table 21-9. CSR2 Field Descriptions (continued)

<table>
<tr><th>Bit(s)</th><th>Field</th><th>Description</th></tr>
<tr><td>5</td><td>—</td><td>Reserved.
Must be cleared.</td></tr>
<tr><td>4–3</td><td>PSTBRM</td><td>PST trace buffer recording mode. Defines the trace buffer recording mode. The start and stop recording conditions are defined by the PSTBSS field.
00 Normal recording mode
01
10
11</td></tr>
<tr><td>2–0</td><td>PSTBSS</td><td>PST trace buffer start/stop definition. Specifies the start and stop conditions for PST trace buffer recording. In certain cases, the start and stop conditions are defined by the breakpoint registers. The remaining breakpoint registers are available for trigger configurations.
<table>
<tr><th>PSTBSS</th><th>Start condition</th><th>Stop condition</th></tr>
<tr><td>000</td><td colspan="2">Trace buffer disabled, no recording</td></tr>
<tr><td>001</td><td colspan="2">Unconditional recording</td></tr>
<tr><td>010</td><td rowspan="2">ABxR{& DBR/DBMR}</td><td>PBR0/PBMR</td></tr>
<tr><td>011</td><td>PBR1</td></tr>
<tr><td>100</td><td rowspan="2">PBR0/PBMR</td><td>ABxR{& DBR/DBMR}</td></tr>
<tr><td>101</td><td>PBR1</td></tr>
<tr><td>110</td><td rowspan="2">PBR1</td><td>ABxR{& DBR/DBMR}</td></tr>
<tr><td>111</td><td>PBR0/PBMR</td></tr>
</table></td></tr>
</table>

21.4.4 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 21-10. CSR3 reference summary

Method	Reference Details
READ_CSR3_BYTE	Reads CSR3[31–24] from the BDM interface. Available in all modes.
WRITE_CSR3_BYTE	Writes CSR3[31–24] from the BDM interface. Available in all modes.
READ_DREG	Reads CSR3[31–0] from the BDM interface. Classified as a non-intrusive BDM command.
WRITE_DREG	Writes CSR3[31–0] from the BDM interface. Classified as a non-intrusive BDM command.
WDEBUG Instruction	No operation during the core's execution of a WDEBUG instruction

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Read	0	BFC DIV8	BFCDIV						0	0	0	0	0	0	0	0
Write																
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
Read	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 21-6. Configuration/Status Register 3 (CSR3)

Table 21-11. CSR3 Field Descriptions

Bit(s)	Field	Description
31	—	Reserved. Must be cleared.
30	BFCDIV8	BDM flash clock divide by 8. 0 Input to the flash clock divider is the bus clock 1 Input to the flash clock divider is the bus clock divided by 8
29–24	BFCDIV	BDM flash clock divider. The BFCDIV8 and BFCDIV fields specify the frequency of the internal flash clock when performing a mass erase operation initiated by setting XCSR[ERASE]. These fields must be loaded with the appropriate values prior to the setting of XCSR[ERASE] to initiate a mass erase operation in the flash memory. This field divides the bus clock (or the bus clock divided by 8 if BFCDIV8 is set) by the value defined by the BFCDIV plus one. The resulting frequency of the internal flash clock must fall within the range of 150–200 kHz for proper flash operations. Program/erase timing pulses are one cycle of this internal flash clock, which corresponds to a range of 5–6.7 μs. The automated programming logic uses an integer number of these pulses to complete an erase or program operation. if BFCDIV8 = 0, then $f_{FCLK} = f_{Bus} \div (BFCDIV + 1)$ if BFCDIV8 = 1, then $f_{FCLK} = f_{Bus} \div (8 \times (BFCDIV + 1))$ where f_{FCLK} is the frequency of the flash clock and f_{Bus} is the frequency of the bus clock.
23–0	—	Reserved for future use by the debug module. Must be cleared.

21.4.5 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 is ignored.

	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
Read																																
Write	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	R	SZ		TT		TM		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1

Figure 21-7. BDM Address Attribute Register (BAAR)

Table 21-12. BAAR field descriptions

Bit(s)	Field	Description
31–8	—	Reserved for future use by the debug module. Must be cleared.
7	R	Read/Write. 0 Write 1 Read
6–5	SZ	Size. 00 Longword 01 Byte 10 Word 11 Reserved
4–3	TT	Transfer type. See the TT definition in the AATR description, “Address Attribute Trigger Register (AATR)” on page 367.
2–0	TM	Transfer modifier. See the TM definition in the AATR description, “Address Attribute Trigger Register (AATR)” on page 367.

21.4.6 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.

	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
Read																																
Write	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	RM	SZM		TTM		TMM			R	SZ		TT		TM		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1

Figure 21-8. Address Attribute Trigger Register (AATR)

Table 21-13. AATR field descriptions

Bit(s)	Field	Description
31–16	—	Reserved. Must be cleared.
15	RM	Read/write mask. Masks the R bit in address comparisons.
14–13	SZM	Size mask. Masks the corresponding SZ bit in address comparisons.
12–11	TTM	Transfer type mask. Masks the corresponding TT bit in address comparisons.
10–8	TMM	Transfer modifier mask. Masks the corresponding TM bit in address comparisons.
7	R	Read/write. R is compared with the Read/$\overline{\text{Write}}$ 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

21.4.7 Trigger Definition Register

TDR configures the operation of the hardware breakpoint logic that corresponds with the ABHR/ABLR/AATR, PBR/PBR1/PBR2/PBR3/PBMR, and DBR/DBMR registers within the debug module. TDR controls the actions taken under the defined conditions. Breakpoint logic may be configured as one- or two-level trigger. TDR[31–16] defines the second-level trigger, and TDR[15–0] defines the first-level trigger.

NOTE

The debug module has no hardware interlocks. To prevent spurious breakpoint triggers while the breakpoint registers are being loaded, disable TDR (clear TDR[L2EBL,L1EBL]) before defining triggers.

A write to TDR clears the CSR trigger status bits, CSR[BSTAT]. TDR is accessible in supervisor mode as debug control register 0x07 using the WDEBUG instruction and through the BDM port using the WRITE_DREG command.

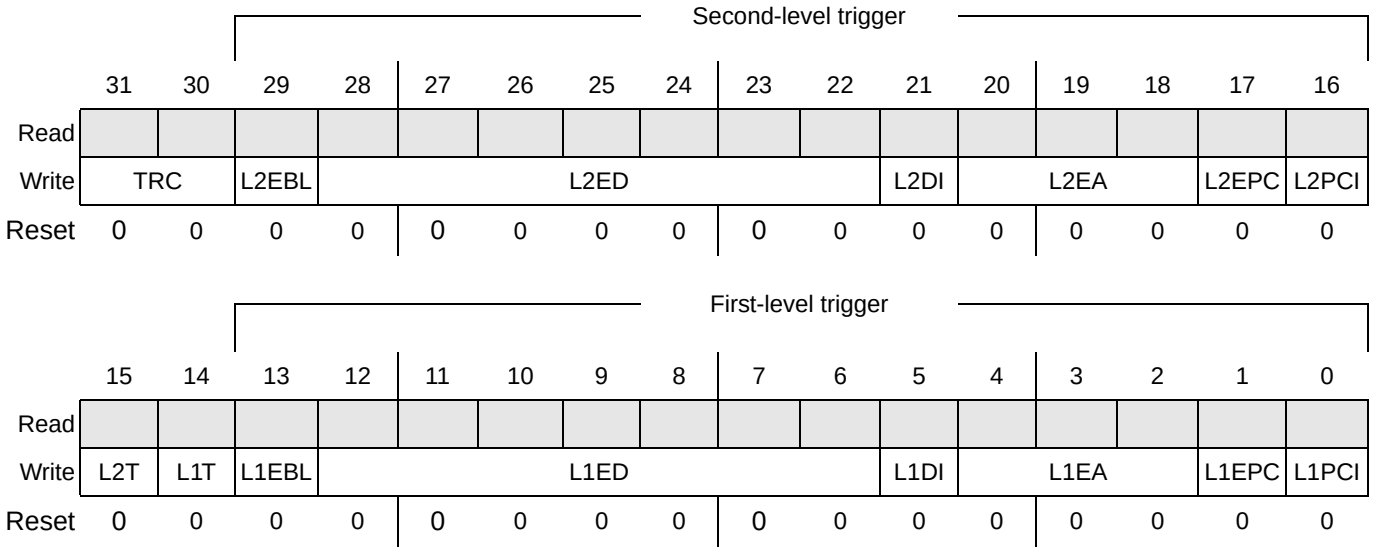


Figure 21-9. Trigger Definition Register (TDR)

Table 21-14. TDR field descriptions

<table>
<tr><th>Bit(s)</th><th>Field</th><th>Description</th></tr>
<tr><td>31–30</td><td>TRC</td><td>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</td></tr>
<tr><td>29</td><td>L2EBL</td><td>Enable level 2 breakpoint. Global enable for the breakpoint trigger.
0 Disables all level 2 breakpoints
1 Enables all level 2 breakpoint triggers</td></tr>
<tr><td>28–22</td><td>L2ED</td><td>Enable level 2 data breakpoint. Setting an L2ED bit enables the corresponding data breakpoint condition based on the size and placement on the processor’s local data bus. Clearing all ED bits disables data breakpoints.
<table>
<tr><th>TDR bit</th><th>Description</th></tr>
<tr><td>28</td><td>Data longword. Entire processor’s local data bus.</td></tr>
<tr><td>27</td><td>Lower data word.</td></tr>
<tr><td>26</td><td>Upper data word.</td></tr>
<tr><td>25</td><td>Lower lower data byte. Low-order byte of the low-order word.</td></tr>
<tr><td>24</td><td>Lower middle data byte. High-order byte of the low-order word.</td></tr>
<tr><td>23</td><td>Upper middle data byte. Low-order byte of the high-order word.</td></tr>
</table></td></tr>
<tr><td>21</td><td>L2DI</td><td>Level 2 data breakpoint invert. Inverts the logical sense of all the data breakpoint comparators. This can develop a trigger based on the occurrence of a data value other than the DBR contents.
0 No inversion
1 Invert data breakpoint comparators.</td></tr>
<tr><td>20–18</td><td>L2EA</td><td>Enable level 2 address breakpoint. Setting an L2EA bit enables the corresponding address breakpoint. Clearing all three bits disables the breakpoint.
<table>
<tr><th>TDR bit</th><th>Description</th></tr>
<tr><td>20</td><td>Address breakpoint inverted. Breakpoint is based outside the range between ABLR and ABHR.</td></tr>
<tr><td>19</td><td>Address breakpoint range. The breakpoint is based on the inclusive range defined by ABLR and ABHR.</td></tr>
<tr><td>18</td><td>Address breakpoint low. The breakpoint is based on the address in the ABLR.</td></tr>
</table></td></tr>
</table>

Table 21-14. TDR field descriptions (continued)

<table>
<tr><th>Bit(s)</th><th>Field</th><th>Description</th></tr>
<tr><td>17</td><td>L2EPC</td><td>Enable level 2 PC breakpoint.
0 Disable PC breakpoint
1 Enable PC breakpoint</td></tr>
<tr><td>16</td><td>L2PCI</td><td>Level 2 PC breakpoint invert.
0 The PC breakpoint is defined within the region defined by PBRn and PBMR.
1 The PC breakpoint is defined outside the region defined by PBRn and PBMR.</td></tr>
<tr><td>15</td><td>L2T</td><td>Level 2 trigger. Determines the logic operation for the trigger between the PC_condition and the (Address_range and Data) condition where the inclusion of a Data_condition is optional. The ColdFire debug architecture supports the creation of single or double-level triggers.
0 Level 2 trigger = PC_condition and (Address_range and Data_condition)
1 Level 2 trigger = PC_condition | (Address_range and Data_condition)</td></tr>
<tr><td>14</td><td>L1T</td><td>Level 1 trigger. Determines the logic operation for the trigger between the PC_condition and the (Address_range and Data) condition where the inclusion of a Data_condition is optional. The ColdFire debug architecture supports the creation of single or double-level triggers.
0 Level 1 trigger = PC_condition and (Address_range and Data_condition)
1 Level 1 trigger = PC_condition | (Address_range and Data_condition)</td></tr>
<tr><td>13</td><td>L1EBL</td><td>Enable level 1 breakpoint. Global enable for the breakpoint trigger.
0 Disables all level 1 breakpoints
1 Enables all level 1 breakpoint triggers</td></tr>
<tr><td>12–6</td><td>L1ED</td><td>Enable level 1 data breakpoint. Setting an L1ED bit enables the corresponding data breakpoint condition based on the size and placement on the processor’s local data bus. Clearing all L1ED bits disables data breakpoints.
<table>
<tr><th>TDR bit</th><th>Description</th></tr>
<tr><td>12</td><td>Data longword. Entire processor’s local data bus.</td></tr>
<tr><td>11</td><td>Lower data word.</td></tr>
<tr><td>10</td><td>Upper data word.</td></tr>
<tr><td>9</td><td>Lower lower data byte. Low-order byte of the low-order word.</td></tr>
<tr><td>8</td><td>Lower middle data byte. High-order byte of the low-order word.</td></tr>
<tr><td>7</td><td>Upper middle data byte. Low-order byte of the high-order word.</td></tr>
</table></td></tr>
<tr><td>5</td><td>L1DI</td><td>Level 1 data breakpoint invert. Inverts the logical sense of all the data breakpoint comparators. This can develop a trigger based on the occurrence of a data value other than the DBR contents.
0 No inversion
1 Invert data breakpoint comparators.</td></tr>
</table>

Table 21-14. TDR field descriptions (continued)

<table>
<tr><th>Bit(s)</th><th>Field</th><th>Description</th></tr>
<tr><td>4–2</td><td>L1EA</td><td>Enable level 1 address breakpoint. Setting an L1EA bit enables the corresponding address breakpoint. Clearing all three bits disables the address breakpoint.
<table>
<tr><th>TDR bit</th><th>Description</th></tr>
<tr><td>4</td><td>Enable address breakpoint inverted. Breakpoint is based outside the range between ABLR and ABHR.</td></tr>
<tr><td>3</td><td>Enable address breakpoint range. The breakpoint is based on the inclusive range defined by ABLR and ABHR.</td></tr>
<tr><td>2</td><td>Enable address breakpoint low. The breakpoint is based on the address in the ABLR.</td></tr>
</table></td></tr>
<tr><td>1</td><td>L1EPC</td><td>Enable level 1 PC breakpoint.
0 Disable PC breakpoint
1 Enable PC breakpoint</td></tr>
<tr><td>0</td><td>L1PCI</td><td>Level 1 PC breakpoint invert.
0 The PC breakpoint is defined within the region defined by PBRn and PBMR.
1 The PC breakpoint is defined outside the region defined by PBRn and PBMR.</td></tr>
</table>

21.4.8 Program Counter Breakpoint/Mask Registers

The PBR*n* registers define instruction addresses for use as part of the trigger. These registers' contents are compared with the processor's program counter register when the appropriate valid bit is set (for PBR1–3) and TDR is configured appropriately. PBR0 bits are masked by setting corresponding PBMR bits (PBMR has no effect on PBR1–3). Results are compared with the processor's program counter register, as defined in TDR. The PC breakpoint registers, PBR1–3, have no masking associated with them, but do include a valid bit. These registers' contents are compared with the processor's program counter register when TDR is configured appropriately.

The PC breakpoint registers are accessible in supervisor mode using the WDEBUG instruction and through the BDM port using the WRITE_DREG command using values shown in "BDM command set descriptions" on page 384.

NOTE

Version 1 ColdFire core devices implement a 24-bit, 16-Mbyte address map. When programming these registers with a 32-bit address, the upper byte should be zero-filled when referencing the flash, RAM, and RGPIO regions, and set to 0xFF when referencing any of the slave peripheral devices.

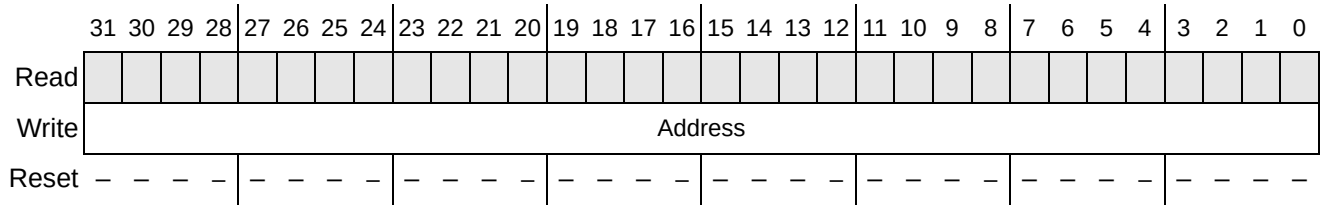


Figure 21-10. Program Counter Breakpoint Register 0 (PBR0)

Table 21-15. PBR0 field descriptions

Bit(s)	Field	Description
31–0	Address	PC breakpoint address. The address to be compared with the PC as a breakpoint trigger. Because all instruction sizes are multiples of 2 bytes, bit 0 of the address should always be zero.

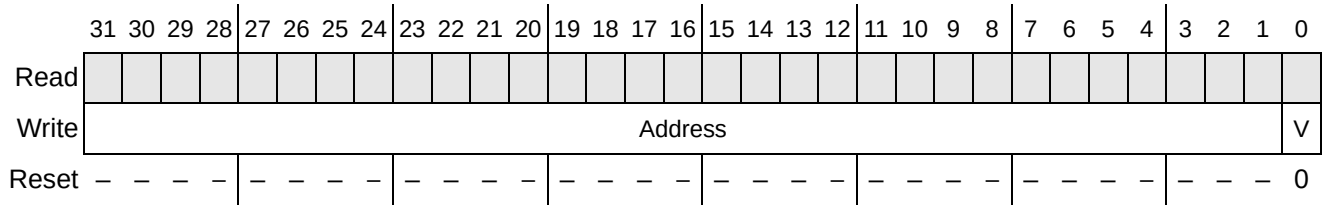


Figure 21-11. Program Counter Breakpoint Register *n* (PBR*n, n = 1,2,3*)

Table 21-16. PBR*n* field descriptions

Bit(s)	Field	Description
31–1	Address	PC breakpoint address. The 31-bit address to be compared with the PC as a breakpoint trigger.
0	V	Valid bit. This bit must be set for the PC breakpoint to occur at the address specified in the Address field. 0 PBR is disabled. 1 PBR is enabled.

Figure 21-12 shows PBMR. PBMR is accessible in supervisor mode using the WDEBUG instruction and via the BDM port using the WRITE_DREG command. PBMR only masks PBR0.

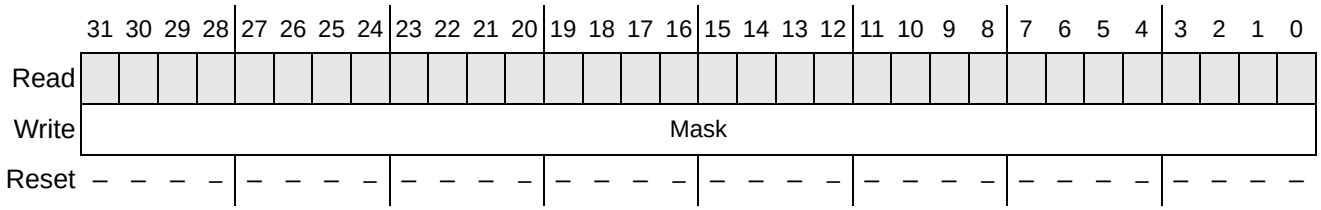


Figure 21-12. Program Counter Breakpoint Mask Register (PBMR)

Table 21-17. PBMR field descriptions

Bit(s)	Field	Description
31–0	Mask	PC breakpoint mask. 0 The corresponding PBR0 bit is compared to the appropriate PC bit. 1 The corresponding PBR0 bit is ignored.

21.4.9 Address Breakpoint Registers

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" on page 384.

NOTE

Version 1 ColdFire core devices implement a 24-bit, 16-Mbyte address map. When programming these registers with a 32-bit address, the upper byte should be zero-filled when referencing the flash, RAM, and RGPIO regions, and set to 0xFF when referencing any of the slave peripheral devices.

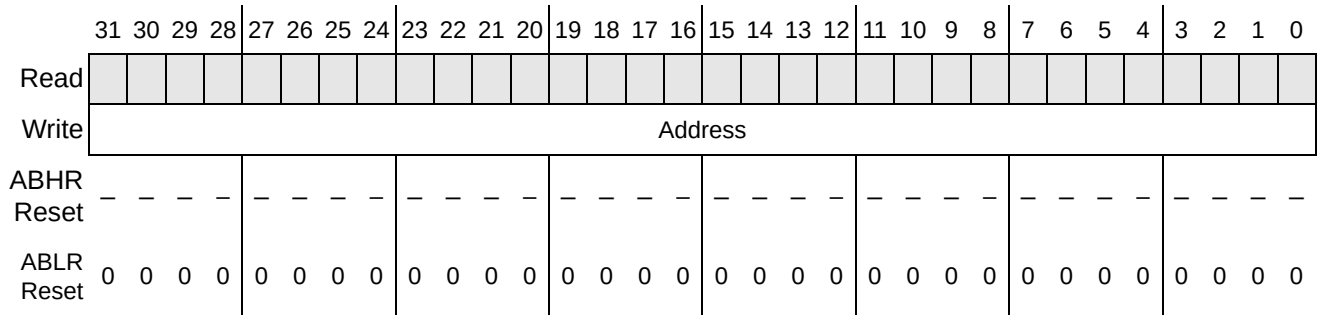


Figure 21-13. Address Breakpoint Registers (ABLR, ABHR)

Table 21-18. ABLR field description

Bit(s)	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 21-19. ABHR field description

Bit(s)	Field	Description
31–0	Address	High address. Holds the 32-bit address marking the upper bound of the address breakpoint range.

21.4.10 Data Breakpoint and Mask Registers

DBR specifies data patterns used as part of the trigger into debug mode. DBR bits are masked by setting corresponding DBMR bits, as defined in TDR.

DBR and DBMR are accessible in supervisor mode using the WDEBUG instruction and through the BDM port using the WRITE_DREG commands.

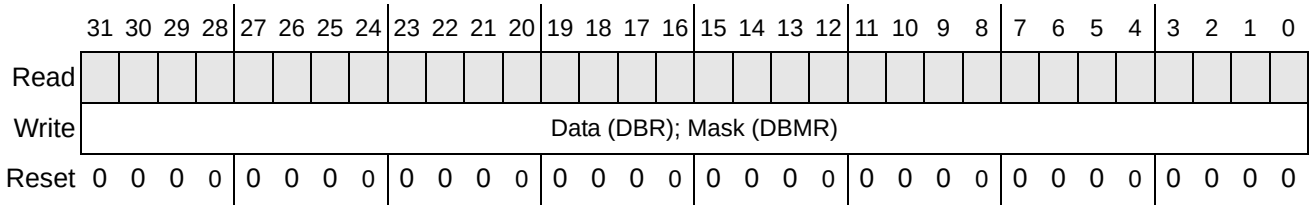


Figure 21-14. Data Breakpoint and Mask Registers (DBR, DBMR)

Table 21-20. DBR field descriptions

Bit(s)	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 21-21. DBMR field descriptions

Bit(s)	Field	Description
31–0	Mask	Data breakpoint mask. The 32-bit mask for the data breakpoint trigger. 0 The corresponding DBR bit is compared to the appropriate bit of the processor's local data bus 1 The corresponding DBR bit is ignored

The DBR supports aligned and misaligned references. Table 21-22 shows the relationships between processor address, access size, and location within the 32-bit data bus.

Table 21-22. Access size and operand data location

Address[1–0]	Access Size	Operand Location
00	Byte	D[31–24]
01	Byte	D[23–16]
10	Byte	D[15–8]
11	Byte	D[7–0]
0*x*	Word	D[31–16]
1*x*	Word	D[15–0]
xx	Longword	D[31–0]

21.4.11 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
- {} denotes an optional additional trigger term

One-level triggers of the form shown in the following example.

Example 21-1. One-level breakpoint trigger

```
if   (PC_breakpoint)
if   (PC_breakpoint || Address_breakpoint{&& Data_breakpoint})
if   (Address_breakpoint {&& Data_breakpoint})
```

Two-level triggers of the form shown in the following example.

Example 21-2. Two-level breakpoint trigger

```
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 (not implemented in this device). For information on this functionality, see “Configuration/Status Register 2 (CSR2)” on page 361.

21.5 Functional description

21.5.1 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 21-1 on page 348, the BDC module interfaces between the single-pin (BKGD) interface and the remaining debug modules, including the ColdFire background debug logic and the real-time debug hardware. This interface provides a convenient means for programming the on-chip flash and other non-volatile memories. The BDC is the primary debug interface for development and allows non-intrusive access to memory data and traditional debug features such as run/halt control, read/write of core registers, breakpoints, and single instruction step.

Features of the background debug controller (BDC) include:

- Single dedicated pin for mode selection and background communications
- Special BDC registers not located in system memory map
- SYNC command to determine target communications rate
- Non-intrusive commands for memory access
- Active background (halt) mode commands for core register access
- GO command to resume execution
- BACKGROUND command to halt core or wake CPU from low-power modes
- Oscillator runs in stop mode, if BDM enabled

Based on these features, BDM is useful for the following reasons:

- In-circuit emulation is not needed, so physical and electrical characteristics of the system are not affected.
- BDM is always available for debugging the system and provides a communication link for upgrading firmware in existing systems.
- Provides high-speed memory downloading, especially useful for flash programming
- Provides absolute control of the processor, and thus the system. This feature allows quick hardware debugging with the same tool set used for firmware development.

21.5.1.1 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 21-23. CPU halt sources

Halt source	Halt timing	Description		
Fault-on-fault	Immediate	Refers to the occurrence of any fault while exception processing. For example, a bus error is signaled during exception stack frame writes or while fetching the first instruction in the exception service routine.		
		CPUCR[ARD] = 1	Immediately enters halt.	
		CPUCR[ARD] = 0	Reset event is initiated.	
Hardware breakpoint trigger	Pending	Halt is made pending in the processor. The processor samples for pending halt and interrupt conditions once per instruction. When a pending condition is asserted, the processor halts execution at the next sample point.		
HALT instruction	Immediate	BDM disabled	CPUCR[IRD] = 0	A reset is initiated since attempted execution of an illegal instruction
			CPUCR[IRD] = 1	An illegal instruction exception is generated.
		BDM enabled, supervisor mode	Processor immediately halts execution at the next instruction sample point. The processor can be restarted by a BDM GO command. Execution continues at the instruction after HALT.	
		BDM enabled, user mode	CSR[UHE] = 0 CPUCR[IRD] = 0	A reset event is initiated, because a privileged instruction was attempted in user mode.
			CSR[UHE] = 0 CPUCR[IRD] = 1	A privilege violation exception is generated.
			CSR[UHE] = 1	Processor immediately halts execution at the next instruction sample point. The processor can be restarted by a BDM GO command. Execution continues at the instruction after HALT.
BACKGROUND command	Pending	BDM disabled or flash secure	Illegal command response and BACKGROUND command is ignored.	
		BDM enabled and flash unsecure	Processor is running	Halt is made pending in the processor. The processor samples for pending halt and interrupt conditions once per instruction. When a pending condition is asserted, the processor halts execution at the next sample point.
			Processor is stopped	Processing of the BACKGROUND command is treated in a special manner. The processor exits the stopped mode and enters the halted state, at which point all BDM commands may be exercised. When restarted, the processor continues by executing the next sequential instruction (the instruction following STOP).

Table 21-23. CPU halt sources (continued)

Halt source	Halt timing	Description	
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.
		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. The debug GO command also clears CSR[26–24].

21.5.1.2 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" on page 380.

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 pull-up so no external pull-up 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" on page 380 for more details.

When no debugger pod is connected to the standard 6-pin BDM interface connector (“Freescale-recommended BDM pinout” on page 409), the internal pull-up 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].

21.5.1.3 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] in Table 21-6.

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.

Figure 21-15 shows an external host transmitting a logic 1 or 0 to the BKGD pin of a target MCU. The host is asynchronous to the target so there is a 0–1 cycle delay from the host-generated falling edge to where the target perceives the beginning of the bit time. Ten target BDC clock cycles later, the target senses the bit level on the BKGD pin. Typically, the host actively drives the pseudo-open-drain BKGD pin during host-to-target transmissions to speed up rising edges. Because the target does not drive the BKGD pin during the host-to-target transmission period, there is no need to treat the line as an open-drain signal during this period.

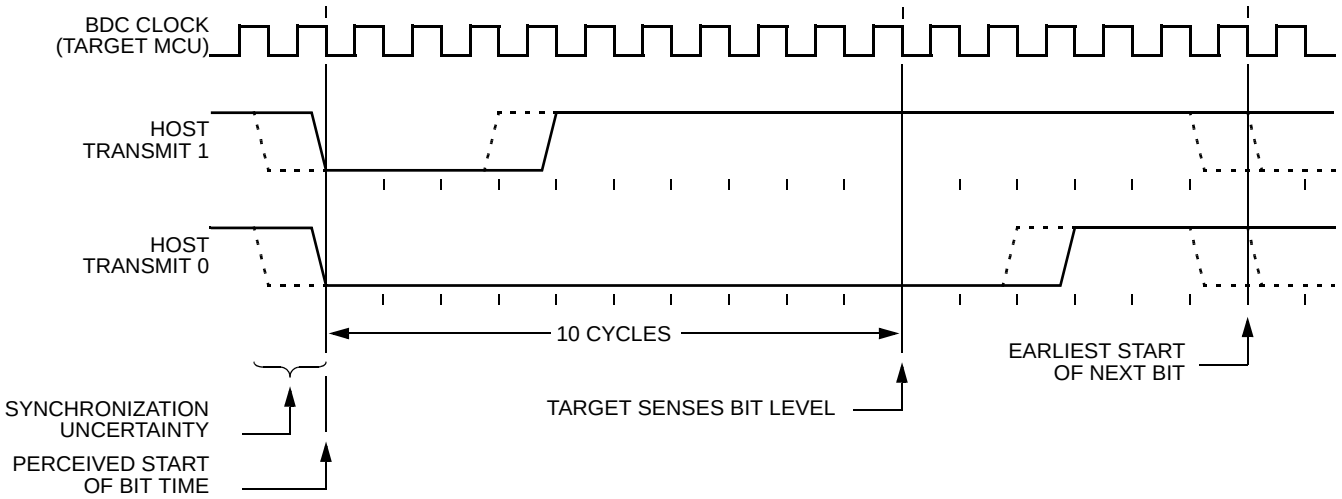


Figure 21-15. BDC host-to-target serial bit timing

Figure 21-16 shows the host receiving a logic 1 from the target MCU. Because the host is asynchronous to the target MCU, there is a 0–1 cycle delay from the host-generated falling edge on BKGD to the perceived start of the bit time in the target MCU. The host holds the BKGD pin low long enough for the target to recognize it (at least two target BDC cycles). The host must release the low drive before the target MCU drives a brief active-high speedup pulse seven cycles after the perceived start of the bit time. The host should sample the bit level about 10 cycles after it started the bit time.

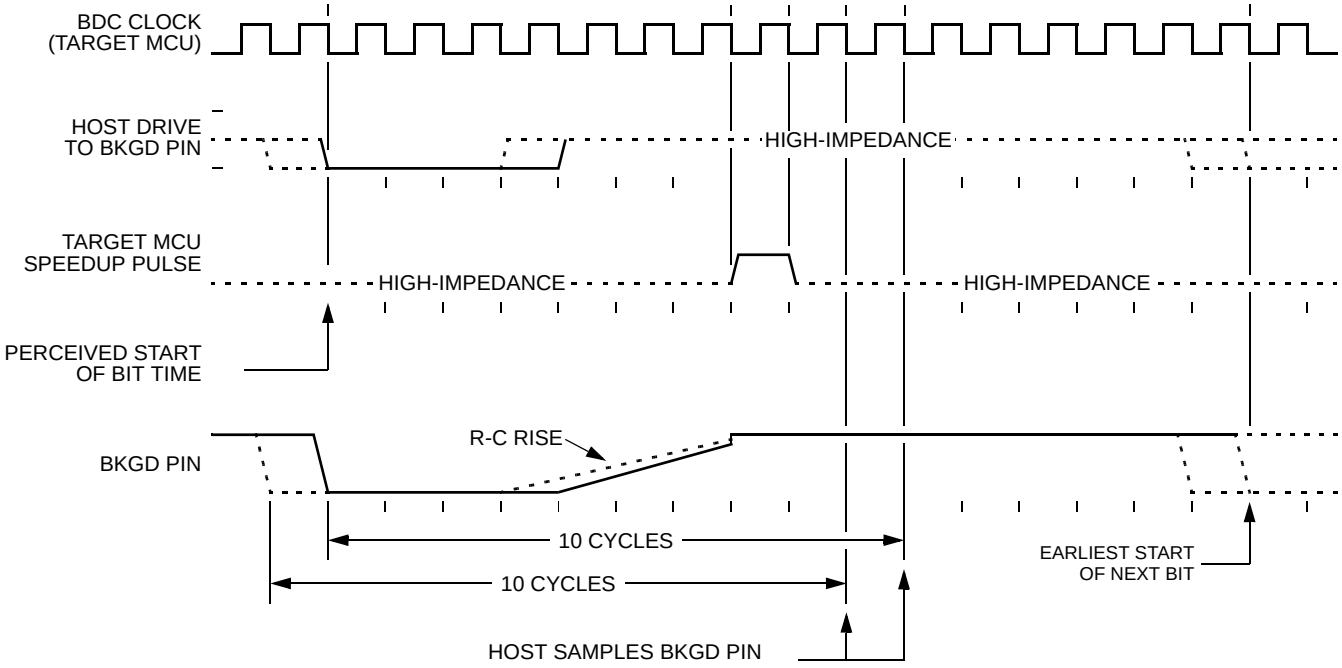


Figure 21-16. BDC target-to-host serial bit timing (Logic 1)

Figure 21-17 shows the host receiving a logic 0 from the target MCU. Because the host is asynchronous to the target MCU, there is a 0–1 cycle delay from the host-generated falling edge on BKGD to the start of the bit time as perceived by the target MCU. The host initiates the bit time, but the target MCU finishes it. Because the target wants the host to receive a logic 0, it drives the BKGD pin low for 13 BDC clock cycles, then briefly drives it high to speed up the rising edge. The host samples the bit level about 10 cycles after starting the bit time.

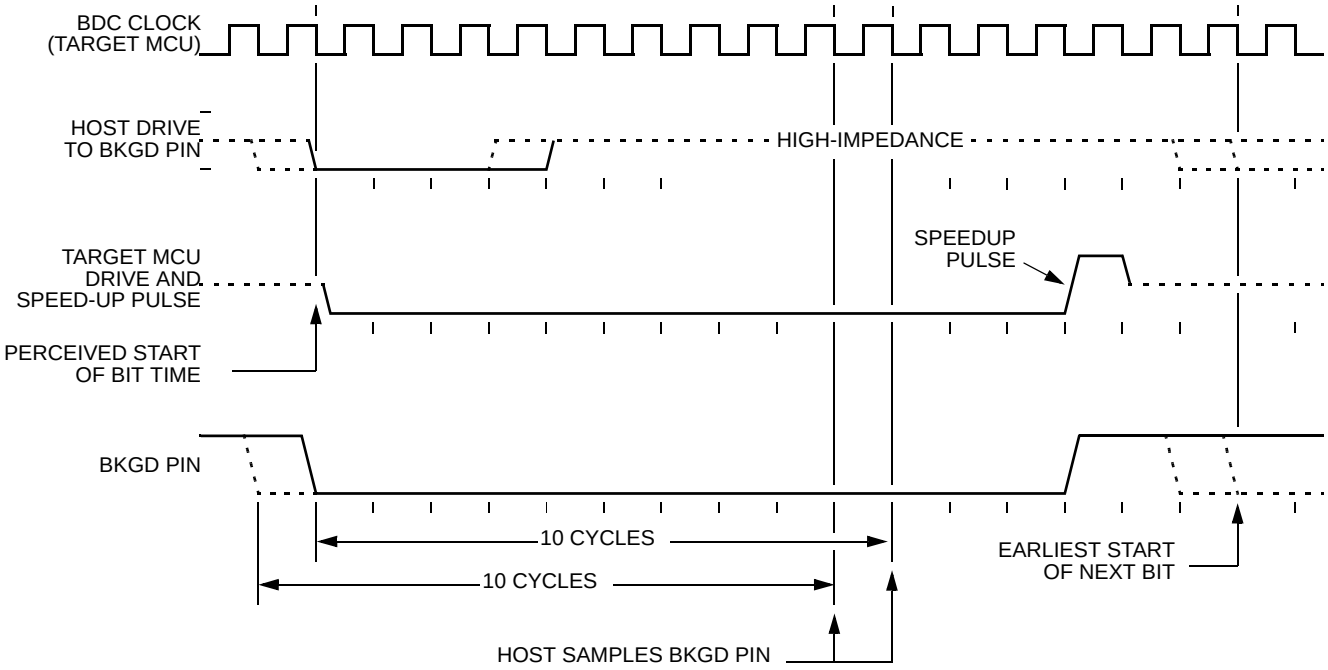


Figure 21-17. BDM target-to-host serial bit timing (Logic 0)

21.5.1.4 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 Figure 21-18.

Miscellaneous Commands

	7	6	5	4	3	2	1	0
Write	0	0	R/$\overline{W}$	0	MSCMD			
Read/Write	Optional Command Extension Byte (Data)							

Memory Commands

	7	6	5	4	3	2	1	0
Write	0	0	R/$\overline{W}$	1	SZ		MCMD	
W if addr, R/W if data	Command Extension Bytes (Address, Data)							

Core Register Commands

	7	6	5	4	3	2	1	0
Write	CRG		R/$\overline{W}$	CRN				
Read/Write	Command Extension Bytes (Data)							

PST Trace Buffer Read Commands

	7	6	5	4	3	2	1	0
Write	0	1	0	CRN				
Read	Trace Buffer Data[31–24], see Figure 21-44 on page 404							
Read	Trace Buffer Data[23–16], see Figure 21-44 on page 404							
Read	Trace Buffer Data[15–08], see Figure 21-44 on page 404							
Read	Trace Buffer Data[07–00], see Figure 21-44 on page 404							

Figure 21-18. BDM command code encoding

Table 21-24. BDM command code field descriptions

Bit(s)	Field	Description
5	R/$\overline{W}$	Read/$\overline{Write}$. 0 Command is performing a write operation. 1 Command is performing a read operation.
3–0	MSCMD	Miscellaneous command. Defines the miscellaneous command to be performed. 0000 No operation 0001 Display the CPU's program counter (PC) plus optional capture in the PST trace buffer 0010 Enable the BDM acknowledge communication mode 0011 Disable the BDM acknowledge communication mode 0100 Force a CPU halt (background) 1000 Resume CPU execution (go) 1101 Read/write of the debug XCSR most significant byte 1110 Read/write of the debug CSR2 most significant byte 1111 Read/write of the debug CSR3 most significant byte
3–2	SZ	Memory operand size. Defines the size of the memory reference. 00 8-bit byte 01 16-bit word 10 32-bit long
1–0	MCMD	Memory command. Defines the type of the memory reference to be performed. 00 Simple write if Read/$\overline{Write}$ = 0; simple read if Read/$\overline{Write}$ = 1 01 Write + status if Read/$\overline{Write}$ = 0; read + status if Read/$\overline{Write}$ = 1 10 Fill if Read/$\overline{Write}$ = 0; dump if Read/$\overline{Write}$ = 1 11 Fill + status if Read/$\overline{Write}$ = 0; dump + status if Read/$\overline{Write}$ = 1
7–6	CRG	Core register group. Defines the core register group to be referenced. 01 CPU's general-purpose registers (An, Dn) or PST trace buffer 10 DBG's control registers 11 CPU's control registers (PC, SR, VBR, CPUCR,...)
4–0	CRN	Core register number. Defines the specific core register (its number) to be referenced. All other CRN values are reserved.

CRG	CRN	Register
01	0x00–0x07	D0–7
	0x08–0x0F	A0–7
	0x10–0x1B	PST Buffer 0–11
10	DRc[4:0] as described in Table 21-4 on page 353	
11	0x00	OTHER_A7
	0x01	VBR
	0x02	CPUCR
	0x0E	SR
	0x0F	PC

21.5.1.5 BDM command set summary

Table 21-25 summarizes the BDM command set. Subsequent paragraphs contain detailed descriptions of each command. The nomenclature below is used in Table 21-25 to describe the structure of the BDM commands.

Commands begin with an 8-bit hexadecimal command code in the host-to-target direction (most significant bit first).

/ = separates parts of the command

d = delay 16 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 21-25. 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.

Table 21-25. BDM command summary (continued)

Command mnemonic	Command classification	ACK if enb?[1]	Command structure	Description
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.
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_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

Table 21-25. BDM command summary (continued)

Command mnemonic	Command classification	ACK if enb?[1]	Command structure	Description
READ_CSR3_BYTE	Always Available	No	0x2F/rd8	Read the most significant byte of the debug module's CSR3
WRITE_CREG	Active Background	Yes	(0xC0+CRN)/wd32/d	Write one of the CPU's control registers
WRITE_DREG	Non-Intrusive	Yes	(0x80+CRN)/wd32/d	Write one of the debug module's control registers
WRITE_MEM.sz	Non-Intrusive	Yes	(0x10+4 x sz)/ad24/wd.sz/d	Write the appropriately-sized (sz) memory value to the location specified by the 24-bit address
WRITE_MEM.sz_WS	Non-Intrusive	No	(0x11+4 x sz)/ad24/wd.sz/d/ss	Write the appropriately-sized (sz) memory value to the location specified by the 24-bit address and report status
WRITE_Rn	Active Background	Yes	(0x40+CRN)/wd32/d	Write the requested general-purpose register (An, Dn) of the CPU
WRITE_XCSR_BYTE	Always Available	No	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 "Hardware handshake abort procedure" on page 406, 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.

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.

ACK_DISABLE

Disable host/target handshake protocol **Always Available**

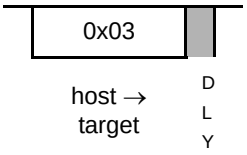


Figure 21-19.

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.

ACK_ENABLE

Enable host/target handshake protocol **Always Available**

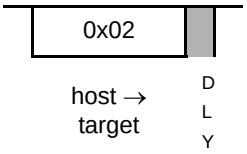


Figure 21-20.

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” on page 404 and “Hardware handshake abort procedure” on page 406.

BACKGROUND

Enter active background mode (if enabled) **Non-intrusive**

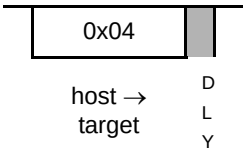


Figure 21-21.

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 16 BDC clock cycles is required after the BACKGROUND command to allow the target MCU to finish its current CPU instruction and enter active background mode before a new BDC command can be accepted.

After the target MCU is reset into a normal operating mode, the host debugger would send a WRITE_XCSR_BYTE command to set ENBDM before attempting to send the BACKGROUND command the first time. Normally, the development host would set ENBDM once at the beginning of a debug session or after a target system reset, and then leave the ENBDM bit set during debugging operations. During debugging, the host would use GO commands to move from active background mode to normal user program execution and would use BACKGROUND commands or breakpoints to return to active background mode.

DUMP_MEM.sz, DUMP_MEM.sz_WS

DUMP_MEM.sz

Read memory specified by debug address register, then increment address **Non-intrusive**

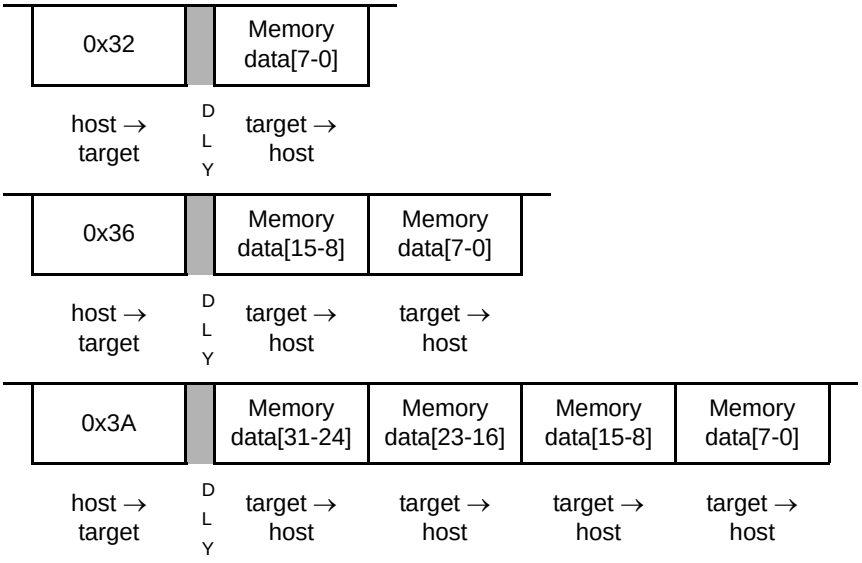


Figure 21-22.

DUMP_MEM.sz_WS

Read memory specified by debug address register with status, then increment address

Non-intrusive

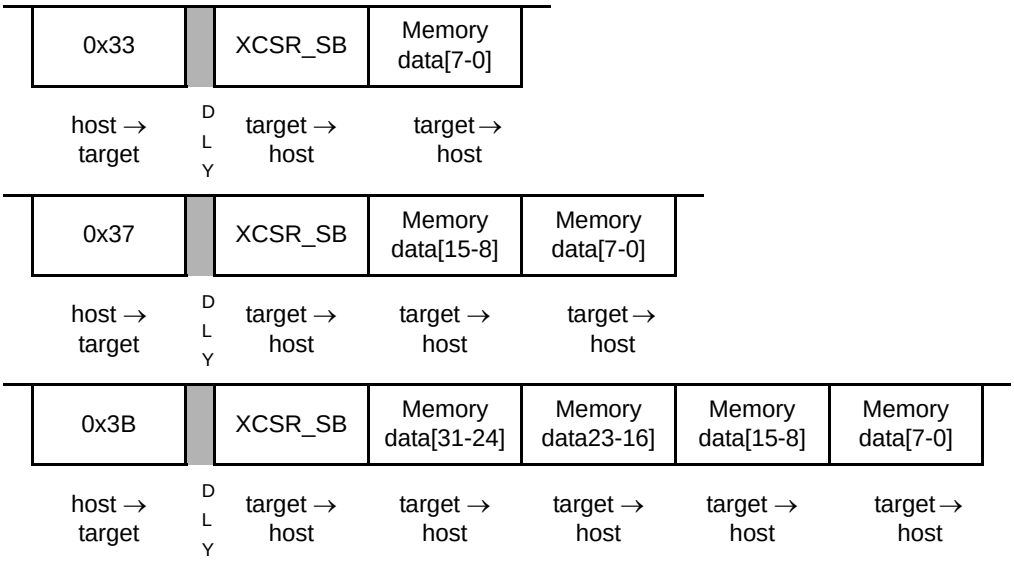


Figure 21-23.

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 contained in XCSR[31–24] (XCSR_SB) is returned before the read data. The XCSR status byte 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.

FILL_MEM.sz, FILL_MEM.sz_WS

FILL_MEM.sz

Write memory specified by debug address register, then increment address **Non-intrusive**

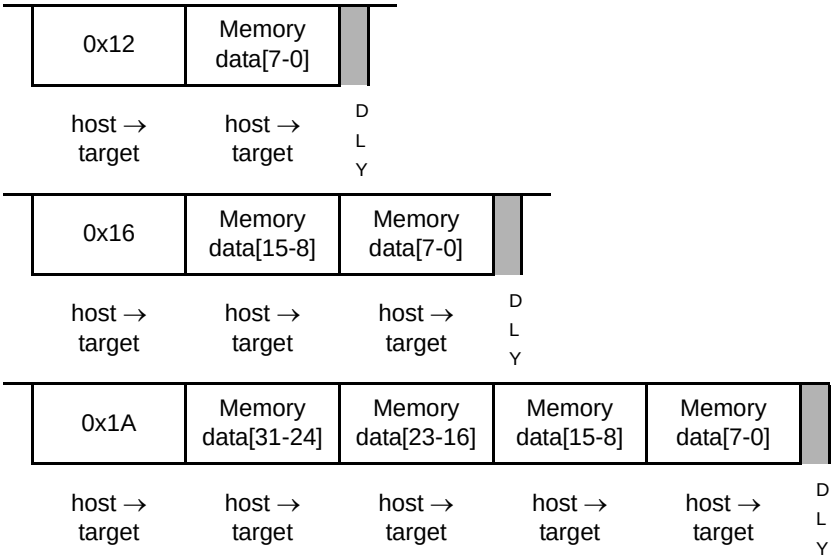


Figure 21-24.

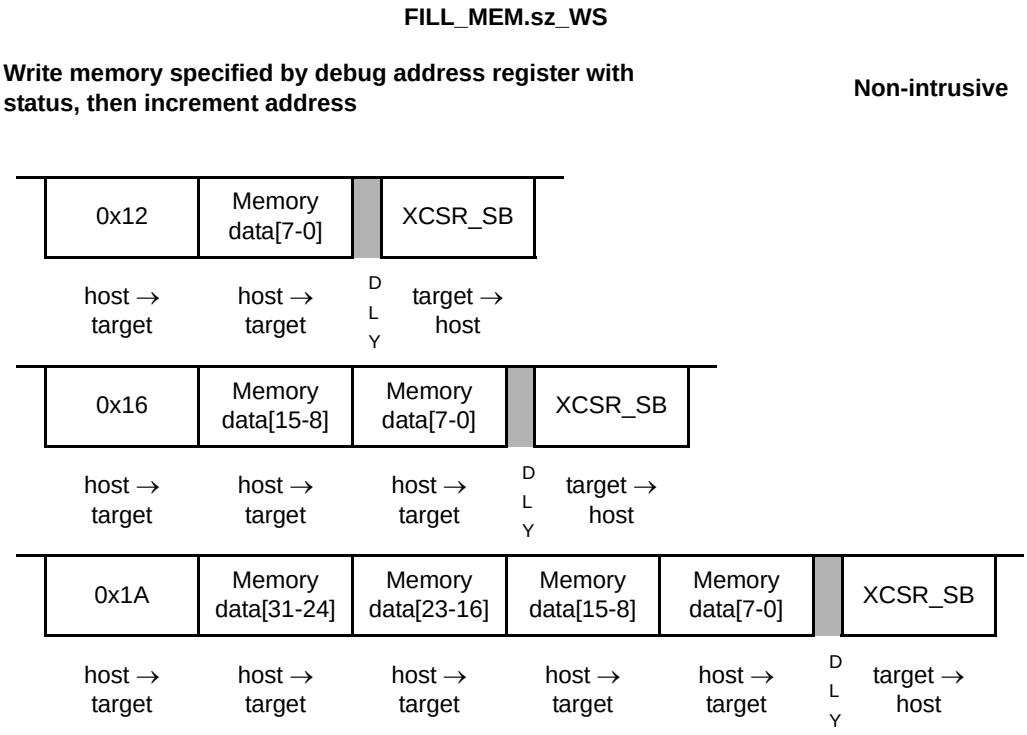


Figure 21-25.

FILL_MEM{_WS} is used with the WRITE_MEM{_WS} command to access large blocks of memory. An initial WRITE_MEM{_WS} is executed to set up the starting address of the block and write the first datum. If an initial WRITE_MEM{_WS} is not executed before the first FILL_MEM{_WS}, an illegal command response is returned. The FILL_MEM{_WS} command stores subsequent operands. The initial address is incremented by the operand size (1, 2, or 4) and saved in a temporary register. Subsequent WRITE_MEM{_WS} commands use this address, perform the memory write, increment it by the current operand size, and store the updated address in the temporary register. If the with-status option is specified, the core status byte contained in XCSR[31–24] (XCSR_SB) is returned after the write data. The XCSR status byte 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.

21.5.1.6 GO

Go **Non-intrusive**

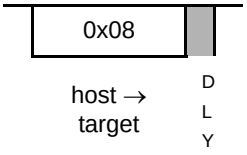


Figure 21-26.

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. Pre fetching 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 perfecting resumes. If a GO command is issued and the CPU is not halted, the command is ignored.

NOP

No operation **Non-intrusive**

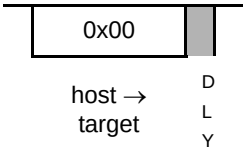


Figure 21-27.

NOP performs no operation and may be used as a null command where required.

READ_CREG

Read CPU control register **Active Background**

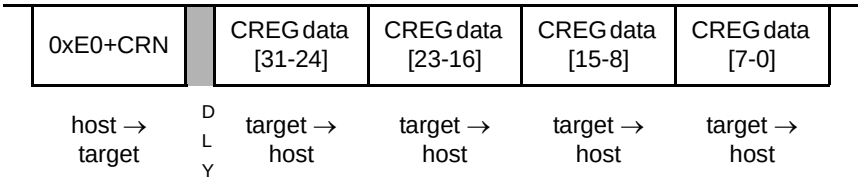


Figure 21-28.

If the processor is halted, this command reads the selected control register and returns the 32-bit result. This register grouping includes the PC, SR, CPUCR, VBR, and OTHER_A7. Accesses to processor control registers are always 32-bits wide, regardless of implemented register width. The register is addressed through the core register number (CRN). See Table 21-24 for the CRN details when CRG is 11.

If the processor is not halted, this command is rejected as an illegal operation and no operation is performed.

READ_DREG

Read debug control register **Non-intrusive**

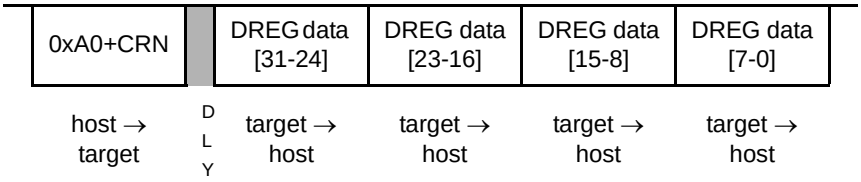


Figure 21-29.

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 21-4 for CRN details.

READ_MEM.sz, READ_MEM.sz_WS

READ_MEM.sz

Read memory at the specified address **Non-intrusive**

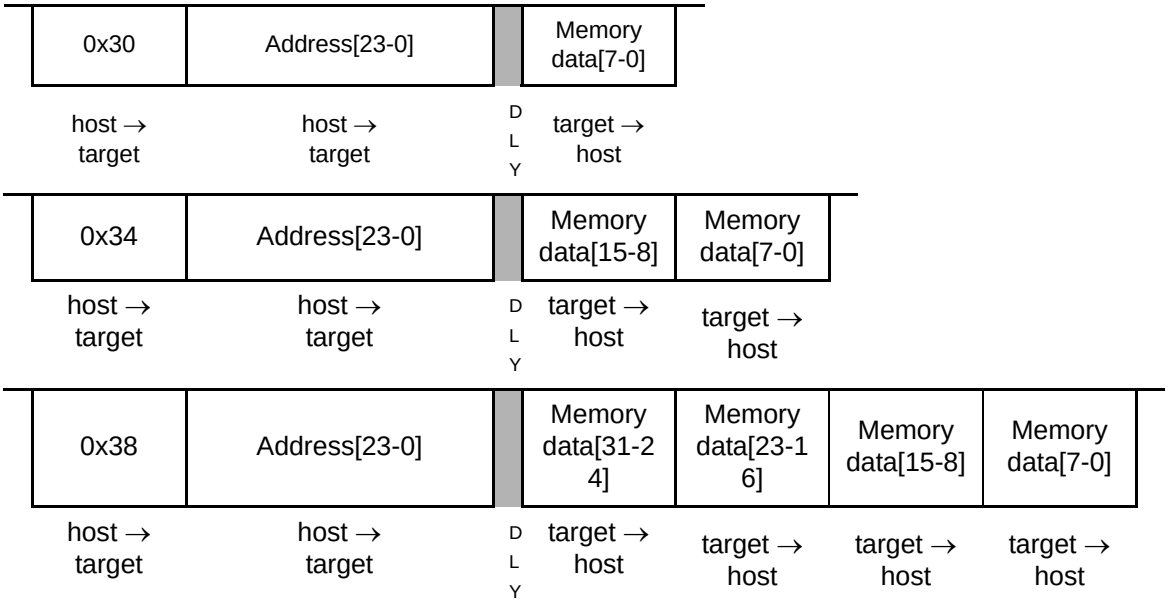


Figure 21-30.

READ_MEM.sz_WS

Read memory at the specified address with status **Non-intrusive**

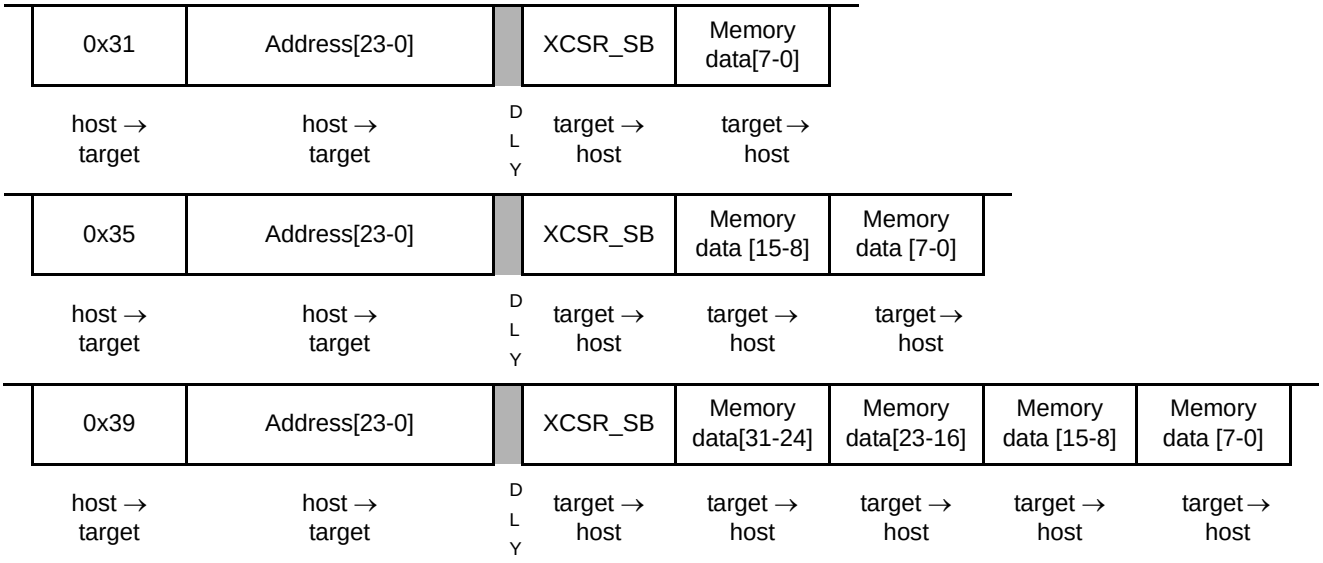


Figure 21-31.

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 contained in XCSR[31–24] (XCSR_SB) is returned before the read data. The XCSR status byte 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.

READ_Rn

Read general-purpose CPU register **Active Background**

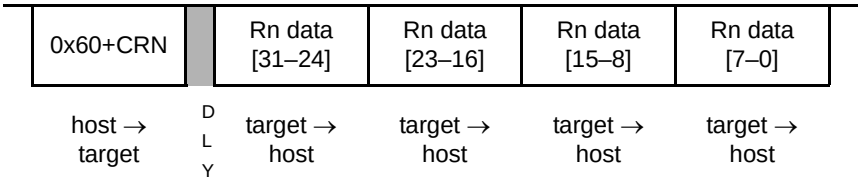


Figure 21-32.

If the processor is halted, this command reads the selected CPU general-purpose register (An, Dn) and returns the 32-bit result. See Table 21-24 for the CRN details when CRG is 01.

If the processor is not halted, this command is rejected as an illegal operation and no operation is performed.

READ_XCSR_BYTE

Read XCSR Status Byte **Always Available**

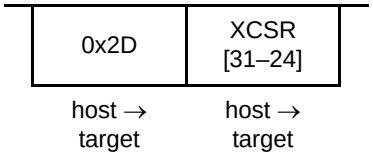


Figure 21-33.

Read the special status byte of XCSR (XCSR[31–24]). This command can be executed in any mode.

READ_CSR2_BYTE

Read CSR2 Status Byte **Always Available**

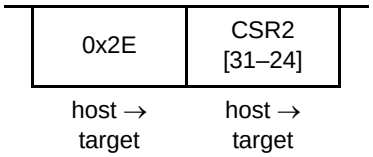


Figure 21-34.

Read the most significant byte of CSR2 (CSR2[31–24]). This command can be executed in any mode.

READ_CSR3_BYTE

Read CSR3 Status Byte **Always Available**

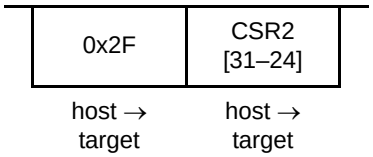


Figure 21-35.

Read the most significant byte of the CSR3 (CSR3[31–24]). This command can be executed in any mode.

WRITE_CREG

Write CPU control register **Active Background**

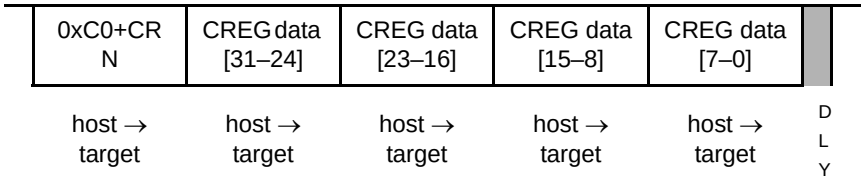


Figure 21-36.

If the processor is halted, this command writes the 32-bit operand to the selected control register. This register grouping includes the PC, SR, CPUCR, VBR, and OTHER_A7. Accesses to processor control registers are always 32-bits wide, regardless of implemented register width. The register is addressed through the core register number (CRN). See Table 21-24 for the CRN details when CRG is 11.

If the processor is not halted, this command is rejected as an illegal operation and no operation is performed.

WRITE_DREG

Write debug control register **Non-intrusive**

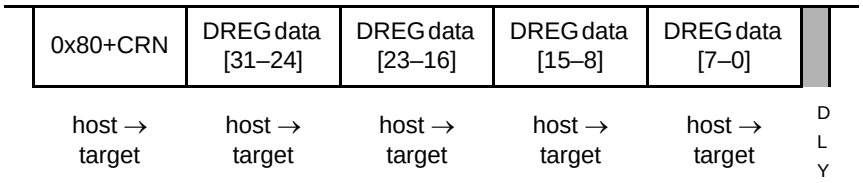


Figure 21-37.

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). See Table 21-4 for CRN details.

WRITE_MEM.sz, WRITE_MEM.sz_WS

WRITE_MEM.sz

Write memory at the specified address **Non-intrusive**

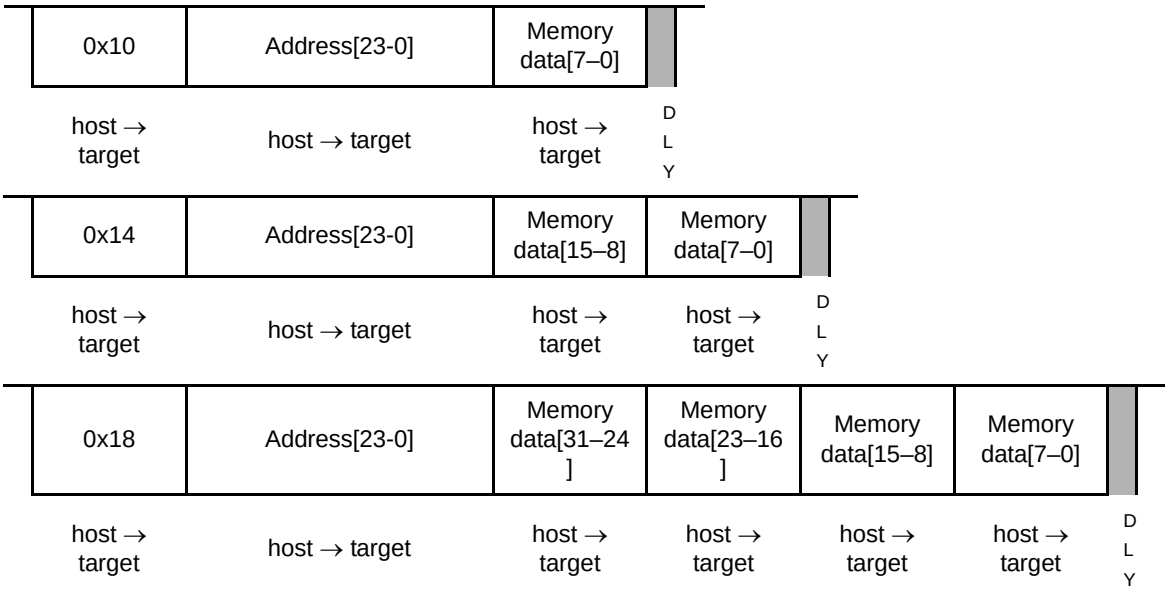


Figure 21-38.

WRITE_MEM.sz_WS

Write memory at the specified address with status **Non-intrusive**

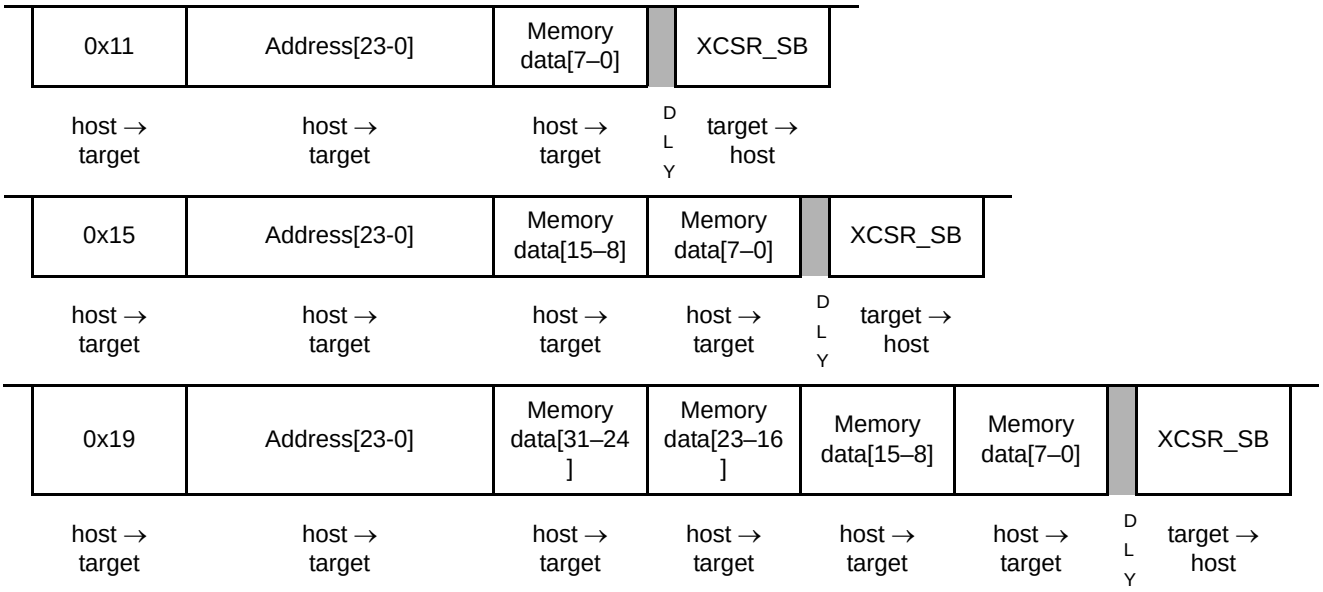


Figure 21-39.

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 contained in XCSR[31–24] (XCSR_SB) is returned before the read data. The XCSR status byte reflects the state after the memory read was performed.

The examples show the WRITE_MEM.B{_WS}, WRITE_MEM.W{_WS}, and WRITE_MEM.L{_WS} commands.

WRITE_Rn

Write general-purpose CPU register **Active Background**

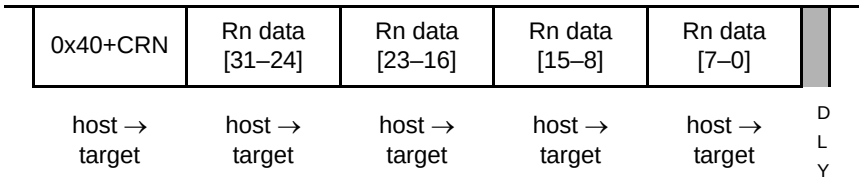


Figure 21-40.

If the processor is halted, this command writes the 32-bit operand to the selected CPU general-purpose register (An, Dn). See Table 21-24 for the CRN details when CRG is 01.

If the processor is not halted, this command is rejected as an illegal operation and no operation is performed.

WRITE_XCSR_BYTE

Write XCSR Status Byte **Always Available**

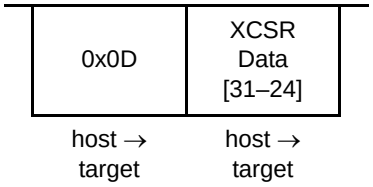


Figure 21-41.

Write the special status byte of XCSR (XCSR[31–24]). This command can be executed in any mode.

WRITE_CSR2_BYTE

Write CSR2 Status Byte **Always Available**

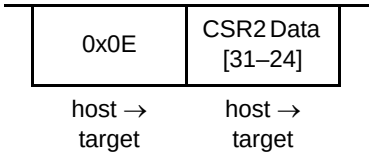


Figure 21-42.

Write the most significant byte of CSR2 (CSR2[31–24]). This command can be executed in any mode.

WRITE_CSR3_BYTE

Write CSR3 Status Byte **Always Available**

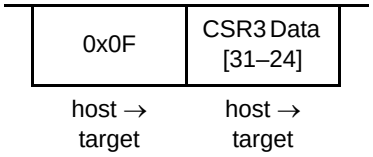


Figure 21-43.

Write the most significant byte of CSR3 (CSR3[31–24]). This command can be executed in any mode.

21.5.1.7 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 21-44. 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). 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.

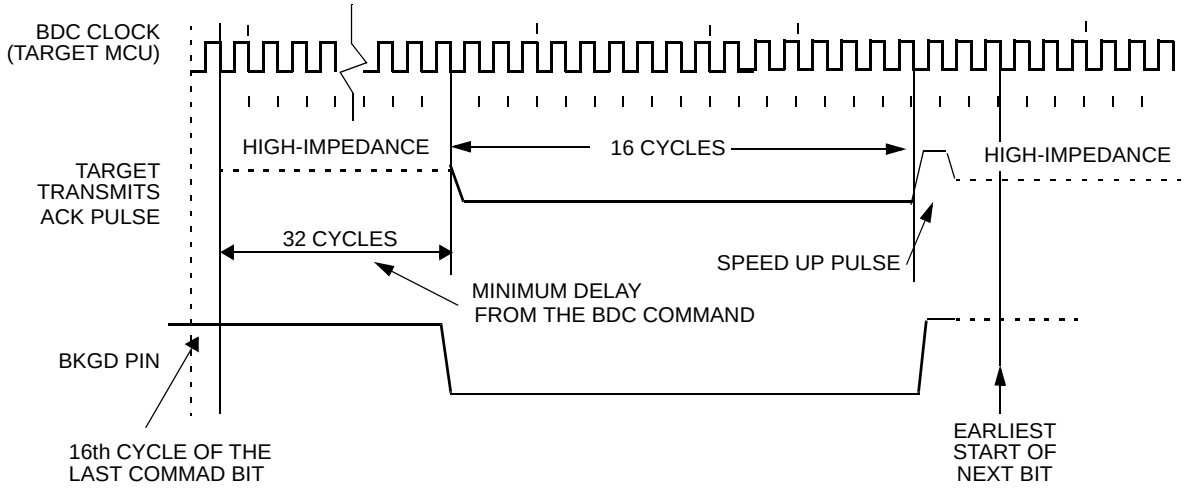


Figure 21-44. Target Acknowledge Pulse (ACK)

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 21-45 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.

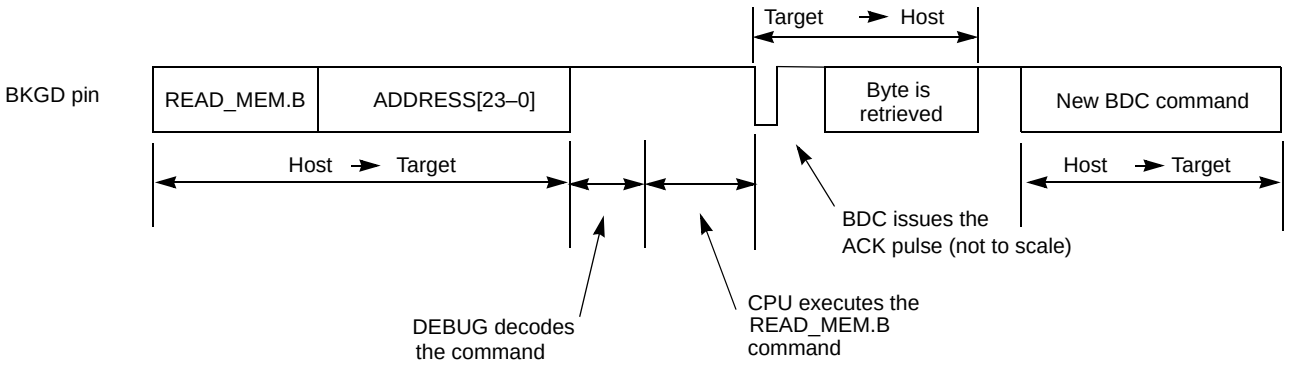


Figure 21-45. Handshake protocol at command level

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 21-45 specifies the timing when the BKGD pin is being driven, so the host should follow these timing constraints 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 time-out. 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” on page 406.

21.5.1.8 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” on page 388), 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 is much faster than the CPU clock frequency or the CPU is 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. (See the following example.)

Example 21-3. Tight loop within single aligned longword

```
  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. The recommended construct is show in the following example.

Example 21-4. Recommended construct

```
  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
```

Figure 21-46 shows a SYNC command aborting a READ_MEM.B. After the command is aborted, a new command could be issued by the host.

NOTE

Figure 21-46 signal timing is not drawn to scale.

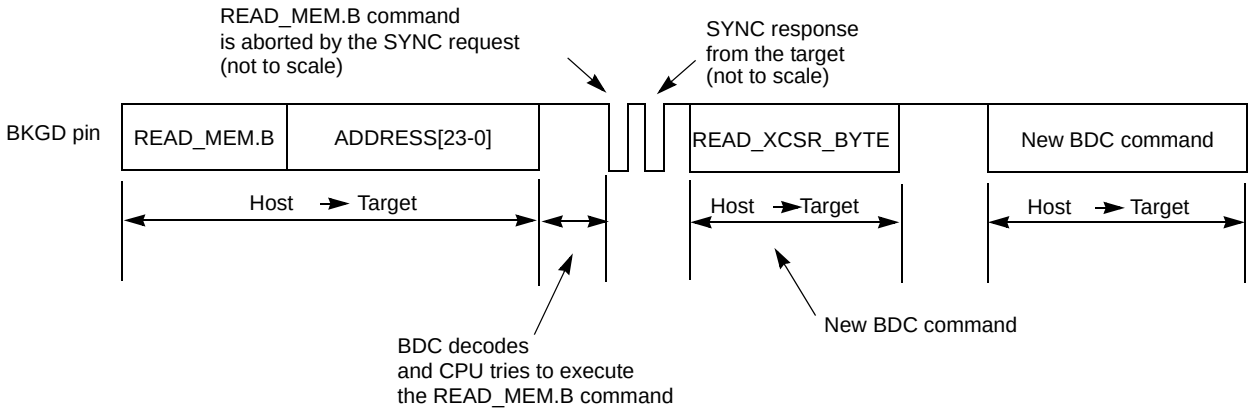


Figure 21-46. ACK abort procedure at the command level

Figure 21-47 a shows a conflict between the ACK pulse and the sync request pulse. This conflict could occur if a pod device is connected to the target BKGD pin and the target is already executing a BDC command. Consider that the target CPU is executing a pending BDC command at the exact moment the pod is being connected to the BKGD pin. In this case, an ACK pulse is issued at the same time as the SYNC command. In this case there is an electrical conflict between the ACK speedup pulse and the sync pulse. Because this is not a probable situation, the protocol does not prevent this conflict from happening.

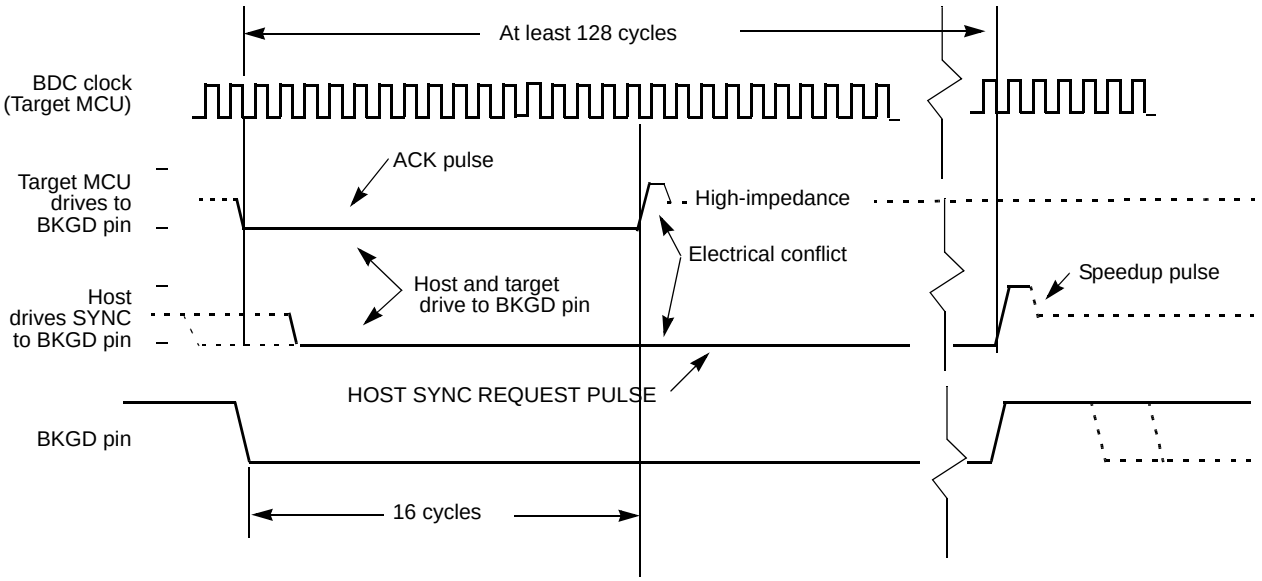


Figure 21-47. ACK pulse and SYNC request conflict

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 21-25 for the complete enumeration of this function.

An exception is the ACK_ENABLE command, which does not require CPU execution but responds with the ACK pulse. This feature can be used by the host to evaluate if the target supports the hardware handshake protocol. If an ACK pulse is issued in response to this command, the host knows that the target supports the hardware handshake protocol. If the target does not support the hardware handshake protocol the ACK pulse is not issued. In this case, the ACK_ENABLE command is ignored by the target, because it is not recognized as a valid command.

21.5.2 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.

21.5.3 Freescale-recommended BDM pinout

Typically, a relatively simple interface pod is used to translate commands from a host computer into commands for the custom serial interface to the single-wire background debug system. Depending on the development tool vendor, this interface pod may use a standard RS-232 serial port, a parallel printer port, or some other type of communications such as a universal serial bus (USB) to communicate between the host PC and the pod. The pod typically connects to the target system with ground, the BKGD pin, $\overline{\text{RESET}}$, and sometimes V_{DD}. An open-drain connection to reset allows the host to force a target system reset, useful to regain control of a lost target system or to control startup of a target system before the on-chip nonvolatile memory has been programmed. Sometimes V_{DD} can be used to allow the pod to use power from the target system to avoid the need for a separate power supply. However, if the pod is powered separately, it can be connected to a running target system without forcing a target system reset or otherwise disturbing the running application program.

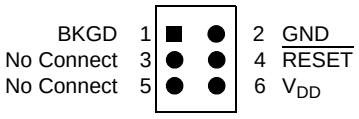


Figure 21-48. Recommended BDM connector

Appendix A
Revision History

This appendix describes corrections to the MMA955xL Intelligent, Motion-Sensing Platform Hardware Reference Manual. For convenience, the corrections are grouped by revision.

A.1 Changes Between Revisions 0 and 1

Rev. 0 of this document was published in June 2011. Rev. 1.0 of this document was published in August 2013.

Table A-1. Changes Between Revisions 0 and 1

Chapter	Description
Book-wide	• Brought register tables into compliance with corporate style • Updated headings, figure and table titles, and body content with new corporate style policies • Made minor corrections to body content and illustrations
	•
	•
	•
	•
	•
	•
	•
	•
	•
	•

Looking for pricing, stock, or lifecycle information?

Click below to explore more details on WIN SOURCE:

- View KITMMA9550LEVM on WIN SOURCE
- NXP / Nexperia Information

Optimize Your Supply Chain with WIN SOURCE Solutions

- Global Sourcing Solution
- Obsolete Management
- Cost Control Management
- Shortage Management
- Alternative Solution
- Excess Inventory Management