# MC1322x

## Advanced ZigBee™ - Compliant SoC Platform for the 2.4 GHz IEEE® 802.15.4 Standard Reference Manual

Document Number: MC1322xRM
Rev. 1.6
01/2012

ARM POWERED®

freescale™ semiconductor

***How to Reach Us:***

**Home Page:**
www.freescale.com

**E-mail:**
support@freescale.com

**USA/Europe or Locations Not Listed:**
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

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

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

**Asia/Pacific:**
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

***For Literature Requests Only*:**
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-521-6274 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

# Contents

## About This Book



## Chapter 1
## MC1322x Introduction

## Chapter 2
## Pins and Connections



## Chapter 3
## MC1322x System Considerations

## Chapter 4
## Memory



## Chapter 5
## System Management (Including CRM)

## Chapter 6
## MC1322x IEEE 802.15.4 2.4 GHz ISM Band Transceiver



## Chapter 7

## Central Processing Unit (CPU)



## Chapter 8

## Interrupt Controller (ITC)



## Chapter 9
## MAC Accelerator (MACA)

## Chapter 10

## Advanced Security Module (ASM)



## Chapter 11
## General Purpose I/O Module

## Chapter 12
## Timer Module (TMR)

## Chapter 13
## Universal Asynchronous Receiver/Transmitter Module (UART)

## Chapter 14
## I2C Module



## Chapter 15
## Serial Peripheral Interface Module (SPI)

## Chapter 16
## Synchronous Serial Interface (SSI)



## Chapter 17

## Analog to Digital Converter Module (ADC)



## Chapter 18
## Development and Debug Support

## Appendix A
## MC1322x Register Address Map

## Appendix B
## MC1322x Software Driver Utilities



## Appendix C
## Bootloader Reference

## About This Book

This manual describes the MC1322x, which is Freescale's third-generation ZigBee platform that incorporates a complete, low power, 2.4 GHz radio frequency transceiver, 32-bit ARM7 core based MCU, hardware acceleration for both the IEEE 802.15.4 MAC and AES security, and a full set of MCU peripherals into a 99-pin LGA Platform-in-Package (PiP).

## Audience

This manual is intended for system designers.

## Organization

This document is organized into 18 chapters and 3 appendices.

| | |
|---|---|
| Chapter 1 | Introduction — This chapter introduces the MC1322x features and functionality. |
| Chapter 2 | Pins and Connections — Describes device pinout and functionality. |
| Chapter 3 | System Considerations — Describes system level considerations of the MC1322x modem and MCU. |
| Chapter 4 | Memory — This chapter details the MC1322x ARM7 core which has a register-based instruction set and CPU registers that are not located in the memory map. |
| Chapter 5 | System Management — This chapter details the management of the PiP including reset, power management, wake-up from low power, clock control, and KBI interface. |
| Chapter 6 | IEEE 802.15.4 Transceiver — This chapter provides the reference for the radio and the modem. The MC1322x transceiver consists of the 2.4 GHz radio, modem, and MAC Accelerator.( MACA). |
| Chapter 7 | CPU — Describes the MC1322x ARM7TDMI-S processor which is a member of the ARM family of general-purpose 32-bit microprocessors. |
| Chapter 8 | Interrupts — Shows how interrupts provide a way for the MC1322x to inform the MCU of onboard events without requiring the MCU to constantly query MC1322x status. |
| Chapter 9 | MACA MAC Accelerator — This chapter describes the low-level MAC and PHY link controller. |
| Chapter 10 | Advanced Security Module (ASM) — The ASM block is a hardware engine that encrypts/decrypts using the Advanced Encryption Standard (AES). |
| Chapter 11 | Parallel IO (GPIO) — Details the 64 GPIOs. |
| Chapter 12 | Timer (TMR) — This chapter describes each Timer module (TMR) that contains four identical counter/timer groups. |
| Chapter 13 | Universal Asynchronous Receiver/Transmitter Module (UART) — This chapter describes the Universal Asynchronous Receiver Transmitter (UART) module. |

| | |
|---|---|
| Chapter 14 | $I^2C$ Module — The inter-IC (IIC or $I^2C$) bus is a two-wire—serial data (SDA) and serial clock (SCL)— bidirectional serial bus. |
| Chapter 15 | Serial Peripheral Interface (SPI) — This chapter describes the Serial Peripheral Interface (SPI) module for external use. |
| Chapter 16 | Synchronous Serial Interface (SSI) — This chapter describes the Synchronous Serial Interface (SSI) architecture, programming model, operating modes, and initialization. |
| Chapter 17 | Analog to Digital Converter (ADC) — The ADC module manages the interface to external sensors, scans multiple channels, and controls the warm-up of the analog portions of the analog to digital system. |
| Chapter 18 | Development Debug and Support — This chapter describes the full set of hardware/software evaluation and development tools. |
| Appendix A | MC1322x Register Address Map — Provides a single table layout of the memory map. |
| Appendix B | Software Utilities in ROM — This appendix describes the extensive set of driver utilities for the platform. |
| Appendix C | Bootloader Reference — This appendix describes the MC1322x ROM-based bootloader in more detail. |

## Revision History

The following table summarizes revisions to this document since the previous release (Rev 1.4).

**Revision History**

| Location | Revision |
|---|---|
| Chapters 5 and 13 | Corrections and updates. |

## Definitions, Acronyms, and Abbreviations

The following list defines the acronyms and abbreviations used in this document.

| | |
|---|---|
| ACK | Acknowledgement Frame |
| API | Application Programming Interface |
| BB | Baseband |
| CCA | Clear Channel Assessment |
| CRC | Cyclical Redundancy Check |
| DCD | Differential Chip Decoding |
| DME | Device Management Entity |
| FCS | Frame Check Sequence |
| FFD | Full Function Device |
| FFD-C | Full Function Device Coordinator |
| FLI | Frame Length Indicator |
| GTS | Guaranteed Time Slot |
| HW | Hardware |
| IRQ | Interrupt Request |
| ISR | Interrupt Service Routine |
| LO | Local Oscillator |
| MAC | Medium Access Control |
| MCPS | MAC Common Part Sublayer |
| MCU | Microcontroller Unit |
| MLME | MAC Sublayer Management Entity |
| MSDU | MAC Service Data Unit |
| NWK | Network |
| PA | Power Amplifier |
| PAN | Personal Area Network |
| PANID | PAN Identification |
| PHY | PHYsical Layer |
| PIB | PAN Information Base |
| PPDU | PHY Protocol Data Unit |
| PSDU | PHY Service Data Unit |
| RF | Radio Frequency |
| RFD | Reduced Function Device |
| SAP | Service Access Point |
| SFD | Start of Frame Delimiter |

| | |
|---|---|
| SPI | Serial Peripheral Interface |
| SSCS | Service Specific Convergence Layer |
| SW | Software |
| VCO | Voltage Controlled Oscillator |

## References

The following sources were referenced to produce this book:


1. Freescale MC1322x Data Sheet
2. Standard for Part 15.4: Wireless medium access control (MAC) and physical layer (PHY) specifications for low rate wireless personal area networks (WPAN). IEEE Std 802.15.4-2006, IEEE, New York, NY, 2006.

# Chapter 1
# MC1322x Introduction

The MC1322x is Freescale's third-generation ZigBee platform which incorporates a complete, low power, 2.4 GHz radio frequency transceiver, 32-bit ARM7 core based MCU, hardware acceleration for both the IEEE 802.15.4 MAC and AES security, and a full set of MCU peripherals into a 99-pin LGA Platform-in-Package (PiP).

The MC1322x family is available as two part numbers. These device types differ only in their ROM contents, all other device hardware, performance, and specifications are identical:

- MC13224V - this is the original version and is the generic part type.
  - The MC13224V is intended for most IEEE 802.15.4 applications including MAC-based, ZigBee-2007 Profile 1, and ZigBee RF4CE targets.
  - It has a more complete set of peripheral drivers in ROM.
- MC13226V - this is a more recent version and is provided specifically for ZigBee-2007 Profile 2 (Pro) applications. Only the onboard ROM image has been changed to optimize ROM usage for the ZigBee Pro profile and maximize the amount of available RAM for application use.
  - The IEEE MAC/PHY functionality has been streamlined to include only that functionality required by the ZigBee specification. The MAC functionality is 802.15.4 compatible.
  - For a typical application, up to 20 kbytes more of RAM is available versus the M13224V
  - Some drivers present in the MC13224 ROM have been removed and these include the ADC, LCDfont, and SSI drivers. These drivers are still available as library functions, but now compile into the RAM space.
  - The Low Level Component (LLC) functionality has also been streamlined for the ZigBee specification

**NOTE**

- See Section 3.11, "MC13224 and MC13226 Device Differences" for details on identifying and using the MC13244V versus the MC13226V.
- When running the Freescale IEEE 802.15.4 MAC (or a related stack) on the MC1322x platform, neither beaconing or GTS are supported.

- See the MC1322x Data Sheet for ordering information, supported software solutions, electrical specifications and mechanical information.

The RF radio interface provides for low cost and the high density as shown in Figure 1-1. An on board balun along with a TX/RX switch allows direct connection to a single-ended 50-Ω antenna. The integrated PA provides programmable output power typically from -30 dBm to +3 dBm, and the RX LNA provides -96 dBm sensitivity. In addition, separate complementary PA outputs allow use of an external PA and/or LNA for extended range applications. This solution also has on board bypass capacitors and crystal load

capacitors for the smallest footprint in the industry. All components are integrated into the package except the crystal and antenna.

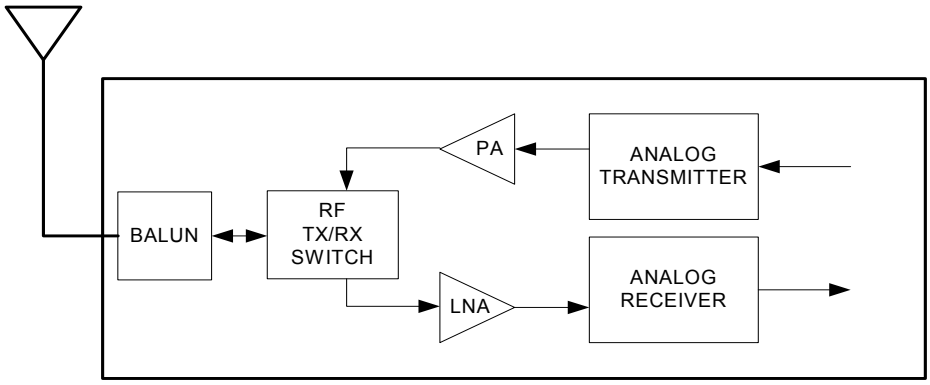



**Figure 1-1. MC1322x RF Radio Interface**

In addition to the best-in-class MCU performance and power, the MC1322x also provides best-in-class power savings. Typical transmit current is 29 mA and typical receive current is 22 with the CPU at 2 MHz operation. Onboard power supply regulation is provided for source voltages from 2.0 Vdc to 3.6 Vdc. Numerous low current modes are available to maximize battery life including sleep or restricted performance operation.

Applications include, but are not limited to, the following:

- Residential and commercial automation
  - Lighting control
  - Security
  - Access control
  - Heating, ventilation, air-conditioning (HVAC)
  - Automated meter reading (AMR)
- Industrial Control
  - Asset tracking and monitoring
  - Homeland security
  - Process management
  - Environmental monitoring and control
  - HVAC
  - Automated meter reading
- Health Care
  - Patient monitoring
  - Fitness monitoring
- Consumer
  - Remote control
  - Entertainment systems
  - Cellular phone attach

## 1.1 Features

This section provides a simplified block diagram and highlights MC1322x features.

### 1.1.1 Block Diagram

Figure 1-2 shows a simplified block diagram of the MC1322x.

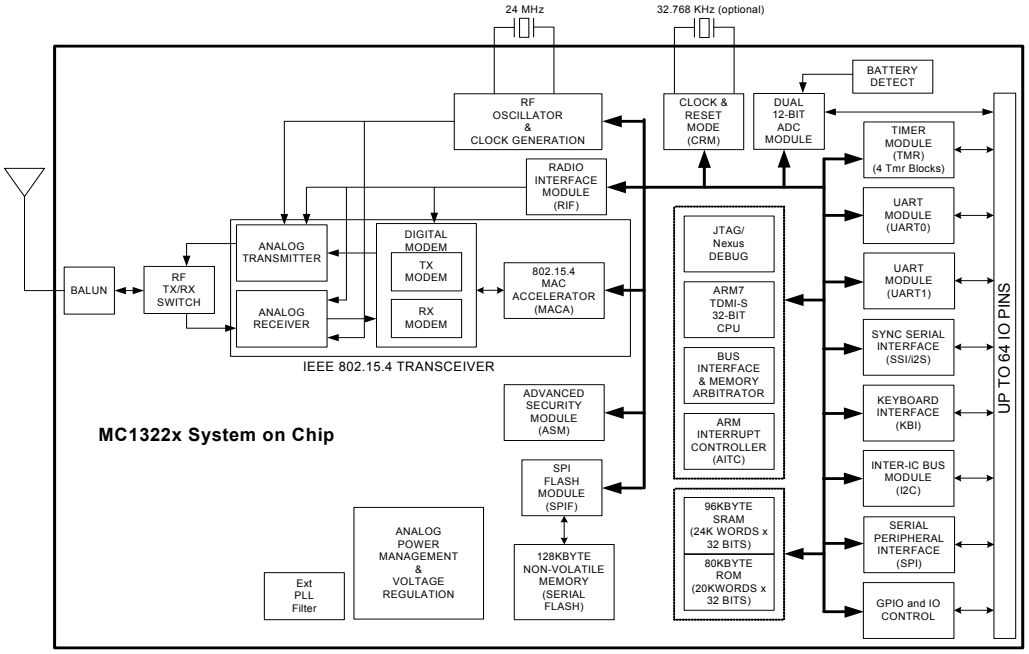



**Figure 1-2. MC1322x Simplified Block Diagram**

### 1.1.2 Features Summary

- IEEE 802.15.4 standard compliant on-chip transceiver/modem
  - 2.4GHz
  - 16 selectable channels
  - Programmable transmitter output power (-30 dBm to +3 dBm typical)
  - World-class receiver sensitivity
    - < -96 dBm typical receiver sensitivity using DCD mode (<1% PER, 20-byte packets)
    - < -100 dBm typical receiver sensitivity using NCD mode (<1% PER, 20-byte packets)
- Hardware acceleration for IEEE 802.15.4 applications
  - MAC accelerator (sequencer and DMA interface)
  - Advanced encryption/decryption hardware engine (AES 128-bit)
- Supports standard IEEE 802.15.4 signalling with 250 kbps data rate
- 32-bit ARM7TDMI-S CPU core with programmable performance up to 26 MHz (24 MHz typical)
- Extensive on-board memory resources

  - 128 Kbyte serial FLASH memory (can be mirrored into RAM)
  - 96 Kbyte SRAM
  - 80 Kbyte ROM
- Best-in-class power dissipation
  - 22 mA typical RX current draw (DCD mode) with radio and MCU active
  - 29 mA typical TX current draw with radio and MCU active (coin cell capable)
  - 3.3 mA typical current draw with MCU active (radio off)
  - 0.8 mA typical current with MCU idle (radio off)
  - 0.85 µA typical Hibernate current (retain 8 Kbyte SRAM contents)
  - 0.4 µA maximum Off current (device in reset)
- Extensive sleep mode control and variation
  - Hibernate and Doze low power modes
  - Programmable degree of power down
  - Clock management
  - Onboard 2 kHz oscillator for wake-up timer.
  - Optional 32.768 kHz crystal oscillator for accurate real-time sleep mode timing and wake-up with a possible sleep period greater than 36.4 hours
  - wake-up through programmable timer, external real-time interrupts, or ADC timer
- Extensive MCU peripherals set
  - Dedicated 802.15.4 modem/radio interface module (RIF)
  - Dedicated NVM SPI interface for managing FLASH memory
  - Two dedicated UART modules capable of 2 Mbps with CTS/RTS support
  - SPI port with programmable master and slave operation
  - 8-pin keyboard interface (KBI) supports up to a 4x4 matrix. Also, provides up to four asynchronous interrupt inputs for wake-up
  - Two 12-bit analog-to-digital converters (ADCs) share 8 input channels
  - Four independent 16-bit timers with PWM capability. These can cascade in combinations up to 64-bit operation
  - Inter-integrated circuit ($I^2C$) interface
  - Synchronous Serial Interface (SSI) with $I^2S$ and SPI capability and FIFO data buffering
  - Up to 64 programmable I/O shared by peripherals and GPIO
- Powerful in-circuit debug and FLASH programming available via on-chip JTAG and/or Nexus debug ports
- System protection features
  - Low battery detect
  - Watchdog timer (COP)
  - Sleep mode timer
- Low external component count

  - Only antenna needed for single-ended 50-Ω RF interface (balun in package)
  - Only a crystal is required for the main oscillator; programmable crystal load capacitors are on-chip
  - All bypass capacitors in package
- Supports single crystal reference clock source (typical 24 MHz crystal with 13 - 26 MHz usable) with on-chip programmable crystal load capacitance or external frequency source. Also provides onboard 2kHz oscillator for wake-up timing or an optional 32.768 kHz crystal for accurate low power timing.
- 2 V to 3.6 V operating voltage with on-chip voltage regulators.
- Optional buck converter for better battery life.
- -40 °C to +105 °C temperature range
- RoHS-compliant 9.5mm x 9.5mm x 1.2mm 99-pin LGA package

## 1.2 High Density, Low Component Count, Integrated IEEE 802.15.4 Solution

The MC1322x is more than a high performance, low power system-on-chip IEEE 802.15.4 solution. Not only are the transceiver (radio) and MCU on an SoC, the packaged solution contains a 128 Kbyte serial FLASH memory, onboard bypass capacitors for critical nodes, and RF components that present a single-ended 50-Ω interface for an external antenna. The radio is a full differential design with an on-chip transmit/receive (TX/RX) switch, and the PiP also has an onboard balun for differential to singled-ended conversion that is wired to present the single-ended interface to the application. On-chip RF matching is also provided to present the proper impedance to the antenna.

To further simplify the application, single crystal operation (typically a 24 MHz crystal) is supported for full radio and MCU operation. The load capacitance to the crystal oscillator is supplied on-chip to eliminate the need for the otherwise required external capacitors.

## 1.3 Integrated IEEE 802.15.4 Transceiver (Radio and Modem)

The MC1322x IEEE 802.15.4 fully-compliant transceiver provides a complete 2.4 GHz radio with 250 Kbps Offset-Quadrature Phase Shift Keying (O-QPSK) data in 5.0 MHz channels and full spread-spectrum encode and decode. The modem supports transmit, receive, clear channel assessment (CCA), Energy Detect (ED), and Link Quality Indication (LQI) as required by the 802.15.4 Standard.

### 1.3.1 RF Interface and Usage

The MC1322x RF interface provides for a single-ended, 50-Ω port that connects directly to an antenna. There is an onboard balun that converts the single-ended interface to a full differential, bi-directional, on-chip interface with transmit/receive switch, LNA, and complementary PA outputs. The required port impedance matching is also onboard. This combination allows for a very small footprint and a very low cost RF solution.

The MC13224V also provides a secondary set of complementary PA outputs that can be used with external RF circuitry such as a additional PA for higher TX power to the antenna. The single-ended port continues as the receive input for this circuit configuration.

The receiver demodulator includes a module called the Differential Chip Detector which has two modes of operation:

- Non-coherent Detection (NCD) with automatic frequency control (AFC)
- Non-coherent Differential Chip Detection (DCD) without AFC

The IEEE 802.15.4 standard allows a maximum clock drift of ±40 ppm (which equals ±80 ppm station-to-station). The MC13224V 802.15.4 demodulator includes two different methods of operating in the presence of such large frequency errors:

| | |
|---|---|
| NCD Mode | Provides an increased ~3.5 dB of sensitivity. However, the addition of the AFC increases the demodulator current drain about 3 mA. |
| DCD Mode | Default receive mode at lower current. |

For longer range applications where external amplification may be desired (LNA and/or PA), additional ports are provided for secondary complementary PA outputs. These can be used as a separate PA interface while the single-ended port through the balun is used as an input only. Also, four control pins and a regulated 20 mA voltage source (VREG_ANA) are provided to control external components and supply power to the PA outputs.

The RF Interface functionality can be summarized as follows:

- Programmable output power — 0 dBm nominal output power, programmable to from -30 to +4 dBm max
- Receive sensitivity (at 1% PER, 20-byte packet) -
  - — < -96 dBm (typical) DCD receive (well above IEEE 802.15.4 specification of -85 dBm)
  - — < -100 dBm (typical) NCD receive (higher current)
- Single-ended 50-Ω port — Uses integrated transmit/receive (T/R) switch, LNA, and onboard balun. Impedance matching onboard on the chip side of the balun
- Maximum flexibility — Optionally, single-ended port becomes RF input only and a separate set of full differential PA outputs are provided. Separate input and outputs allow for a variety of RF configurations including external LNA and PA for increased range
- Four programmable control signals for external components
- Regulated voltage source for PA biasing and powering external components

### 1.3.2 Modem

The modem supports the full requirement of the IEEE 802.15.4 Standard to transmit and receive data packets. In additional, the mechanism is present to measure received signal level to provide CCA, ED, and LQI as required by the 802.15.4 Standard.

## 1.4 High Performance, Low Power 32-Bit ARM7 Processor

- The ARM7TDMI-S processor is a member of the 32-bit ARM family of general-purpose 32-bit microprocessors that offers high performance with very low-power consumption
- A three stage instruction pipeline (fetch, decode, execute) increases the speed of the flow of instructions to the processor
- Data access can be 8-bit bytes, 16-bit half words, or 32-bit words. Words must be aligned to 4-byte boundaries. Half words must be aligned to 2-byte boundaries
- The ARM7TDMI-S processor supports two instruction sets, i.e., the 32-bit ARM instruction set and the 16-bit Thumb instruction set. The Thumb mode incorporates 16-bit instructions for higher code density while retaining all the benefits of a 32-bit architecture, i.e., full 32-bit registers, 32-bit operations, and 32-bit memory transfer. The use of the instruction sets can be intermixed for maximizing performance while retaining higher code density

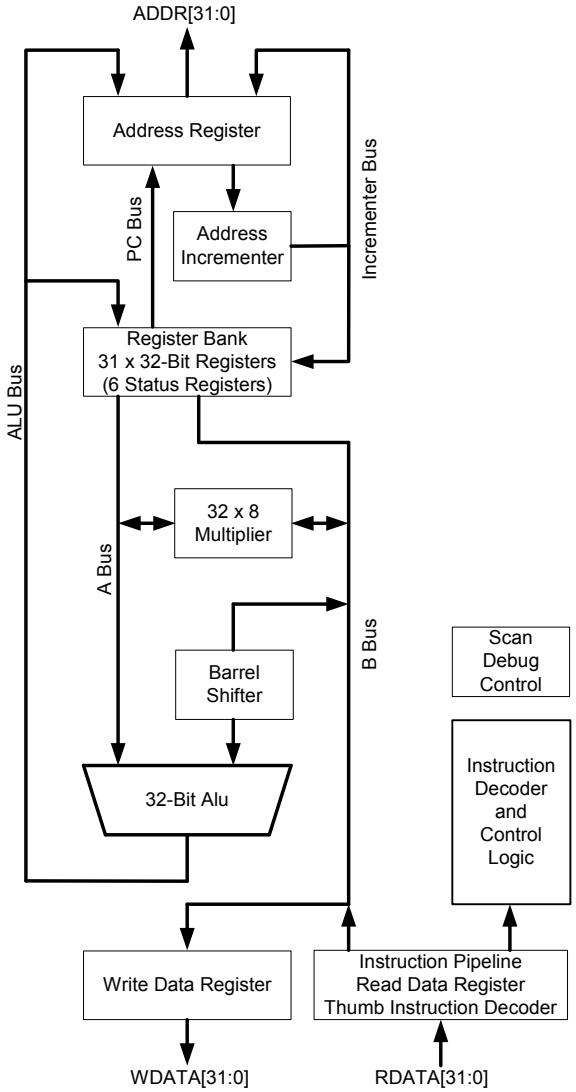



**Figure 1-3. ARM7TDMI-S 32-Bit CPU Core**

## 1.5 Low Power Operation and Power Management

The MC1322x is inherently a very low power device, but it also has extensive power management and an onboard buck regulator option to maximize battery life.

### 1.5.1 Operating Current

There are two basic low power modes of Hibernate and Doze, and both have options of how much RAM contents are retained. The primary difference between Hibernate and Doze is that Doze mode keeps the primary reference oscillator running.

**NOTE**

The user is directed to the MC1322x Data Sheet (MC1322x.pdf) for exact current specifications.

- Low Power modes
  - Off (reset active)
  - Hibernate (capable of operating current from <1 µA to about 5 µA)
  - Doze (capable of operating current of <60 µA)
- Idle (lowest run mode)
- RX and TX current only present when operation in progress

### 1.5.2 Power Management

The MC1322x power management is controlled through the Clock and Reset Module (CRM). The CRM is a dedicated module to handle MCU clock, reset, and power management functions which includes control of the power regulators. All these functions have impact on attaining lowest power.

#### 1.5.2.1 CRM Features

The CRM features include:

- Control system reset
- Control clock gating for power savings
- Sleep mode (Hibernate and Doze) management
  - Degree of chip power down
  - Retention of programmed parameters
  - Programmable retention of RAM contents
  - Clock management
- Wake-up management
  - Graceful power-up
  - Clock management
  - Wake-up via programmable timer or external interrupts.
- Wake-up timer

  - — Hibernate mode - based on onboard 2 kHz oscillator or optional 32.768 kHz crystal oscillator
  - — Doze mode - based on main reference oscillator, typically 24 MHz
- Controls reference clocks based on default 24 MHz crystal oscillator or optional 13-26 MHz oscillator with PLL for 24 MHz frequency synthesis.
- MCU watchdog timer (COP)
- Software initiated reset
- Management control of onboard linear regulators and optional buck regulator

#### 1.5.2.2 CRM Operation

The CRM has primary control of the entire system:

- Reset and power up — After release of the hardware RESETB signal, the CRM will perform a power up sequence of the MCU after the POR. The linear regulators and clock sources are managed for a graceful start-up of the MCU and its resources. The radio is not powered until needed
- Normal operation of MCU — The clock management of the MCU and its resources are controlled by the CRM. The processor clock is programmable from low frequencies up to the maximum reference frequency (13-26 MHz optional w/24 MHz standard) to allow the application to trade-off processing speed versus power savings
- Sleep modes and recovery — There are two sleep modes of Hibernate and Doze. The primary difference is that Doze mode keeps the reference oscillator running. Both modes can retain critical programmed parameters and have selectable sizes of RAM retention. Hibernate has lowest power, but Doze allows high accuracy sleep timing. The CRM manages the recovery from low power, similar to power-up from reset, providing regulator and clock management.
  - — Wake-up can be based on external interrupts through 4 KBI inputs
  - — Wake-up can be from internal interrupts
  - — Wake-up can be based on an RTI (wake-up) timer.
- The RTI timer with 2 possible frequency sources provides a very low power wake-up option from sleep
  - — One option is a onboard, low accuracy 2 kHz oscillator
  - — A second option is to add an external 32.768 kHz crystal for the RTI clock source
  - — A 32-bit timer allows greater than a 36.4 hour wake-up delay with the 32.768 crystal oscillator
- Other features of the CRM:
  - — An optional COP watchdog timer to monitor CPU program activity
  - — A programmable software reset

### 1.5.3 Optional Buck Regulator

For battery based applications, an optional buck regulator is provided to maximize battery life. Figure 1-4 shows the configuration of the buck regulator versus the normal connection. An on board MOSFET is used as a switch with an external 100µH inductor and 10µF capacitor when the buck regulator is enabled.

The buck regulator drops the higher battery voltage to 1.8 - 2.0 Vdc that is applied to the onboard linear regulators. This allows lower net current from the battery to maximize the life of the battery.

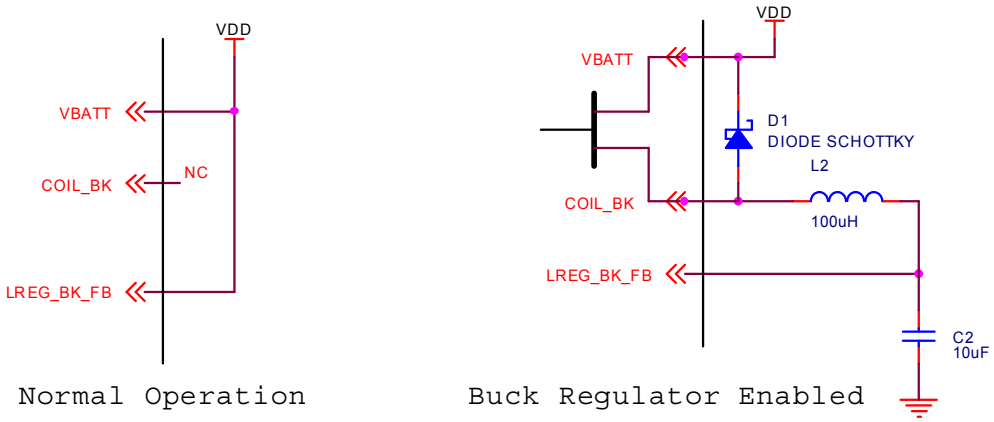



**Figure 1-4. Optional Buck Regulator**

## 1.6 Battery Voltage Monitor

An optional feature of the ADC module is battery voltage monitor capability. An onboard 1.2 Vdc reference voltage can be sampled by the ADC module. The battery-sourced supply voltage is used as the high reference voltage for the ADC and as the supply voltage lowers due to battery usage, the onboard reference voltage reading will become greater because this fixed voltage is a higher percentage of the reduced supply voltage.

Programmable high and low thresholds are provided for an ADC analog sample channel to monitor the reference voltage. This feature can be used as a trigger to provide low battery indication, protection for data that may be lost due to end-of-life for the battery, monitoring charging, and controlling buck regulator operation.

## 1.7 IEEE 802.15.4 Acceleration Hardware

The MC1322x contains two blocks that provide acceleration for IEEE 802.15.4 applications that use the 802.15.4 MAC and AES encryption/decryption.

### 1.7.1 802.15.4 MAC Accelerator (MACA) Overview

The MC1322x contains a hardware block that provides a low-level MAC and PHY link controller, which together with software running on the ARM core, implements the baseband protocols and other low-level link routine control and link control. Components of the MACA include a sequencer/controller (with

timers), TX and RX packet buffers, DMA block, frame check sequence (FCS) generator/checker, and control registers. Figure 1-5 shows a MACA simplified block diagram.

As part of the 802.15.4 protocol, packets are generated and transmitted, packets are received and verified, and channel energy is measured via a clear channel assessment (CCA). Also, combinations or sequences of events are required as part of the protocol such as an ACK response following a received packet. The MACA facilitates these activities via control of the transceiver and off loads the functions from the CPU. A dedicated DMA function moves data between the MACA buffers and RAM on a cycle steal basis and does not require intervention from the CPU.

The MACA is responsible for construction of packets for TX including FCS, and for parsing the received packets. The MACA will also handle ACKs and TxPoll sequences independent of the ARM processor. During TX the MACA will construct the entire packet. This includes preamble and SFD (start of frame delimiter). During receive, the modem will recognize preamble and SFD, then begin receiving the packet with the first bit of frame length, and finally, will check the FCS.

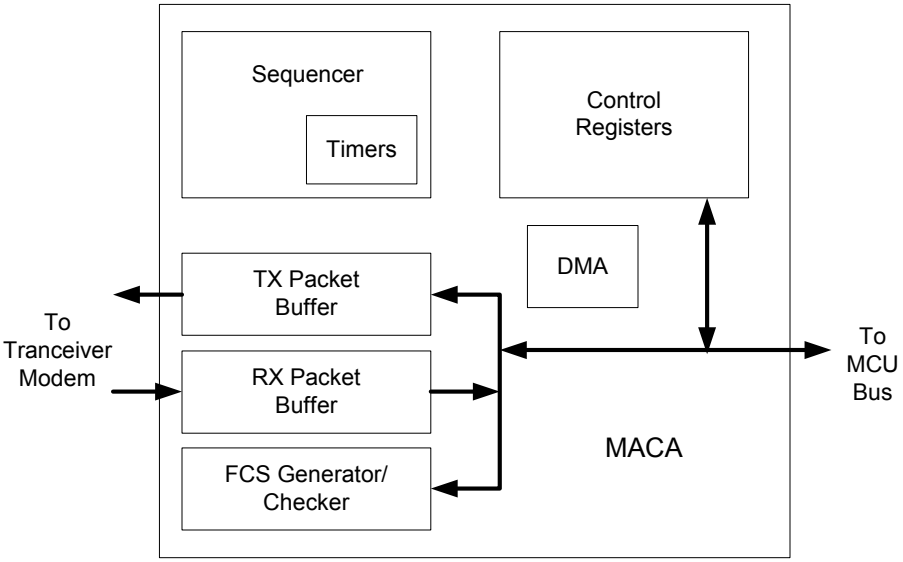



**Figure 1-5. MAC Accelerator Simplified Block Diagram**

### 1.7.1.1 MACA Features

In order to reduce CPU load, the MACA module has embedded features for controlling parts of the IEEE 802.15.4 PHY and MAC layer requirements. The MACA core features include:

- Sequence manager auto sequences
  - Automatic acknowledgment frame reception on transmitted packets
  - Automatic acknowledgment frame transmission on received packets
  - Auto-RX for continuous reception as coordinator
  - Auto sequence for transmitted MAC data.request
  - Assist for efficient response to MAC data.requests
  - Embedded channel assessment in sequence
  - Support for sequences with slotted mode access

  - — Timer triggered and immediately executed actions
  - — Support for extended RX for reception in random backoff and battery life extension
  - — Support for promiscuous mode
- Programmable auto sequence timing - Each CCA, RX, or TX event is an independent operation. The radio gets through a power-up or “warm-up” sequence for each operation (including VCO), and there is also a power-down or “warm-down” time. Sequences are combinations of radio operations and are highly configurable.
  - — RX warm-up is 72 µs
  - — TX warm-up is 92 µs
  - — Turnaround times
    - – The IEEE 802.15.4 Standard requires a TX-to-RX or a RX-to-TX turnaround time to be less than or equal to 12 symbols times (192 µs).
    - – Best practice for maximum station-to-station performance is to minimize TX-to-RX turnaround time and to maximize (within spec) RX-to-TX turnaround time.
    - – Auto sequences should use recommended turnaround times of:
      a) 11 symbols times (176 µs) RX-to-TX
      b) 96 µs TX-to-RX.
- Dedicated DMA for transfer of TX/RX data from/to RAM
- Maskable, event-driven interrupt generation
- Address header filtering for received packets. A promiscuous mode allows bypass of the filtering for monitoring network traffic
- Packet manager
  - — Handles preamble data
  - — Handles frame check sequence (FCS) a.k.a CRC
  - — Embedded header filter for received packets
- Control/status registers mapped into CPU memory map
- 32-Bit random number generator — Runs at the bus clock rate, a 32-bit Linear Feedback Shift Register (LFSR) can be set with a seed value and uses a 32-bit primitive polynomial. A 32-bit random number is fetched with every read of the proper control register

## 1.8 Advanced Security Module (ASM)

The IEEE 802.15.4 Standard and the ZigBee Standard both provide for optional use of data encryption.The ASM engine is a hardware block that accelerates encryption/decryption using the Advanced Encryption Standard (AES). The engine can perform “Counter” (CTR) and Cipher Block Chaining (CBC) encryption. The combination of these two modes of encryption are known as CCM mode encryption. CCM is short for Counter with CBC-MAC. CCM is a generic authenticate and encrypt block cipher mode. CCM is only defined for use with 128 bit block ciphers, such as AES. The definition of CCM mode encryption is documented in the NIST publication SP800-38C.

The ASM has the following features:

- 32-Bit wide bus interface
- CTR encryption in 13 clock cycles
- CBC encryption in 13 clock cycles
- Encrypts 128 bits as a unit
- The 128-bit registers are aligned on quad word boundaries (16 byte)
- Self-test mode
- Maskable “action complete” interrupt

## 1.9 Memory

The MC1322x memory resources consist of RAM, ROM, and serial FLASH.

### 1.9.1 RAM and ROM

The static RAM and ROM resources are located on the primary chip. Features include:

- 96 Kbytes RAM.
  - RAM0: 8 Kbytes, 2 Kwords (2048 x 32 bits)
  - RAM1: 24 Kbytes, 6 Kwords (6144 x 32 bits)
  - RAM2: 32 Kbytes, 8 Kwords (8192 x 32 bits)
  - RAM3: 32 Kbytes, 8 Kwords (8192 x 32 bits)
- All read or write accesses require a minimum of two system clock cycles
- Stall signal generated for read after write cycles
- Clock is enabled only on the accessed memory device for low power consumption
- RAMs have been divided to allow for power savings. While sleeping, combinations of the 4 RAMs can be turned off and the RAM remainder can be placed in a low voltage mode for data retention. If more RAMs are turned on, then less battery life will be achieved. Depending on the amount of RAM powered during sleep, the boot time may be longer with less RAM as the non-powered RAM must be reloaded from FLASH
- 80 Kbytes ROM
  - 20 Kwords (20480 x 32 bits)

— Initially contains bootstrap code, 802.15.4 MAC and drivers. The MAC software builds on the lower level hardware capability of the transceiver and MACA. All code except the bootstrap is “patchable”

### 1.9.2 Serial FLASH (NVM)

The MC1322x also contains a 128 Kbyte serial FLASH memory that can be mirrored into the 96 Kbyte RAM. The serial FLASH is accessed via a dedicated SPI module designated as the SPIF. The FLASH erase, program, and read capability are programmed through the SPIF port. The FLASH is accessed at boot time to load/initialize RAM. All actual CPU program and data access is typically from RAM or ROM.

## 1.10 MCU Peripherals

The MC1322x has a rich set of MCU peripherals. Figure 1-6 shows the peripheral modules.

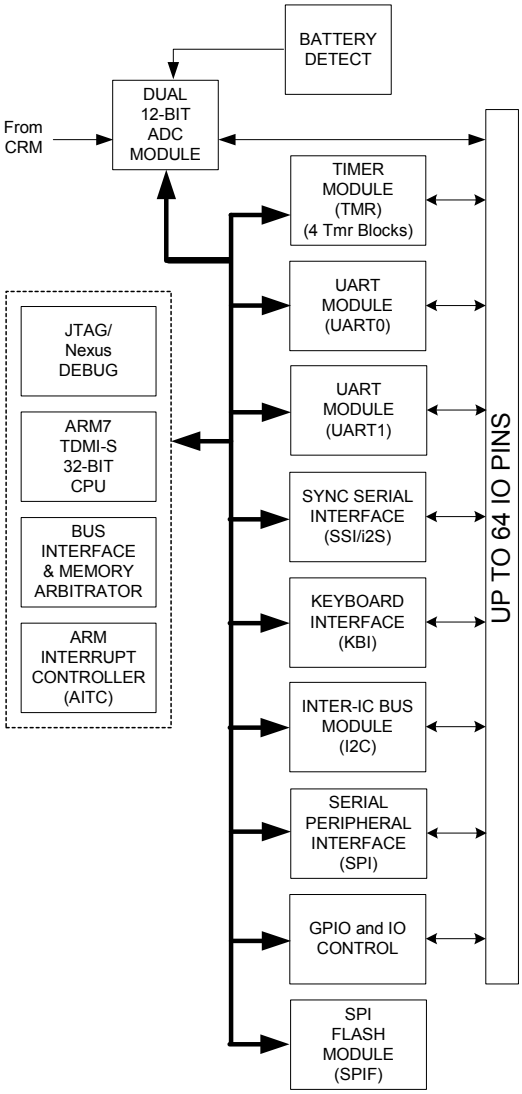



**Figure 1-6. MCU Peripherals**

## 1.11 Parallel IO (GPIO)

The parallel I/O features include:

- A total of 64 general-purpose I/O pins
- Individual control (direction and output state) for each pin when in GPIO mode
- Pad hysteresis enables
- Software-controlled pull-ups/pull-downs on each input pin
- When not used as GPIO, the IO provide alternative functions
  - Debug ports for JTAG (four signals) and Nexus (fourteen signals) modules
  - Four control signals for external RF components such as an LNA, PA, and antenna switch
  - Eight analog inputs for ADC input channels
  - Four signals for ADC reference voltages
  - Eight signals for UART1 and UART2
  - Two $I^2C$ signals
  - Four timer block signals
  - Four SPI block signals
  - Four SSI block signals
  - Eight KBI signals
- Eight KBI pins are kept alive during Hibernate or Doze. Four KBI are output and four are inputs. The input can be used as wake-up interrupts

## 1.12 Keyboard Interface (KBI) Interrupts

The MC1322x designates 8 pins (KBI_0 to KBI_7) as typically used for a keyboard interface, where four of these signals are outputs and four are inputs (KBI_4 to KBI_7) that support asynchronous interrupts for wake-up (during sleep modes). These 8 pins can be used as a matrix interface to support up to 16 switches or buttons, such as a keypad. These signals can also be used as general purpose IO if a keyboard is not present.

During Hibernate or Doze, the KBI are unique in that they are kept alive. Four KBI are outputs and four KBI are inputs. The inputs can be enabled as asynchronous interrupts to wake-up the MC13224V from the sleep mode.

## 1.13 Timer (TMR) Module

The MC1322x provides a timer module (TMR) that contains four identical counter/timer groups. Each group is capable of many variants of input capture, output compare and pulse-width modulation. The wide range of operational modes is useful for many control and sensor applications.

Figure 1-7 shows a block diagram of an individual timer group.

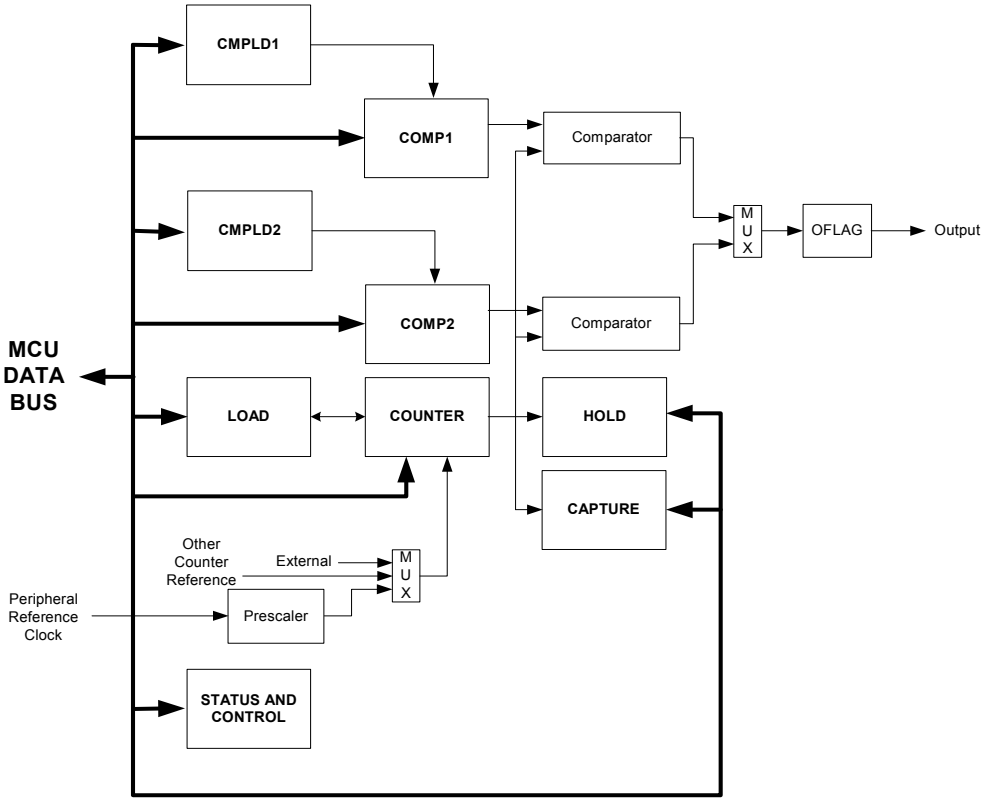



**Figure 1-7. Timer Group Block Diagram**

Each 16-bit counter/timer group contains a prescaler, a counter, a load register, a hold register, a capture register, two compare registers, and status and control registers.

- Load Register — Provides the initialization value to the counter when the counter's terminal value has been reached
- Hold Register — Captures the counter's value when other counters are being read. This feature supports the reading of cascaded counters
- Capture Register — Enables an external signal to take a snap shot of the counter's current value
- COMP1 and COMP2 Registers — Provides the values to which the counter is compared. If a match occurs, the OFLAG signal can be set, cleared, or toggled. At match time, an interrupt is generated (if enabled), and the new compare value is loaded into the COMP1 or COMP2 registers from CMPLD1 and CMPLD2 if enabled
- The Prescaler provides different time bases useful for clocking the counter/timer
- The Counter provides the ability to count internal or external events

- Control and Status Registers — Provides operational mode control of the counter, status, clock source control, interrupt control, and external interface control

Four GPIO pins (TMR0 -TMR3) are programmable and can be used with any counter/timer group.

The TMR module feature include:

- Four - 16 bit counters/timers groups
- Up/down count
- Counters can be cascaded for up to 64-bit delay counter
- Programmable count modulo.
- Peripheral reference clock is same as bus clock
- External clock max count rate equals peripheral clock divided by 2
- Internal clock max count rate equals peripheral clock.
- Count once or repeatedly
- Counters can be preloaded
- Compare registers can be preloaded
- Counters share available four GPIO pins (programmable as inputs or outputs and programmable for falling or rising edge)
- Separate prescaler for each counter
- Each counter has capture and compare capability
- Optional input glitch filter
- Functional modes include stop, count, edge-count, gated-count, quadrature-count, signed-count, triggered-count, one-shot, cascade-count, pulse-output, fixed frequency PWM, and variable-frequency PWM

## 1.14 UART Modules

The MC1322x has two universal asynchronous receiver/transmitter (UART) modules. Each UART has an independent fractional divider, baud rate generator that is clocked by the peripheral bus clock (typically 24 MHz) which enables a broad range of baud rates up to 1,843.2 Kbaud. Transmit and receive use a common baud rate for each module.

Each UART provides the following features:

- 8-Bit only data
- One or two stop bits
- Programmable parity (even, odd, and none)
- Full duplex four-wire serial interface (RXD, TXD, RTS, and CTS)
- Hardware flow control support for RTS and CTS signals
- 32-byte receive FIFO and 32-byte transmit FIFO
- Programmable sense for RTS/CTS pins (high true/low true)
- Status flags for various flow control and FIFO states
- Receiver detects framing errors, start bit error, break characters, parity errors, and overrun errors.
- Voting logic for improved noise immunity (16X/8X oversampling)
- Maskable interrupt request
- Time-out counter, which times out after eight non-present characters
- Receiver and transmitter enable/disable
- Low-power modes
- Baud rate generator to provide any multiple-of-2 baud rate between 1.2 kbaud and 1,843.2 kbaud

## 1.15 Inter-Integrated Circuit ($I^2C$) Module

The MC1322x provides an Inter-Integrated Circuit ($I^2C$) module for the $I^2C$ which is a two-wire, serial data (SDA) and serial clock (SCL), bidirectional serial bus. The $I^2C$ allows for data exchange between the MC1322x and other devices such as MCUs, serial EEPROM, serial ADC and DAC devices, and LCDs. The $I^2C$ minimizes interconnections between devices and is a synchronous, multi-master bus that allows additional devices to be connected and still handle system expansion and development. The bus includes collision detection and arbitration to prevent data corruption if two or more masters attempt to simultaneously control the $I^2C$.

The $I^2C$ module is driven by the peripheral bus clock (typically 24 MHz) and the SCL bit clock is generated from a prescaler. The prescaler divide ratio can be programmed from 61,440 to 160 (decimal) which gives a maximum bit clock of 150 kbps.

The $I^2C$ module supports the following features:

- Two-wire (SDA and SCL) interface
- Multi-master operation
- Master or slave mode
- Arbitration lost interrupt with automatic mode switching from master to slave
- Calling address identification interrupt
- START and STOP signal generation/detection
- Acknowledge bit generation/detection
- Bus busy detection
- Software-programmable bit clock frequency up to 150 kbps
- Software-selectable acknowledge bit
- On-chip filtering for spikes on the bus

## 1.16 Serial Peripheral Interface (SPI) Modules

The MC1322x has two SPI modules that use a common architecture

### 1.16.1 External SPI Module

The MC1322x offers a dedicated Serial Peripheral Interface (SPI) module for external use. The SPI is a high-speed synchronous serial data input/output port used for interfacing with serial memories, peripheral devices, or other processors. The SPI allows a serial bit stream of a programmed length (1 to 32 bits) to be shifted simultaneously into and out of the device at a programmed bit-transfer rate (called 4-wire mode). There are four pins associated with the SPI port (SPI_SCK, SPI_MOSI, SPI_MISO, and SPI_SS).

The SPI module can be programmed for master or slave operation. It also supports a 3-wire mode where for master mode the MOSI becomes MOMI, a bidirectional data pin, and for slave mode the MISO becomes SISO, a bidirectional data pin. In 3-wire mode, data is only transferred in one direction at a time.

The SPI bit clock is derived from the peripheral reference clock (typically 24 MHz with a maximum of 26 MHz). A prescaler divides the peripheral reference clock with a programmed divide ratio from 2 to 256. Typical bit clock range will be from 12 MHz to 93.75 kHz.

The SPI has the following features:

- Master or slave mode operation
- Transfer length programmable from 1 to 32 bits
- MSB-first shifting
- Data buffer is 4 bytes (32 bits) in length (not double buffered)
- Programmable transmit bit rate (typically 12 MHz max)
- Serial clock phase and polarity options
- Full-duplex (4-wire) or bidirectional data (3-wire) operation
- SPI transaction can be polled or interrupt driven
- Slave select output
- Low Power (SPI Master uses gated clocks. SPI Slave clock derived completely from SPI_SCK.)
- SPI Slave does not need system clock to be active

### 1.16.2 SPI FLASH Module (SPIF)

The SPIF is an internal SPI block dedicated to control, reading, and writing of the serial FLASH memory (NVM). It uses the same architecture as the general SPI block, but will be limited by the characteristics of the FLASH SPI interface.

## 1.17 Synchronous Serial Interface (SSI) Module

The MC1322x provides a versatile Synchronous Serial Interface (SSI) which is a full-duplex, serial port that allows communication with a variety of serial devices. These serial devices can be standard CODer-DECoder (CODECs), digital signal processors (DSPs), MCUs, peripherals, and popular industry audio CODECs that implement the Inter-Integrated Circuit sound bus standard (I2S).

The SSI typically transfers samples in a periodic manner and it consists of independent transmitter and receiver sections with common clock generation and frame synchronization. The external signals include the bit clock (SSI_BITCK), frame sync (SSI_FSYN), RX data (SSI_RX), and TX data (SSI_TX). The SSI has the following basic operating modes all with synchronous protocol:

- Normal mode — The simplest SSI mode transfers data in one time slot per frame
- Network mode — Creates a Time Division Multiplexed (TDM) network, such as a TDM CODEC network or a network of DSPs
- Gated Clock mode — Hooks up to SPI-type interfaces on MCUs or external peripheral chips in addition to providing communication

With its multi-modes, the SSI can be programmed for two very useful functions:

- A second SPI port augmenting the MC1322x SPI module
- I2S interface - the SSI is capable of generating the required clock frequencies and data format to drive a serial stereo audio DAC

The SSI includes the following features:

- Synchronous transmit and receive sections with shared internal/external clocks and frame syncs, operating in Master or Slave mode.
- Normal mode operation using frame sync
- Network mode operation allowing multiple devices to share the port with as many as 32 time slots
- Gated Clock mode operation requiring no frame sync
- SSI clock source is Peripheral Clock (typically 24 MHz); maximum SSI transfer rate is 6.0 MHz
- Separate Transmit and Receive FIFOs. Each of which is 8x24 bits
- Programmable data interface modes including $I^2S$, LSB, MSB aligned
- Programmable word length (8, 10, 12, 16, 18, 20, 22 or 24 bits)
- Program options for frame sync and clock generation
- Programmable $I^2S$ modes (Master, Slave)
- Programmable internal clock divider
- Time Slot Mask Registers for reduced CPU overhead (for Tx and Rx both)
- SSI power-down feature

## 1.18 Analog-to-Digital Converter (ADC) Module

The MC1322x ADC module provides two 12-bit (8-9 ENOB) analog-to-digital converters (ADC1 and ADC2) with 8 external channels (ADC7 - ADC0) that can be multiplexed to either ADC. ADC1 can also sample the battery voltage for monitoring purposes. External pins (ADC2_VREFH, ADC2_VREFL, ADC1_VREFH, and ADC1_VREFL) are provided for independent ADC reference voltages. The minimum sample time is 20 us. Figure 1-8 shows a block diagram of the ADC module.

Each ADC can be programmed to scan multiple selected channels on a timed basis. The primary clock to the ADC module is the peripheral reference clock (typically 24 MHz). For the time period between scan sequences, the primary clock is first divided by an 8-bit prescale (1-255), and the derived clock drives both the 32-bit delay timer and the ADC sequencer. Each ADC has its own delay timer and sequencer.

Once a scan sequence has been initiated, all selected channels can be sampled. Registers are provided to define thresholds that can be enabled for the sampled channels. A threshold can be assigned to a specific channel and can be programmed to be a less-than or greater-than threshold. Multiple thresholds can be assigned to a single channel. Warm-up of the analog portion of the circuitry is provided (for power management), and a separate 300 kHz ADC clock must be programmed via its own divider.

The battery monitor has two (2) dedicated threshold registers to set the high and low limits of the battery sample channel.

Sample values are stored in a 8x16-bit FIFO. The FIFO accumulates samples from both ADCs, and the 12-bit sample value and a 4-bit channel tag are saved for each sample. The FIFO is read by the CPU from a register address.

The module can be programmed to interrupt the processor based on the timed sample activity. Sample activity, sequencer activity, or FIFO “fullness” can all be enabled to generate an interrupt.

The ADCs can also be overridden to sample on command as opposed to sequencer, time-based activity.

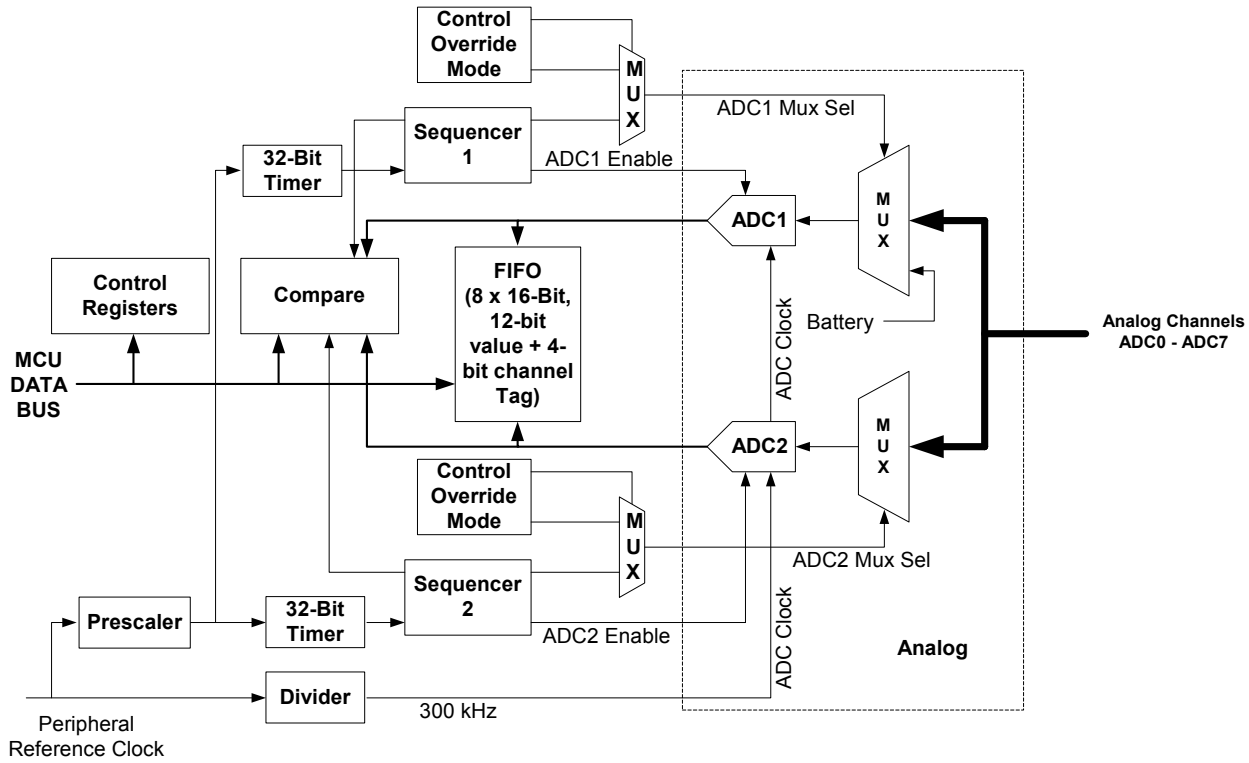


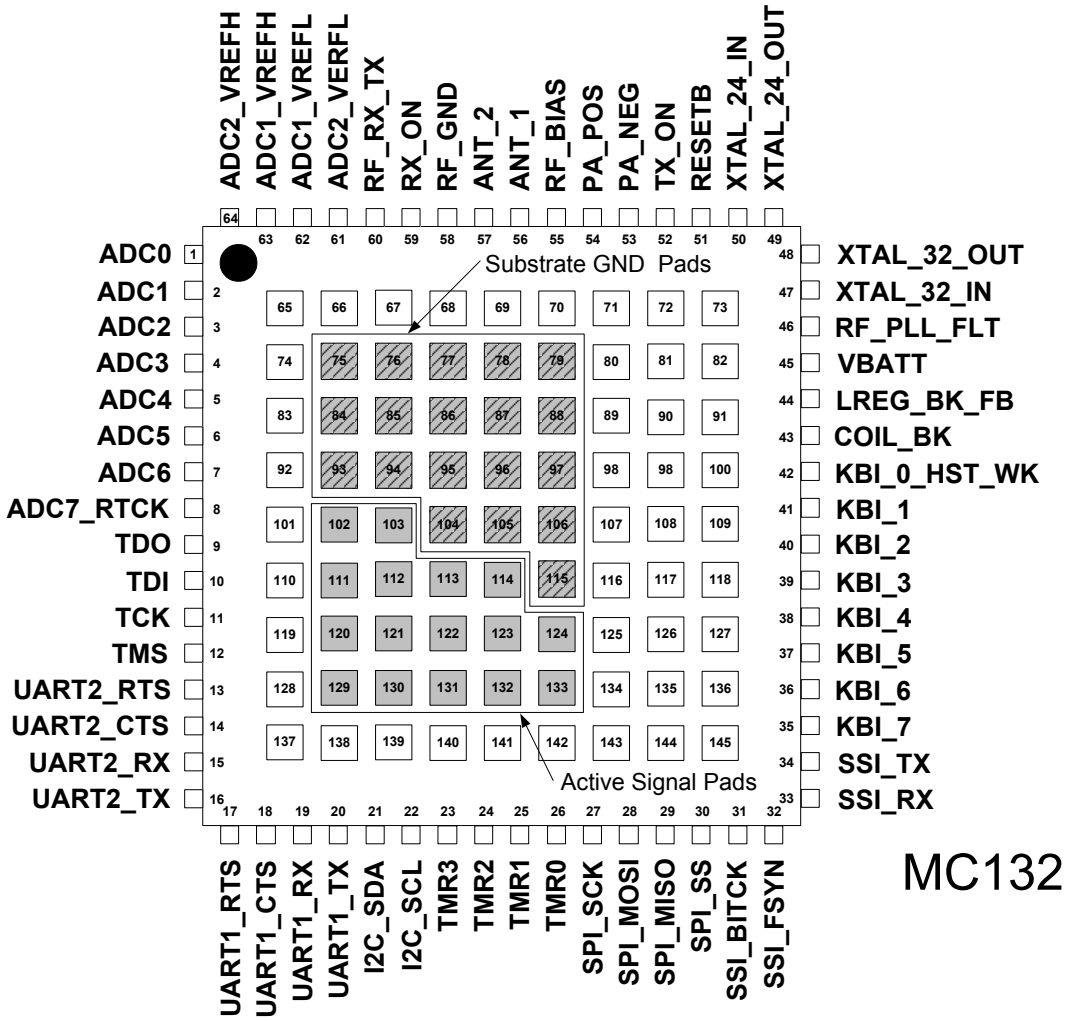

**Figure 1-8. ADC Module Block Diagram**

The ADC module has the following features:

- 12 bit resolution. Effective number of bits 8-9
- Valid usable input voltage range: [Vref_high-0.2V] to [Vref_low+0.2V]
- Maximum input range: VBATT to VSS
- Minimum sample time 20 µs
- Peripheral Clock (set by CRM) uses an 8-bit prescaler to provide the time base for the module
- Two independent channels, each with a 32-bit timer
- ADC1 has 9 channels: 8 external analog inputs plus battery reference voltage
- ADC2 has 8 channels: 8 external analog inputs
- Active channels for each ADC are programmable
- Eight active monitors plus battery reference monitors can generate a IRQ
- An 8-deep FIFO for recording data
- IRQs can be generated by the channel compare values, FIFO status, and sequencers

# Chapter 2
# Pins and Connections

## 2.1 Pin Assignments and Connections




**Notes:**

1. **Bottom pads 75-79, 84-88, 93-97, 104-106, and 115 are Substrate Ground.**
2. **Bottom pads 102-103, 111-114, 120-124, and 129-133 are active pads.**
3. **All remaining bottom pads are isolated from ground (NC), and are provided here for mechanical strength.**
4. **Figure 15 (Mechanical Diagram), is the bottom view, not the top view as shown here.**

**Figure 2-1. MC1322x Pinout (Top View; bottom pads shown)**

## 2.2 Pin Definitions

Table 2-1 details the MC1322x pinout and functionality.

**Table 2-1. Pin Function Description**

| Pin # | Pin Name | Type | Description[1] | Functionality |
|---|---|---|---|---|
| 1 | ADC0 | Analog Input or Digital Input/Output | ADC analog input Channel 0 / GPIO30 | ADC sample channel can be used by either ADC1 or ADC2. |
| 2 | ADC1 | Analog Input or Digital Input/Output | ADC analog input Channel 1/ GPIO31 | ADC sample channel can be used by either ADC1 or ADC2. |
| 3 | ADC2 | Analog Input or Digital Input/Output | ADC analog input Channel 2/ GPIO32 | ADC sample channel can be used by either ADC1 or ADC2. |
| 4 | ADC3 | Analog Input or Digital Input/Output | ADC analog input Channel 3/ GPIO33 | ADC sample channel can be used by either ADC1 or ADC2. |
| 5 | ADC4 | Analog Input or Digital Input/Output | ADC analog input Channel 4/ GPIO34 | ADC sample channel can be used by either ADC1 or ADC2. |
| 6 | ADC5 | Analog Input or Digital Input/Output | ADC analog input Channel 5/ GPIO35 | ADC sample channel can be used by either ADC1 or ADC2. |
| 7 | ADC6 | Analog Input or Digital Input/Output | ADC analog input Channel 6/ GPIO36 | ADC sample channel can be used by either ADC1 or ADC2. |
| 8 | ADC7_RTCK | Analog Input or Digital Input/Output | ADC analog input Channel 7 / ReTurn ClocK / GPIO37 | ADC sample channel can be used by either ADC1 or ADC2. Alternately, the signal returns TCK for JTAG to support adaptive clocking. |
| 9 | TDO | Digital Input/Output | JTAG Test Data Output / GPIO49 | JTAG debug port serial data output. |
| 10 | TDI | Digital Input/Output | JTAG Test Data Input / GPIO48 | JTAG debug port serial data input. This pin has an internal pullup resistor. |
| 11 | TCK | Digital Input/Output | JTAG Test Clock Input / GPIO47 | JTAG debug port clock input. |
| 12 | TMS | Digital Input/Output | JTAG Test Mode Select Input / GPIO46 | JTAG debug port test mode select input. This pin has an internal pullup resistor. |
| 13 | UART2_RTS | Digital Input/Output | UART2 Request to Send input / GPIO21 | UART2 RTS control input. |
| 14 | UART2_CTS | Digital Input/Output | UART2 Clear to Send output / GPIO20 | UART2 CTS control output. |
| 15 | UART2_RX | Digital Input/Output | UART2 RX data input / GPIO19 | UART2 receive data input. |
| 16 | UART2_TX | Digital Input/Output | UART2 TX data output / GPIO18 | UART2 transmit data output. |

**Table 2-1. Pin Function Description (continued)**

| Pin # | Pin Name | Type | Description[1] | Functionality |
|---|---|---|---|---|
| 17 | UART1_RTS | Digital Input/Output | UART1 Request to Send input / GPIO17 | UART1 RTS control input. |
| 18 | UART1_CTS | Digital Input/Output | UART1 Clear to Send output / GPIO16 | UART1 CTS control output. |
| 19 | UART1_RX | Digital Input/Output | UART1 RX data input / GPIO15 | UART1 receive data input. |
| 20 | UART1_TX | Digital Input/Output | UART1 TX data output / GPIO14 | UART1 transmit data output. |
| 21 | I2C_SDA | Digital Input/Output | $I^2C$ Bus data / GPIO13 | $I^2C$ bus signal SDA |
| 22 | I2C_SCL | Digital Input/Output | $I^2C$ Bus clock / GPIO12 | $I^2C$ bus signal SCL |
| 23 | TMR3 | Digital Input/Output | Timer 3 IO signal / GPIO11 | Pin is used as counter output or counter input clock. |
| 24 | TMR2 | Digital Input/Output | Timer 2 IO signal / GPIO10 | Pin is used as counter output or counter input clock. |
| 25 | TMR1 | Digital Input/Output | Timer 1 IO signal / GPIO9 | Pin is used as counter output or counter input clock. |
| 26 | TMR0 | Digital Input/Output | Timer 0 IO signal / GPIO8 | Pin is used as counter output or counter input clock. |
| 27 | SPI_SCK | Digital Input/Output | SPI Port clock / GPIO7 | SPI port clock. |
| 28 | SPI_MOSI | Digital Input/Output | SPI Port MOSI/ GPIO6 | SPI Port Master Out Slave In (MOSI) data signal. |
| 29 | SPI_MISO | Digital Input/Output | SPI Port MISO / GPIO5 | SPI Port Master In Slave Out (MISO) data signal. |
| 30 | SPI_SS | Digital Input/Output | SPI Port SS / GPIO4 | SPI Port Slave Select (SS) signal. |
| 31 | SSI_BITCK | Digital Input/Output | SSI Bit Clock / GPIO3 | SSI serial TX/RX clock and is bi-directional. |
| 32 | SSI_FSYN | Digital Input/Output | SSI Frame Sync / GPIO2 | SSI frame sync for data (RX or TX) and is bi-directional. |
| 33 | SSI_RX | Digital Input/Output | SSI RX data input / GPIO1 | SSI serial RX data input. |
| 34 | SSI_TX | Digital Input/Output | SSI TX data output / GPIO0 | SSI serial TX data output. |
| 35 | KBI_7 | Digital Input/Output | Keyboard Interface Bit 7 / GPIO29 | Asynchronous interrupt input. |
| 36 | KBI_6 | Digital Input/Output | Keyboard Interface Bit 6 / GPIO28 | Asynchronous interrupt input. |

**Table 2-1. Pin Function Description (continued)**

| Pin # | Pin Name | Type | Description[1] | Functionality |
|---|---|---|---|---|
| 37 | KBI_5 | Digital Input/Output | Keyboard Interface Bit 5 / GPIO27 | Asynchronous interrupt input. |
| 38 | KBI_4 | Digital Input/Output | Keyboard Interface Bit 4 / GPIO26 | Asynchronous interrupt input. |
| 39 | KBI_3 | Digital Input/Output | Keyboard Interface Bit 3 / GPIO25 | Used as output for keyboard interface. |
| 40 | KBI_2 | Digital Input/Output | Keyboard Interface Bit 2 / GPIO24 | Used as output for keyboard interface. |
| 41 | KBI_1 | Digital Input/Output | Keyboard Interface Bit 1 / GPIO23 | Used as output for keyboard interface. |
| 42 | KBI_0_HST_WK | Digital Input/Output | Keyboard Interface Bit 0 / HoST wake-up output / GPIO22 | Used as output for keyboard interface / Alternative function as a wake-up output (based on a timer) to external device. |
| 43 | COIL_BK | Power Switch Output | Buck converter coil drive output | Onboard buck converter connection to external coil, driven by onboard MOSFET. |
| 44 | LREG_BK_FB | Power Input | Voltage input to onboard regulators, buck regulator feedback voltage | • When using onboard buck converter, connect to load side of coil.<br>• When not using buck converter, connect to VBATT. |
| 45 | VBATT | Power Input | High side supply voltage to buck regulator switching MOSFET and IO buffers | Connect to battery. |
| 46 | RF_PLL_FLT | Analog Voltage | PLL filter connection | • Connection for PLL filter (Type 2, 2nd Order) when using primary crystal with frequency other than 24 MHz (13-26 MHz).<br>• No Connect for 24 MHz crystal. |
| 47 | XTAL_32_IN | Analog Input | Optional 32.768 kHz crystal oscillator input | Connect to 32.768 kHz crystal |
| 48 | XTAL_32_OUT | Analog Output | Optional 32.768 kHz crystal oscillator output | Connect to 32.768 kHz crystal |
| 49 | XTAL_24_OUT | Analog Output | Primary 24 MHz crystal oscillator output | • Connect to 13-26 MHz crystal (24 MHz default).<br>• No load capacitor required<br>• Do not load with any capacitance. |
| 50 | XTAL_24_IN | Analog Input | Primary 24 MHz crystal oscillator input | • Connect to 13-26 MHz crystal (24 MHz default).<br>• No load capacitor required<br>• Do not load with any capacitance. |
| 51 | RESETB | Digital Input | System reset input | Active low, asynchronous reset |
| 52 | TX_ON | Digital Input/Output | Control output for external RF component / GPIO44 | Programmable control pin |

**Table 2-1. Pin Function Description (continued)**

| Pin # | Pin Name | Type | Description[1] | Functionality |
|---|---|---|---|---|
| 53 | PA_NEG | RF Output | RF power amplifier (PA) output negative | • Open drain. Must be connected to RF_BIAS through a bias network.<br>• Only used for external dual port operation.<br>• Do not use for single port operation. No Connect. |
| 54 | PA_POS | RF Output | RF power amplifier (PA) output positive | • Open drain. Must be connected to RF_BIAS through a bias network.<br>• Only used for external dual port operation.<br>• Do not use for single port operation. No Connect. |
| 55 | RF_BIAS | Analog Power Output | Analog VDD regulator output | When using dual port operation, tie to PA_POS and PA_NEG through bias networks. |
| 56 | ANT_1 | Digital input / Output | Control output for external RF component / GPIO42 | Programmable control pin. |
| 57 | ANT_2 | Digital input / Output | Control output for external RF component / GPIO43 | Programmable control pin. |
| 58 | RF_GND | Power Input | RF ground. | Connect to ground VSS. |
| 59 | RX_ON | Digital input / Output | Control output for external RF component / GPIO45 | Programmable control pin. |
| 60 | RF_RX_TX | RF Input/Output | RF single-ended, single port input and output | • Interfaces to onboard balun. 50 Ω impedance<br>• Full bidirectional port with onboard T/R switch.<br>• Used as single-ended RF input port for dual port operation with PA_NEG and PA_POS PA outputs. |
| 61 | ADC2_VREFL | Analog Input or Digital Input / Output | Low reference voltage for ADC2 / GPIO39 | VREFL for ADC2. |
| 62 | ADC1_VREFL | Analog Input or Digital Input / Output | Low reference voltage for ADC1 / GPIO41 | VREFL for ADC1. |
| 63 | ADC1_VREFH | Analog Input or Digital Input / Output | High reference voltage for ADC1 / GPIO40 | VREFH for ADC1. |
| 64 | ADC2_VREFH | Analog Input or Digital Input / Output | Low reference voltage for ADC2 / GPIO38 | VREFH for ADC2. |
| 75-79 | VSS | Power input | External package GND pads. Common VSS. | Connect to ground. |
| 84-88 | VSS | Power input | External package GND pads. Common VSS. | Connect to ground. |
| 93-97 | VSS | Power input | External package GND pads. Common VSS. | Connect to ground. |

**Table 2-1. Pin Function Description (continued)**

| Pin # | Pin Name | Type | Description[1] | Functionality |
|---|---|---|---|---|
| 102 | MDO01 | Digital Input/Output | Message Data Out Bit 1 output / GPIO52 | Nexus debug port message data output Bit 1. |
| 103 | MDO00 | Digital Input/Output | Message Data Out Bit 0 output / GPIO51 | Nexus debug port message data output Bit 0. |
| 104-106 | VSS | Power input | External package GND pads. Common VSS. | Connect to ground. |
| 111 | MDO03 | Digital Input/Output | Message Data Out Bit 3 output / GPIO54 | Nexus debug port message data output Bit 3. |
| 112 | MDO02 | Digital Input/Output | Message Data Out Bit 2 output / GPIO53 | Nexus debug port message data output Bit 2. |
| 113 | MSEO1_B | Digital Input/Output | Message Start / End Out Bit 1 output / GPIO60 | Nexus debug port message start / end output Bit 1. Signal is active low. |
| 114 | MSEO0_B | Digital Input/Output | Message Start / End Out Bit 0 output / GPIO59 | Nexus debug port message start / end output Bit 0. Signal is active low. |
| 115 | VSS | Power input | External package GND pads. Common VSS. | Connect to ground. |
| 120 | MDO05 | Digital Input/Output | Message Data Out Bit 5 output / GPIO56 | Nexus debug port message data output Bit 5. |
| 121 | MDO04 | Digital Input/Output | Message Data Out Bit 4 output / GPIO55 | Nexus debug port message data output Bit 4. |
| 122 | RDY_B | Digital Input/Output | Ready output / GPIO61 | Nexus debug port ready output. Signal is active low. |
| 123 | EVTO_B | Digital Input/Output | Event Out output / GPIO62 | Nexus debug port event out output. Signal is active low. |
| 124 | DIG_REG | Digital Power Output | Digital core logic VDD supply. | 1.2 VDC internally regulated VDD supply to digital logic core. No Connect.. For test only |
| 129 | MDO07 | Digital Input/Output | Message Data Out Bit 7 output / GPIO58 | Nexus debug port message data output Bit 7. |
| 130 | MDO06 | Digital Input/Output | Message Data Out Bit 6 output / GPIO57 | Nexus debug port message data output Bit 6. |
| 131 | MCKO | Digital Input/Output | Message Clock Out output / GPIO50 | Nexus debug port message clock output. |
| 132 | EVTI_B | Digital Input/Output | Event In input / GPIO63 | Nexus debug port event in input. Signal is active low. |

**Table 2-1. Pin Function Description (continued)**

| Pin # | Pin Name | Type | Description[1] | Functionality |
|---|---|---|---|---|
| 133 | NVM_REG | NVM Power Output | FLASH (NVM) VDD supply. | VDD supply to FLASH. Typically No Connect. Can be connected to VDD when regulated 1.8Vdc mode is used. |
| 65-74, 80-83, 89-92, 98-101, 107-110, 116-119, 125-128, 134-145 | NC | | No Connect | These pads are provided for extra mechanical attach strength to meet demanding requirements of drop tests. |

[1] Pins described as GPIO have an alternative general purpose IO function.

## 2.3 Hardware Development Interface Interconnects

The MC1322x supports two development hardware interfaces.

### 2.3.1 ARM JTAG Interface Connector

The MC1322x supports connection to a subset of the ARM JTAG connector. The JTAG hardware interface is a standard 0.1 inch spacing, 20-pin header. Table 2-2 shows the device pins that are connected to the associated JTAG header pinouts if the JTAG connector is used.

**Table 2-2. ARM JTAG 20-Pin Connector Assignments**

| Name[1] | Pin # | Pin # | Name |
|---|---|---|---|
| VBATT | 1 | 2 | VBATT |
| NC[2] | 3 | 4 | GND |
| TDI | 5 | 6 | GND |
| TMS | 7 | 8 | GND |
| TCK | 9 | 10 | GND |
| RTCK | 11 | 12 | GND |
| TDO | 13 | 14 | GND |
| VBATT (pullup)[3] | 15 | 16 | GND |
| NC | 17 | 18 | GND |
| NC | 19 | 20 | GND |

1 NC = No Connect.

2 MC1322x does not support separate JTAG reset TRST.

3 VBATT through a 100k-Ω pullup.

### 2.3.2 Nexus Mictor Interface Connector

The MC1322x also supports connection to a subset of the defined Nexus Mictor connector. The hardware interface is a 38-pin Mictor target connector. Table 2-3 shows the device pins that are connected to the associated Mictor pin outs if the Mictor connector is used.

**Table 2-3. Nexus 38-Pin Mictor Connector Assignments**

| Name[1] | Pin # | Pin # | Name |
|---|---|---|---|
| NC | 1 | 2 | NC |
| NC | 3 | 4 | NC |
| NC | 5 | 6 | RTCK |
| NC | 7 | 8 | NC |
| VBATT (pullup)[2] | 9 | 10 | EVTI_B |
| TDO | 11 | 12 | VBATT[3] |

**Table 2-3. Nexus 38-Pin Mictor Connector Assignments (continued)**

| Name[1] | Pin # | Pin # | Name |
|---|---|---|---|
| NC | 13 | 14 | RDY_B |
| TCK | 15 | 16 | MDO07 |
| TMS | 17 | 18 | MDO06 |
| TDI | 19 | 20 | MDO05 |
| VBATT (pullup)[4] | 21 | 22 | MDO04 |
| NC | 23 | 24 | MDO03 |
| NC | 25 | 26 | MDO02 |
| NC | 27 | 28 | MDO01 |
| NC | 29 | 30 | MDO00 |
| NC | 31 | 32 | EVTO_B |
| NC | 33 | 34 | MCKO |
| NC | 35 | 36 | MSEO1_B |
| NC | 37 | 38 | MSEO0_B |

1 NC means No Connect.

2 VBATT through a 100k-Ω pullup.

3 VBATT isolated by a 1k-Ω resistor.

4 VBATT through a 100k-Ω pullup.

# Chapter 3
# MC1322x System Considerations

## 3.1 Introduction

The MC1322x is the embodiment of a IEEE 802.15.4 node in a single Platform in a Package (PiP) which can provide solutions to proprietary, IEEE 802.15.4 MAC-compatible, or full ZigBee-compatible networks. This chapter presents information on application and operation of the node from a system level. The areas considered here are also covered in greater detail in other sections of this manual.

## 3.2 Power Connections and Design

The MC1322x can be used in three different power supply configurations that are detailed in the following sections:

- Varying VDD voltage from 2.0 - 3.6 VDC using onboard regulation
- Varying VDD voltage from 2.1 - 3.6 VDC with optional onboard buck regulator
- Fixed regulated VDD voltage at 1.8 VDC.

### 3.2.1 Power Pin Descriptions

The power pin connections for the PiP are shown in Table 3-1.

**Table 3-1. Power Pin Descriptions**

| Pin # | Pin Name | Type | Description | Functionality |
|---|---|---|---|---|
| 45 | VBATT | Power Input | High side supply voltage to:<br>• Digital logic regulators<br>• Oscillator regulators<br>• Buck regulator logic and switching MOSFETs<br>• KBI pads<br>• IO buffers | Connect to battery. |
| 43 | COIL_BK | Power Switch Output | Buck converter coil drive output | Onboard buck converter connection to external coil, driven by onboard MOSFETs. |
| 44 | LREG_BK_FB | Power Input | Voltage input to onboard regulators, buck regulator feedback voltage, analog regulators, and NVM | • When using onboard buck converter, connect to load side of coil.<br>• When not using buck converter, connect to VBATT. |

**Table 3-1. Power Pin Descriptions (continued)**

| Pin # | Pin Name | Type | Description | Functionality |
|---|---|---|---|---|
| 55 | RF_BIAS | Analog Power Output | Analog VDD 1.5V regulator output. Use only to supply analog voltage to RF PA outputs. | When using dual port RF operation, tie to PA_POS and PA_NEG through bias networks. |
| 58 | RF_GND | Power Input | RF ground. | Connect to ground (VSS). |
| 75-79 | VSS | Power input | External package GND pads. Common VSS. | Connect to ground. |
| 84-88 | VSS | Power input | External package GND pads. Common VSS. | Connect to ground. |
| 93-97 | VSS | Power input | External package GND pads. Common VSS. | Connect to ground. |
| 104-106 | VSS | Power input | External package GND pads. Common VSS. | Connect to ground. |
| 115 | VSS | Power input | External package GND pads. Common VSS. | Connect to ground. |
| 124 | DIG_REG | Digital Power Output | Digital core logic VDD supply. | 1.2 VDC internally regulated VDD supply to digital logic core. No Connect,. For test purposes only. |
| 133 | NVM_REG | NVM Power Output | FLASH (NVM) VDD supply. | VDD supply to FLASH. Typically No Connect. Must be connected to VDD when regulated 1.8Vdc mode is used. |

When designing power to the MC1322x PiP, the following points need to be considered:

- The PiP LGA package has a single common ground flag (VSS) with multiple solder pads. The multiple pad approach provides better solderability.

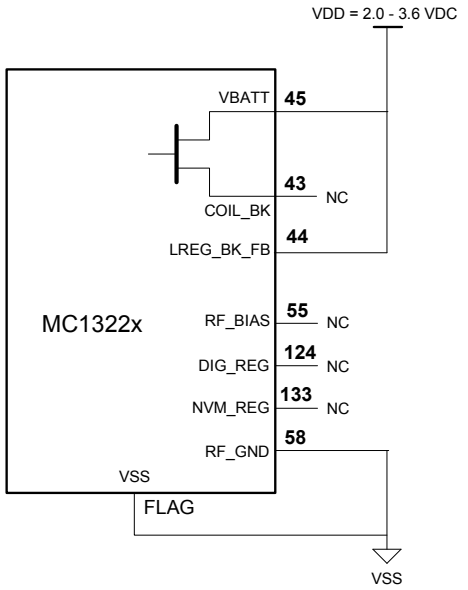
**NOTE**

Extra pads are provided on the package bottom simply for greater mechanical attach strength. These are “No Connect” and need not be connected to ground.

- The VBATT pin provides the high side supply to the P-channel MOSFET for the optional buck regulator and to the IO pads.
- For logic level compatibility between the MC1322x and external logic devices, VBATT, must be connected to a common logic supply with the external devices. The MC1322x IO are not 5 VDC tolerant.
- LREG_BK_FB feeds the common supply to the analog and NVM circuitry regulators. Bypass capacitance for voltage regulators is provided onboard the PiP.

### 3.2.2 Varying VDD from 2.0 - 3.6 VDC Using Onboard Regulation

The most common power connection is shown in Figure 3-1. This connection allows VDD to vary from 2.0 - 3.6 VDC and uses a minimum of components.




NOTE:
RF_BIAS is used with dual-port RF operation.

**Figure 3-1. Varying VDD from 2.0 - 3.6 VDC Using Onboard Regulation**

### 3.2.3 Varying VDD from 2.1 - 3.6 VDC Using Onboard Buck Regulator

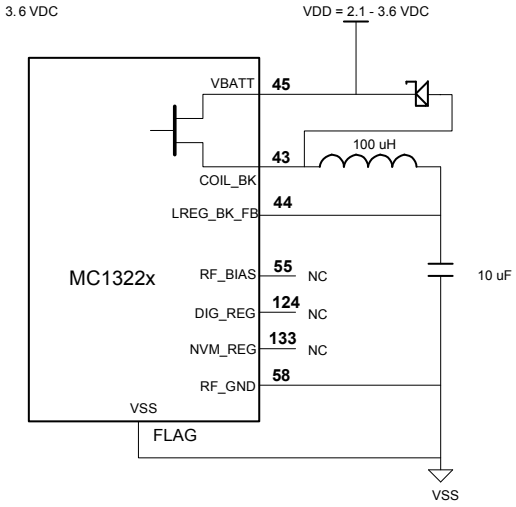



**Figure 3-2. Varying VDD from 2.1 - 3.6 VDC Using Buck Regulator**

For some battery-based applications, an optional, on board buck regulator is provided to help maximize battery life. Figure 3-2 shows the configuration of the buck regulator. On board MOSFETs are used to switch an external inductor and capacitor as a synchronous-rectifier regulator when enabled. The buck regulator drops the higher battery voltage to about 2.0Vdc that is applied to the on board linear regulators. This allows lower net current from the battery to maximize the life of the battery.

#### 3.2.3.1 Buck Regulator Hardware Considerations

For use of the buck regulator consider the following:

- The switch-mode regulator is a synchronous rectifier design. As a consequence, an external clamping Schottky diode to ground is not required.
- Due to potential breakdown voltage considerations, a Schottky diode is added from Pin 43 to Pin 45 to clamp overshoot on the drive side of the inductor to a diode drop above VBATT
- The maximum "ON"-resistance of the P-channel MOSFET switch is 4 ohms. This impedance impacts maximum current load for the switcher output.
- The recommended external inductor
  - 100 µH ± 20% inductance
  - 0.4-0.5 ohm maximum series resistance
  - Suggested part TDK #SLF6028T-101MR42
- The external capacitor is a 10µF ceramic component
- The buck regulator is actually used as a switcher in the 2.5-3.6 Vdc input supply range.
  - It is most useful for Lithium technology batteries that have an initial source voltage of 3.4 - 3.6 volts.
  - Manganese Dioxide Lithium, Polycarbonmonofluoride Lithium, and Alkaline (2 AAA cells) technologies are assisted less by the buck regulator because their initial source voltage is 3.0 - 3.1 volts.
- The buck regulator output serves only the analog regulator (for the radio) and the NVM regulator. It does not serve the MCU and other digital logic.

#### 3.2.3.2 Buck Regulator Usage/Software Considerations

This section describes procedures for use of the buck regulator.

**NOTE**

Although control of the buck regulator is described in this manual, Freescale also supplies application note AN3932, *MC13224 Configuration and Operation with the Buck Regulator*, which provides detailed information and example code and applications for use of the buck regulator.

##### 3.2.3.2.1 Buck Control and Status Mechanisms

Control of the buck regulator is managed in the CRM (see Chapter 5, "System Management (Including CRM)") primarily through the VREG_CNTL register (see Section 5.9.18, "Voltage Regulator Control (VREG_CNTL)")

- The buck design switch frequency is nominally 1.6 MHz. This controlled by the BUCK_CLKDIV[3:0] field of the VREG_CNTL register. If the normal 24 MHz crystal is used, the default value for this field can be used; if not, it must be programmed as required.
- Enabling the buck is accomplished by -
  - Set the BUCK_EN and BUCK_SYNC_REC_EN bits of the VREG_CNTL register

  - This can be done simultaneously; the default condition are both disabled
  - The buck ready status can be observed via the VREG_BUCK_RDY status bit in the CRM STATUS register (see Section 5.9.7, “Status (STATUS)”)
  - Proper start-up procedure must be followed
- Enabling the buck regulator from a cold start (POR) also requires buck enable in the PWR_SOURCE[1:0] field of the SYS_CNTL register (see Section 5.9.1, “System Control (SYS_CNTL)”). Default is NOT buck mode.
- The voltage output from the buck (supplied back to the series regulators via pin LREG_BK_FB) drives only the 1.8V NVM regulator and the 1.5V Analog regulator for the radio/modem.
  - These regulators get enabled by the VREG_1P5V_EN[1:0] and the VREG_1P8V_EN fields of the VREG_CNTL register
  - Enable is possible after warmup of the switcher
  - The regulator ready status can be observed via the VREG_1P5V_RDY and VREG_1P8V_RDY status bits in the CRM STATUS register
  - Proper start-up procedure must be followed
- After the battery voltage drops to <= 2.5V, the buck should be put into “bypass” mode
  - At 2.5V and below, the switcher efficiency drops and the switcher function is shut-off; the P-channel transistor then is fully turned-on and the P-channel and inductor become a simple series circuit to supply power to the series regulators
  - The application software must monitor the VBATT voltage via the ADC and cause the switch-over
    - For high duty cycle applications, a timer-based interrupt may be necessary
    - For low duty cycle applications, especially when the device is powered-down for long periods, a good strategy is to test the VBATT voltage at wakeup. The MC1322x recovers in bypass mode; if the VBATT voltage were too low, leave it in bypass and do not enable the buck regulator as a switcher.
  - Buck bypass is enabled by the BUCK_BYPASS_EN bit in the VREG_CNTL register

#### 3.2.3.2.2 Buck Control Procedures

Use of the buck regulator must be managed properly to get best efficiency and allow proper operation with varying battery voltage

- Device start-up from reset/POR - The circuit configuration of Figure 3-2 presents a conundrum for start-up: the NVM must be powered to allow the boot process to read the NVM, load RAM, and start executing from RAM. However, the default for power supply configuration is VBATT selected (PWR_SOURCE[1:0] = 0b00), where LREG_BK_FB is normally tied directly to the NVM regulator input. This allows the regulator to be enabled to access the NVM.

  With use of the buck, the P-channel MOSFET switch must be turned-on (BUCK_BYPASS_EN bit set) BEFORE the NVM can be ceased (the NVM regulator input has to be enabled through the switch). THE ROM BOOT CODE PROVIDES THIS FUNCTION.

**NOTE**

The user application code need not provide this function, because it resides in ROM-based boot code. The information as provided in the previous section is for user understanding only.

- Buck start-up after Boot - Once the device has booted (executing from RAM), the application can start-up the buck regulator (the boot code has disabled the bypass function). The following procedure must be followed:
  - Set PWR_SOURCE[1:0] field of SYS_CNTL Register = 2'b01 (enable buck mode)
  - Write the VREG_CNTL Register:
    - BUCK_CLKDIV[3:0] = 0xF (default), for 24MHz, otherwise as required by the reference oscillator
    - Set BUCK_SYNC_REC_EN, enable sync rectifier
    - Set BUCK_EN, enable buck regulator
  - Poll CRM STATUS Register: wait for VREG_BUCK_RDY status to be set (indicates buck is ready)
  - Write the VREG_CNTL Register:
    - These conditions remain - BUCK_CLKDIV[3:0] = 0xF, BUCK_SYNC_REC_EN = 1, BUCK_EN = 1
    - Write VREG_1P5V_EN[1:0] as required to enable radio (typically 2'b11)
    - Write VREG_1P5V_SEL[1:0] as required to set radio current (recommended 2'b11)
    - Set VREG_1P8V_EN as required to enable NVM regulator
  - Poll CRM STATUS Register: wait for VREG_1P5V_RDY and/or VREG_1P8V_RDY status to be set (indicates regulated voltage is ready)
  - Continue with application
- Monitor battery voltage (VBATT) for 2.5V and enable bypass - The application code must periodically sample VBATT via ADC1. Once VBATT has dropped below 2.5V, it is recommended the buck mode be changed to bypass (the switcher disabled and bypass mode enabled).

**NOTE**

A suggested means to enable ADC_1 sampling is use of the RTC interrupt request on a periodic basis.

  - Write the VREG_CNTL Register:
    - Clear BUCK_SYNC_REC_EN = 0, BUCK_EN = 0 to disable switcher
    - Write VREG_1P5V_EN[1:0] to disable radio (2'b00)
    - Clear VREG_1P8V_EN to disable NVM regulator
    - Set BUCK_BYPASS_EN to enable bypass transistor
  - Poll CRM STATUS Register: wait for VREG_BUCK_RDY status to be set (indicates buck bypass is ready)
  - Write the VREG_CNTL Register:
    - These conditions remain - BUCK_SYNC_REC_EN = 0, BUCK_EN = 0

– Write VREG_1P5V_EN[1:0] as required to enable radio (typically 2’b11)
– Write VREG_1P5V_SEL[1:0] as required to set radio current (recommended 2’b11)
– Set VREG_1P8V_EN as required to enable NVM regulator

— Poll CRM STATUS Register: wait for VREG_1P5V_RDY and/or VREG_1P8V_RDY status to be set (indicates regulated voltage is ready)

— Continue with application

#### 3.2.3.2.3 Buck Efficiency

The buck regulator can be used with a VBATT voltage range of 3.6 - 2.5 Vdc. Figure 3-3 shows typical buck efficiency versus load current at VBATT = 2.5 V.

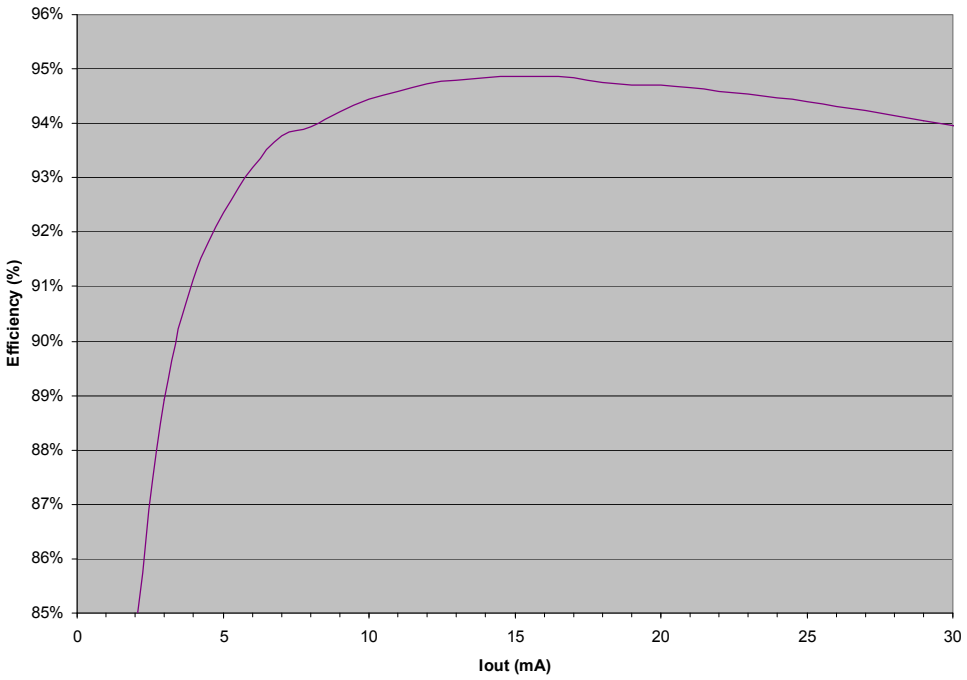



**Figure 3-3. Typical Buck Efficiency vs. Load Current (25ºC and VBATT = 2.5V)**

## 3.3 System Reset and Low Power GPIO Signal Connections

The MC1322x has three basic low power modes:

- Reset (off) - Master reset input RESETB is held asserted low and completely disables the device. Current is the absolute lowest with less than 0.3 µA dissipated. There is no onboard resistor to contribute to this current. Also, the RESETB pin has a top-side ESD diode to VDD like all other IO.
- Hibernate - This state shuts down all clocks with the exception a low power onboard oscillator or an optional 32.768 kHz oscillator for a wakeup timer. Depending on program options current dissipation is can be as low as 1 to 5 µA.

- Doze - This state shuts down most clocks but allows the reference oscillator to run. Depending on program options current dissipation is can be as low as 20 to 28 µA.

The use of the RESETB and GPIO signals during low power modes are detailed in this section.

### 3.3.1 GPIO Pin State During Reset, Default, and Low Power Modes

Table 11-6 shows the state of the GPIO signals versus mode of the device. The user is strongly advised to review this information.

Reset and low power operation must consider the following points:

- With RESETB active or in Hibernate or Doze, all GPIO except the KBI signals are unpowered (disconnected from VBATT).
- It is best practice not to drive a GPIO high or tie a GPIO high through a resistor when the GPIO is unpowered.

**NOTE**

Driving a GPIO high when not powered can forward bias the ESD diode and cause excess current. No damage will be done to the device, but excess leakage current can be high and defeat the desire for very low current while in a low power mode.

- The KBI signals are powered during reset, Hibernate, and Doze. - when reset is active all the KBI revert to inputs pulled low. When in low power Hibernate or Doze mode, 4 KBI are outputs to disable external circuitry and 4 KBI are inputs to allow an external asynchronous interrupt to wakeup the device.

### 3.3.2 Using GPIO in Low Power Modes

The MC1322x was designed to have extremely low current during low power modes. Because of the large number of GPIO on the device, the GPIO are generally disconnected from VBATT when a low power mode is enabled, with the exception of the KBI signals as described in Section 3.3.1, “GPIO Pin State During Reset, Default, and Low Power Modes”.

It is envisioned that the MC1322x will be generally used in two basic configurations:

- As a peripheral communication channel for a primary device such as a cell phone.
- Standalone (perhaps with its own peripheral functions), where the MC1322x is the primary device.

#### 3.3.2.1 MC1322x as a Peripheral Device

In a battery operated device such as a cell phone, the peripheral function must use as little power as possible, especially when disabled and not in use. For this scenario, the reset condition is a desired low power state because of absolute lowest current. Figure 3-5 shows a simple interconnect diagram for this application.

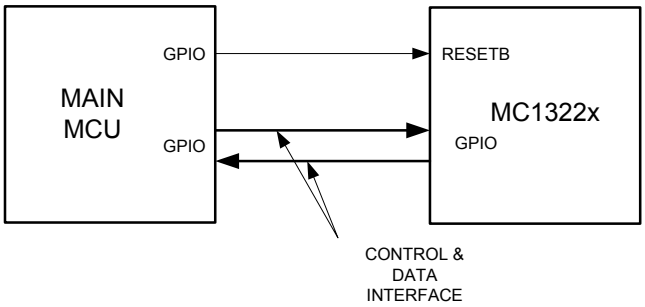



**Figure 3-4. MC1322x as a Peripheral Function**

- The MC1322x RESETB should be driven from a main MCU GPIO that is always active. When RESETB is active all MC1322x GPIO (except KBI) will be unpowered and need not be tied to any voltage.
- It is best that other peripheral devices not be tied to the MC1322x.
- The control and data interface between the devices would typically be a SPI, SSI, UART, or other serial interface.
  - MC1322x normal outputs (when reset active) revert to no power and will not drive the main MCU inputs. The main MCU may either provide a pull-down under this condition or be over-programmed as outputs in the low condition to hold the MC1322x signals low.
  - The MC1322x inputs should be driven low by the main MCU outputs.
  - If the $I^2C$ is used as the data interface, any required external pull-up resistors must be disabled when the RESETB is active.
- The disadvantage to using reset as a low power condition is the recovery time of the MC1322x. The device must recover from reset, start its clocks, and load its RAM to be able to be active.
- Before driving the MC1322x to a reset condition after the device has been active, the host should insure that important data or parameters on board the MC1322x be saved in non-volatile memory for later recovery.

### 3.3.2.2 MC1322x as Primary Device

The more common scenario is where the MC1322x is the primary device and may have peripheral devices such as sensors tied to it. In this situation reset is not typically used as a low power condition. Figure 3-5 shows examples of how external connections can be controlled.

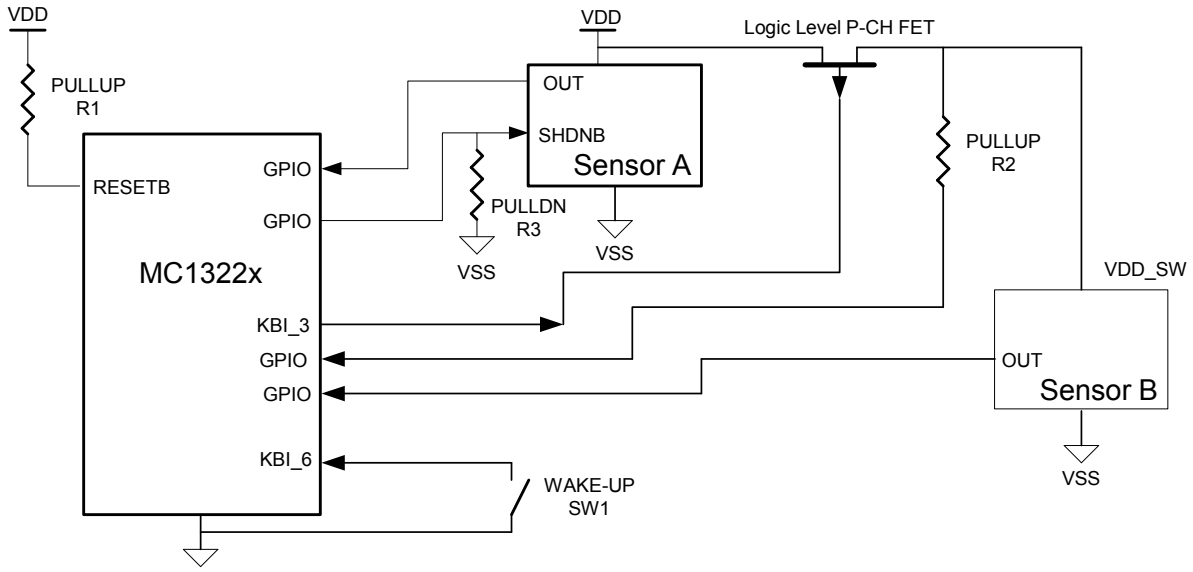



**Figure 3-5. MC1322x as a Primary Device**

- A pull-up resistor should be connected to RESETB.
- If the peripheral device has a shutdown input (active low) such as Sensor A, then a standard GPIO can be used to disable the device. The external pull-down is required to be sure the SHDNB input is pulled low in the reset or low power state.
- Other external connections such as pull-up resistors or a device without a shutdown option (Sensor B) can be controlled by an external P-channel MOSFET. A logic level gate voltage device is recommended, and the gate can be driven from KBI_3 through KBI_0. These outputs retain their state under low power states and default to a high condition to keep the FET off during reset (and Hibernate and Doze if desired).
- For a Hibernate or Doze wakeup option, an external switch can be connected to KBI_4 through KBI_7.

## 3.4 Using KBI Signals as Keypad/Pushbutton Interface

The KBI signals are unique in that they are the only GPIO that normally remain powered during low power operation. Of these, KBI_7:KBI_4 are always inputs and KBI_3:KBI_0 are always outputs during low power operation. The KBI_7:KBI_4 signals are also unique in that they are the only GPIO inputs that can generate an interrupt request from an external transition on the input (either low power or normal operation).

A common use is as a keypad or pushbutton array interface. With only four inputs and four outputs, a square matrix of 16 keys can be supported, as would be exemplified by a hex keypad. This is the max number of switches that can be supported for low power mode with wakeup capability.

If a larger number of keys is required a non-square array of N columns vs. 4 rows may be used. Additional non-KBI GPIO can be used as column drivers against the normal 4 row KBI signals. The added columns cannot provide wakeup during low power as they are not powered during powered.

Figure 3-6 shows two keypad configurations:

- 16-Key 4x4 array wired to the KBI signals - Designated as the “KBI Array”, signals KBI_3:KBI_0 as outputs provide the column drive, and KBI_7:KBI_4 as inputs provide the row sense. This example can be used for both low power and normal operation.
- 32-Key 8x4 expanded array - In the example, 16 additional keys are driven in 4 more columns by 4 standard (non-KBI) GPIO. During normal operation, all keys can be used with event-driven interrupts through the KBI_7:KBI_4 inputs. However, only the original 16-key matrix can still be used for event-driven wakeup from low power.

**NOTE**

For proper use of the event-driver interrupt requests on KBI_7:KBI_4, see Section 5.9.2, “Wake-up Control (WU_CNTL)”. The interrupts are always controlled via the CRM. However, during normal operation all the KBI pads are controlled by the GPIO Controller and must be programmed as accordingly.

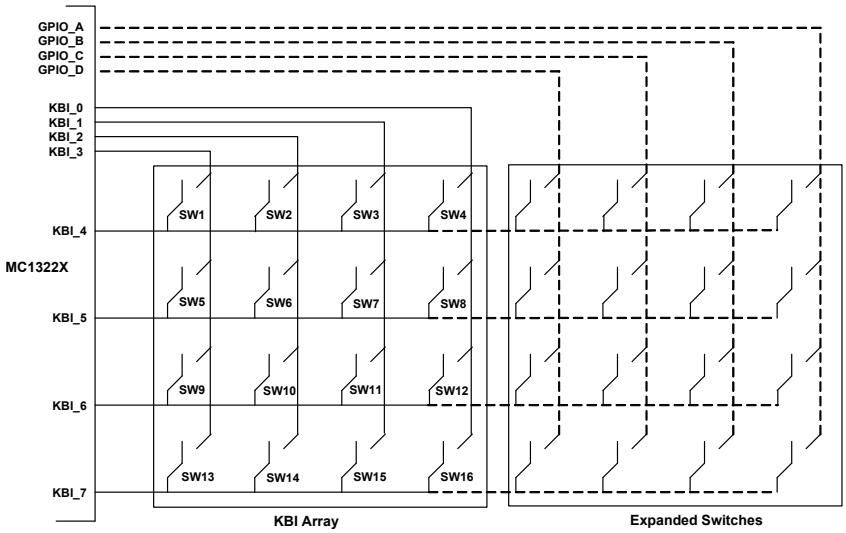



**Figure 3-6. Keypad Array Configurations**

## 3.5 External Clock Connections

The MC1322x uses clock sources for two primary functions:

- Reference clock oscillator - all active clocks for the radio, CPU, and peripherals are derived from the reference oscillator.
- Real Time Clock (RTC)- wakeup timer for Hibernate or Doze modes. The RTC input clock can be derived from three sources:
  - 2 kHz internal ring oscillator
  - Reference oscillator divided by 128 (nominally 24MHz/128)
  - Optional external 32 kHz crystal oscillator

The reference clock typically uses an external 24 MHz crystal, although it can be driven from an external source. The reference oscillator can also use a crystal ranging from 13 to 26 MHz in place of the 24 MHz standard, but there is an onboard PLL that must be used with the non-standard crystal.

A second optional 32.768 KHz crystal oscillator is available for the RTC. Again an external crystal is required. The advantage of the 32.768 KHz oscillator is that it is very low power while retaining excellent long term accuracy.

The MC1322x external clock connections are shown in Figure 3-7.

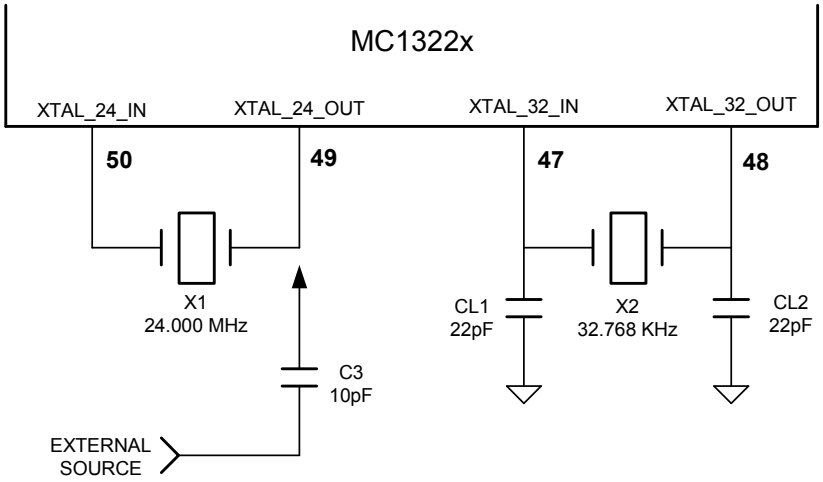



**Figure 3-7. MC1322x Clock Connections**

## 3.6 Reference Oscillator

The reference oscillator must always be functional and can use an external crystal or an external source.

### 3.6.1 Description

The MC1322x has a built-in Pierce reference crystal oscillator. It uses an off-chip crystal with frequency selection range from 13 MHz to 26 MHz with 24 MHz the default frequency. The oscillator is powered by a separate on-chip regulator and all the load capacitors are internal and programmable.

The reference crystal oscillator is able to drive a variety of crystals with a typical load of 9pF. The integrated load capacitors on each leg include a separately enabled course tune 4pF capacitor, a coarse tune capacitor array of 1, 2, 4 and 8pF (4 program bits), and fine tune capacitors of 5pF in 160 fF steps (5 program bits). The crystal load capacitance on each crystal leg is a total of 24pF + $C_{stray}$.

Figure 3-8 shows the user oscillator model.

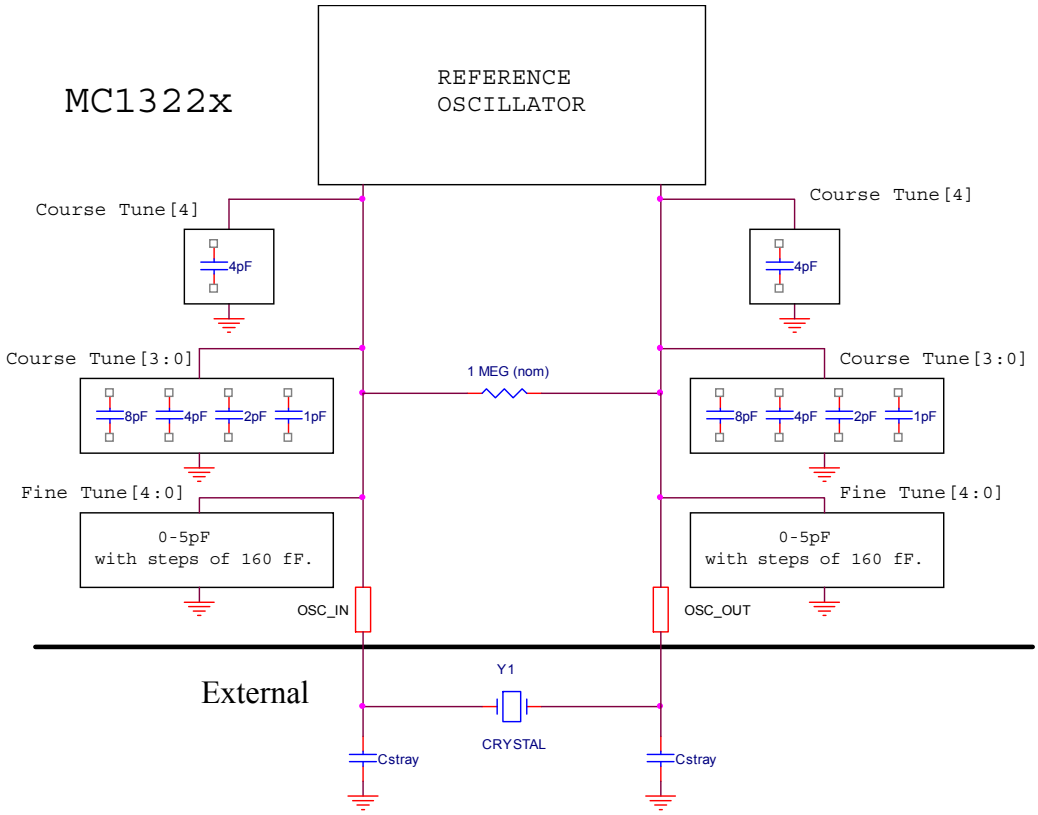



**Figure 3-8. MC1322x Reference Oscillator Model**

The model shows that with the onboard capacitor array, external load capacitors for the oscillator crystal are not necessary. Stray capacitance (sum of pad, package, and trace capacitance) is typically ~1.5pF.

The load capacitance $C_L$ to the crystal is symmetric and equals 1/2 ($C_{course}$ + $C_{fine}$ + $C_{stray}$). Target load capacitance is 9pF.

An onboard feedback resistor (1 Megohm, nominal) provides dc-biasing for the oscillator amplifier.

## 3.6.2 Reference Oscillator 24 MHz Crystal Specifications

The recommended 24 MHz crystal specifications are shown in Table 3-2.

**Table 3-2. Recommended 24 MHz Crystal Specifications**

| Parameter Name | Minimum | Typical | Maximum | Units |
|---|---|---|---|---|
| Crystal frequency | | 24.000 | | MHz |
| Operating temperature range<br>Industrial<br>Commercial<br>Extended | <br>-40<br>0<br>-40 | | <br>+85<br>+70<br>+105 | °C |
| Frequency tolerance @ 25 °C ± 3 °C | | | ± 10 | ppm |
| Frequency stability with temperature (full temp range, reference 25 °C) | | | ± 16 | ppm |
| Load capacitance | | 9 | | pF |
| Equivalent series resistance (ESR) | | | 40 | Ω |
| Shunt capacitance | | 1.3 | | pF |
| Tolerated drive level | 100 | | | µW |
| Mode of vibration | AT-cut / fundamental | | | |

## 3.6.3 Crystal Trimming

The MC1322x reference oscillator frequency can be trimmed (via changing the load capacitance) through programming the CRM register Reference XTAL Control (XTAL_CNTL) as described in Section 5.9.16, "Reference XTAL Control (XTAL_CNTL)". The Reference XTAL Control register of the CRM has two control fields that control the trimming.

- XTAL_CTUNE[4:0] - selects course load capacitance
  - XTAL_CTUNE[4] - 4 pF load to crystal
  - XTAL_CTUNE[3:0] - values 0-F select load capacitance equal to programmed number, i.e., value 7 selects 7 pF
- XTAL_FTUNE[4:0] - selects fine tuning for load capacitance; selects from 0-5 pF in 32 steps of approximately 156 fF

The trimming procedure varies the frequency by a few hertz per step; as the trim value is increased, the frequency is decreased. This feature is useful to set the crystal frequency for the radio accuracy as required by the IEEE 802.15.4 specification.

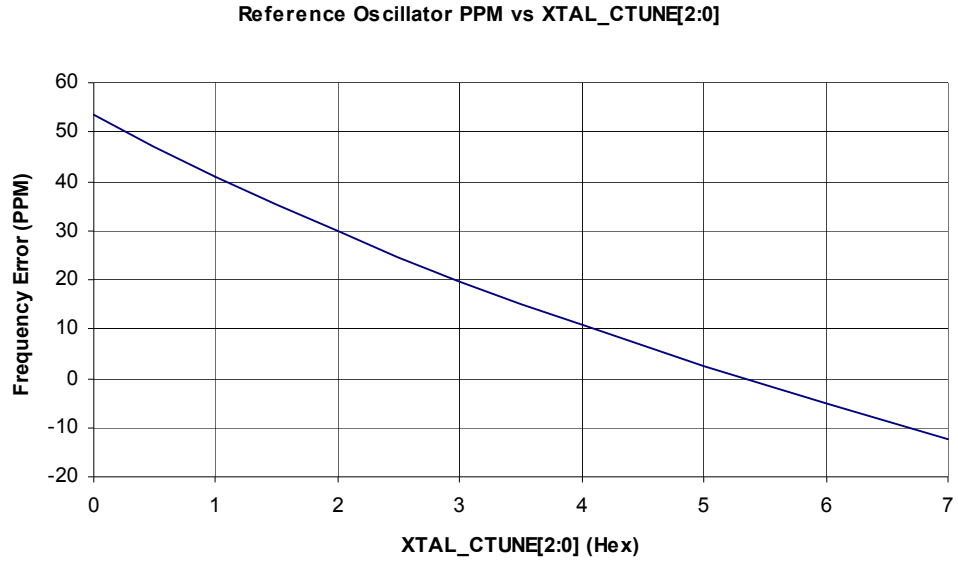



**Figure 3-9. MC1322x Reference Oscillator Variation vs. XTAL_CTUNE[2:0] (XTAL_CTUNE[4] = 1, XTAL_FTUNE[4:0] = 0x10, XTAL $C_L$ = 9pF)**

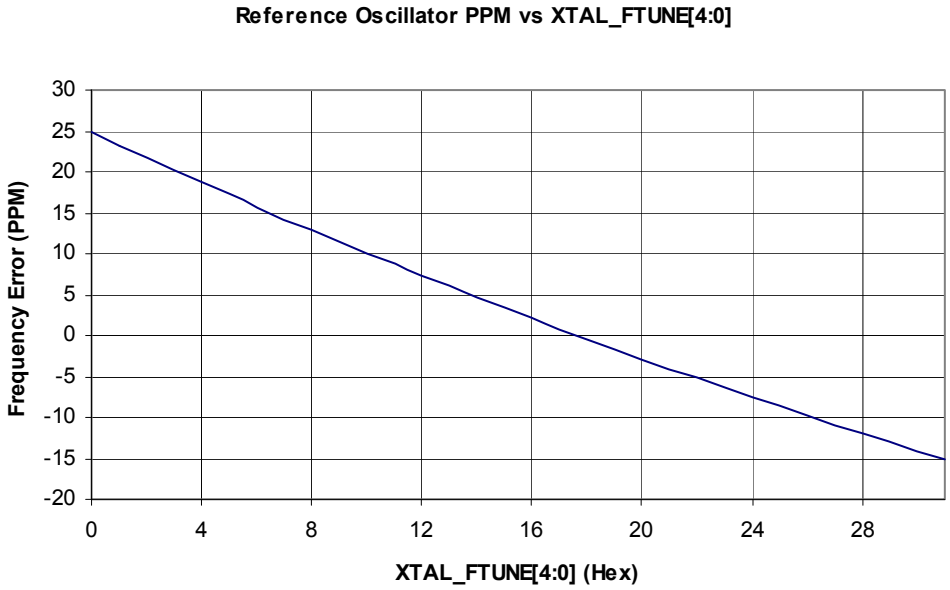



**Figure 3-10. MC1322x Reference Oscillator Variation vs. XTAL_CTUNE[2:0] (XTAL_CTUNE[4] = 1, XTAL_CTUNE[2:0] = 0x5, XTAL $C_L$ = 9pF)**

Figure 3-9 and Figure 3-10 illustrate the variation is oscillator frequency vs. capacitive loading for a 9pF $C_L$ crystal. For Figure 3-9 the separate 4pF course load in enabled, the fine tune is centered and the course tune array is varied. The result is that the frequency was reasonably centered with the XTAL_CTUNE[2:0] = 0x5. From this result, the XTAL_CTUNE[2:0] was set equal to 0x5 and the fine tune varied in Figure 3-10. The result is that for a 9pF load crystal suggested field settings are:

- XTAL_CTUNE[4:0] = 0x15
- XTAL_FTUNE[4:0] = 0x18

**NOTE**

- The suggested field settings are a typical starting point
- Good design practice dictates that a user characterize their board to find nominal trim settings based on their crystal, layout, and pcb characteristics. This should be done with a number of board examples to center the trim values across board variation
- With device-to-device and crystal variation, it can be expected that the reference frequency accuracy can vary as much as +/-10ppm to 12ppm with nominal trim settings. For a tighter starting accuracy, individual module trim at manufacture and test may be required.

### 3.6.4 Using a Reference Frequency Other Than 24 MHz

The reference oscillator can use a crystal from 13 to 26 MHz or use an external source in the same frequency range. If the default frequency of 24 MHz is NOT used, then the Modem Synthesizer (see Section 6.4, “Modem Synthesizer”) must activated and programmed for proper use of the radio. The synthesizer requires use of an external loop filter for the PLL. The external loop filter is a second order RC filter and is an integral part of the synthesizer loop, and the charge pump and VCO are connected to the loop filter. Figure 3-11 shows the external components for the required external loop filter.

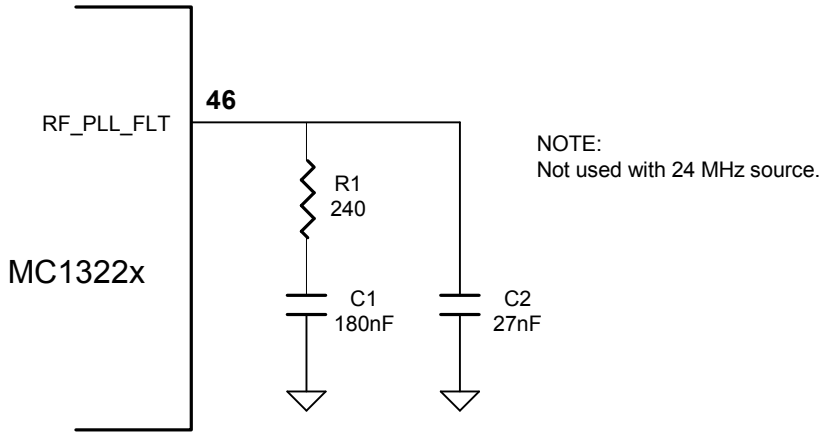



**Figure 3-11. External PLL Loop Filter**

### 3.6.5 Application Requirements for Reference Oscillator Performance

The IEEE 802.15.4 Standard has a required reference frequency tolerance of ±40 ppm max. Using a reference crystal as specified in Table 3-2, should provide performance no worse than ± (25-30) ppm over temperature. With the capacitive trim capability, even tighter tolerance (although not required) can be accomplished.

### 3.6.6 Using an External Reference Source

If an external source is desired in place of the reference oscillator crystal, the source can be connected as shown in Figure 3-7. The crystal is not installed and the reference source is injected at XTAL_24_OUT via a 10 pF coupling capacitor

- The source must have the same ±40 ppm frequency accuracy as an onboard reference.
- The source amplitude should be approximately 3 $V_{p-p}$ with the 10 pF capacitor. A lower amplitude may require a larger coupling capacitor.

## 3.7 32.768 kHz Crystal Oscillator (use optional)

The 32.768 KHz oscillator is strictly optional as the RTC can also be supplied by a low accuracy onboard 2 KHz oscillatory or by keeping the reference oscillator alive during sleep (Doze mode). The 32 KHz oscillator has the disadvantage of extra cost, but is much lower in power than doze mode and much more accurate than the onboard 2 KHz oscillator.

The use of the 32.768 KHz oscillator versus the on board 2 KHz oscillator is controlled via a register in the CRM (see Section 5.9.17, “32kHz XTAL Control (XTAL32_CNTL)”).

### 3.7.1 Description

The MC1322x has a built-in 32kHz Pierce crystal oscillator used to clock the RTC and the sleep state machine in the CRM module. The oscillator is powered by a 1 VDC power supply. It uses an external 32.768 kHz crystal with a typical load capacitance of 12.5 pF. The load capacitors are external.

Figure 3-12 shows the user oscillator model. The model shows that there are external load capacitors for the oscillator crystal. On each leg of the oscillator (**osc_in** and **osc_out**) there is:

- A fixed $C_L$ capacitor
- Stray capacitance - sum of pad, package, and trace capacitance (typically ~1.5pF)

The load capacitance $C_L$ to the crystal is symmetric and equals 1/2 ($C_L$ + $C_{stray}$). Target load capacitance is 12.5pF.

An active onboard feedback resistor provides dc-biasing for the oscillator amplifier.

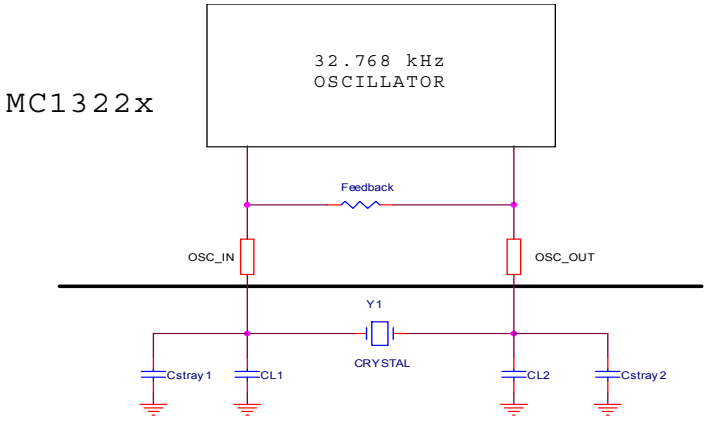



**Figure 3-12. MC1322x 32.768 kHz Oscillator Model**

### 3.7.2 32.768 kHz Crystal Specifications

Table 3-3 lists the recommended crystal specifications.

**Table 3-3. 32.768 kHz Crystal Specification**

| Parameter Name | Minimum | Typical | Maximum | Units |
|---|---|---|---|---|
| Crystal frequency | | 32.768 | | kHz |
| Storage temperature range | -55 | | +125 | °C |
| Operating temperature range Industrial | -40 | | +85 | °C |
| Frequency tolerance @ 25 °C | | | ± 20 | ppm |
| Frequency tolerance with temperature | -0.034 ±0.006ppm / (25-T) | | | ppm |
| Load capacitance | 11 | 12.5 | 13 | pF |
| Equivalent series resistance (ESR) | | | 60 | kΩ |
| Shunt capacitance | | | 1.35 | pF |
| Tolerated drive level | | | 1 | µW |

This specification is consistent with the Abracom Corporation crystal part number ABS25-32.768-12.5-B

## 3.8 Device Version ID

The MC1322x provides a device version ID code in the MACA_MC1322X_ID Register found in the MACA register set. See Section 9.7.6, “MC1322X_ID Register (MACA_MC1322x_ID)”

## 3.9 Transceiver RF Operation

For lowest cost and simplicity, the MC1322x RF operation requires only an external antenna. Additionally, a second TX PA port and dedicated control signals are available for utilization of an external PA and/or LNA for greater output power and/or sensitivity.

### 3.9.1 Standard Single-Ended RF Connection

The MC1322x radio interface provides for lowest cost and highest integration as shown in Figure 3-13. The PiP concept in the LGA package contains components interconnected with the IC to provide the integrated RF network.

- A single-ended 50-Ω antenna connection (RF_RX_TX pin) is provided via an onboard balun. The balun provides the conversion from single-ended external antenna node to the internal differential interface.
- The IC contains a fully integrated RX switch, RX LNA, differential TX PA and balun bias supply.
- Programmable output power ranges typically from -30 dBm to +4 dBm with 0 dBm typical default.
- The RX LNA provides <-94 dBm (typical) receive sensitivity - At 1% PER, 20-byte packet (well above IEEE 802.15.4 specification of -85 dBm).

- Onboard bypass capacitors for the radio.

As illustrated in Figure 3-13, there is an integrated internal RX switch and separate differential PA on the IC. The IC signals RFIN/OUT_P and RFIN/OUT_M are bidirectional and are connected for both for TX and RX. When receiving, the RX switch is enabled to the internal LNA and the PA is disabled. When transmitting, the RX switch is disabled (isolating the LNA) and PA is enabled. The bias supply provides the reference voltage to the balun that converts a single-ended antenna to the differential interface required by the device.

The user need only provide a proper impedance match to the 50-Ω single-ended RF interface signal (RF_RX_TX). The antenna would commonly be a 50-Ω impedance element.

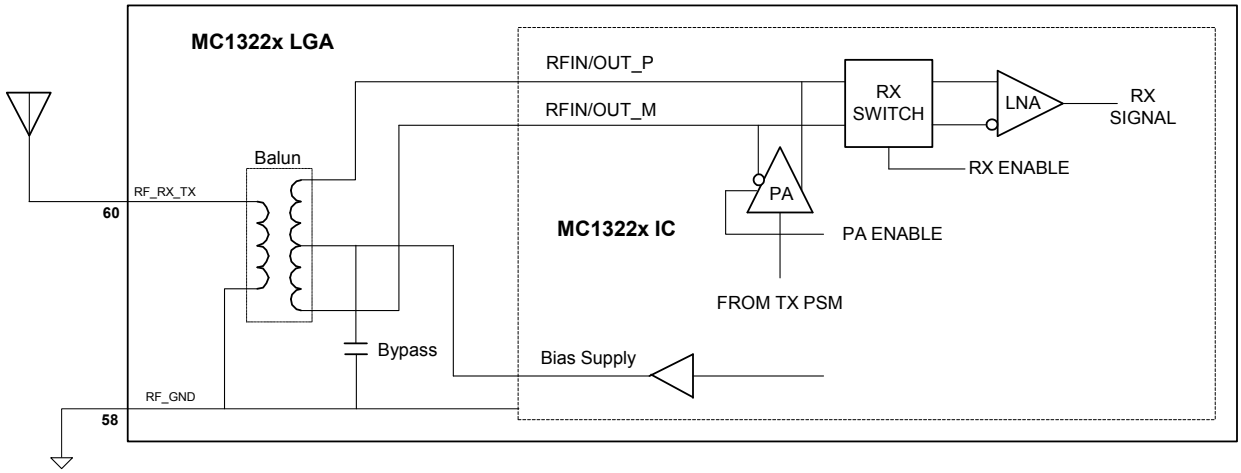



**Figure 3-13. MC1322x Standard Single-Ended RF Connection**

## 3.9.2 Extended RF Performance

For those applications choosing to include an external LNA for greater sensitivity, an external PA for greater output gain, or even multiple antennas, the MC1322x provides hardware support. Figure 3-14 illustrates that there are two basic modes for using the MC1322x with external RF hardware:

- Single-ended, single port only - as shown in the top configuration of Figure 3-14, with proper control, the single-ended port can be routed to a PA for TX and a secondary path (shown here with an LNA) for RX. The LNA may not be included because of the excellent receive sensitivity of the MC1322x
- Dual port, i.e., using secondary complementary PA outputs for amplified power along with the single-ended RF_RX_TX port - as shown in the bottom configuration of Figure 3-14, an alternative approach uses the device secondary PA outputs with an external PA as a separate TX path and the single-ended port as simply the receive path.

Other variations are possible, and the MC1322x provides optional dedicated RF hardware control signals, different modes, and special PA power control for higher TX power applications.

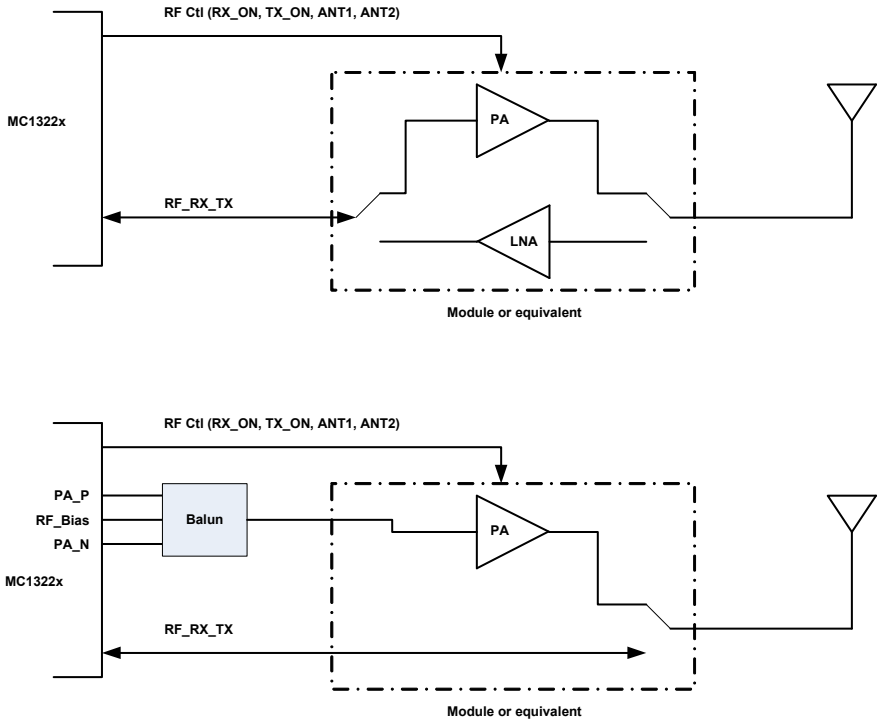



**Figure 3-14. Using the MC1322x with External RF Components**

### 3.9.2.1 RF Analog Signals

The RF analog signals consist of the bi-directional RF port (RF_RX_TX), the secondary complementary PA outputs (PA_P and PA_N), and the RF bias switch for the PA outputs (RF_Bias).

- RF_RX_TX - as illustrated in Figure 3-13 this RF port has an on-package balun that converts the single-ended external signal to a differential port at the device silicon. This differential port has an on-silicon LNA and a primary complementary PA that is automatically controlled for receive or transmit. This RF_RX_TX port can be used for RX, TX, or both. It is the only RX input however.
- PA_POS and PA_NEG - a separate set of differential PA outputs are provided for a secondary TX path. The differential PA outputs are typically used with an external PA for increased output power after they are converted to a single-ended signal through a balun( see Figure 3-15).

**NOTE**

The differential PA outputs are separate from the PAs that drive the RF_RX_TX single-port node and must be enabled via software control.

- RF_BIAS - this is a regulated supply node (VDDA = 1.5 V) to power the secondary PA outputs. This bias source should not be used to supply an external PA.

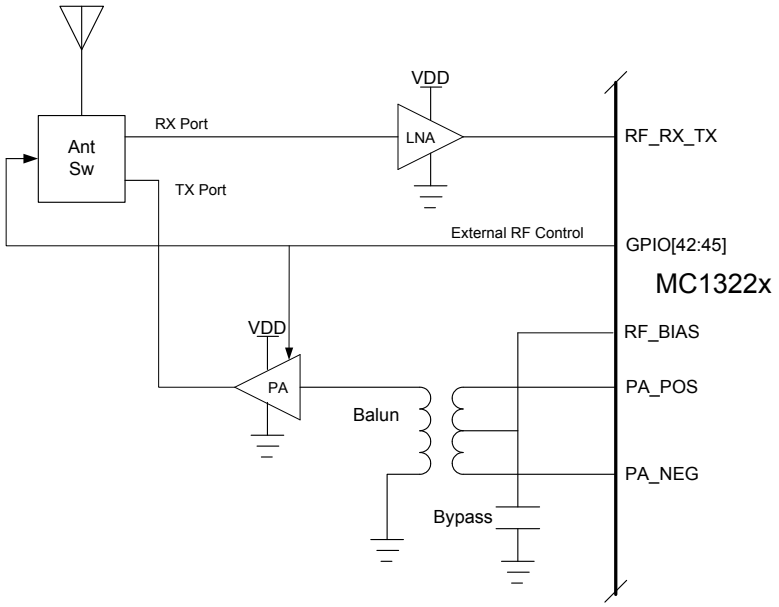



**Figure 3-15. Typical MC1322x Dual Port RF Configuration using Secondary PA Outputs**

### 3.9.2.2 TX Power Levels

The MC1322x TX power levels are set by software utility commands and there are two distinct modes:

- Standard power levels - for non-amplified applications
- "Powerlock" - restricted power level operation for use with external PAs. For powerlock mode:
  - The highest 802.15.4 channel (Channel 26) is disabled, and its use is disallowed
  - The allowable power levels are restricted.
  - The user must enable the powerlock via a software function call.

Table 3-4 lists the available typical transmit power levels for Normal and Powerlock modes.

**Table 3-4. MC1322x PA Level vs. Output Power**

| Power Level (Hex) | Transmit Power[1] (dBm) | Available for PowerLock[2] |
|---|---|---|
| 0 | -30 | **Yes** |
| 1 | -28 | **Yes** |
| 2 | -27 | **Yes** |
| 3 | -26 | **Yes** |
| 4 | -24 | **Yes** |
| 5 | -21 | **Yes** |
| 6 | -19 | **Yes** |
| 7 | -17 | **Yes** |
| 8 | -16 | No |

**Table 3-4. MC1322x PA Level vs. Output Power (continued)**

| Power Level (Hex) | Transmit Power[1] (dBm) | Available for PowerLock[2] |
|---|---|---|
| 9 | -15 | No |
| A | -11 | No |
| B | -10 | No |
| C | -4.5 | **Yes** |
| D | -3 | No |
| E | -1.5 | No |
| F | -1 | No |
| 10 | 1.7 | No |
| 11 | 3 | No |

1 The listed output power is measured at the RF_RX_TX port of the device

2 When Power Lock is enabled only the power settings shown as available may be used. This feature is intended for use with an external PA.

### 3.9.2.3 RX Demodulator Modes (Increased Sensitivity)

The MC1322x receiver demodulator includes a module called the Differential Chip Detector which has two modes of RX operation:

- Non-coherent Differential Chip Detection (DCD) Mode without automatic frequency control (AFC) - this is the default mode of operation in the Freescale applications. This is the more robust mode with the lowest current.
- Non-coherent Detection (NCD) Mode with AFC - This is a secondary mode of operation that can provide 3-4 dBm of increased sensitivity but increases the demodulator current drain about 3 mA.

The default mode is DCD, although the NCD mode can be enabled via software services, refer to Table 3-6.

### 3.9.2.4 RF Control Outputs

Four separate GPIO signals are available to control/enable external components and are designated as:

- ANT_1/GPIO42
- ANT_2/GPIO43
- TX_ON/GPIO44
- RX_ON/GPIO45

The MC1322x GPIO Function Select (see Section 11.4.1, “GPIO Function Select”) in the GPIO Module enables the ANT_1 and ANT_2 signals as one of two modes:

1. The designated GPIO (Function 0, default) - general purpose, not RF control
2. Controlled by transceiver state machine (Function 1, i.e., ANT_1 or ANT_2) - if enabled, the signal has a default rest state and toggles as defined in Table 3-5.

The MC1322x GPIO Function Select in the GPIO Module also enables the TX_ON and RX_ON signals as one of three modes:

1. The designated GPIO (Function 0, default) - general purpose, not RF control
2. Controlled by TX state machine (Function 1) - the TX state machine is pre-programmed to have a rest state for the IO (high or low) and toggles the signal while the state machine is active.
3. Controlled by RX state machine (Function 2) - the RX state machine is pre-programmed to have a rest state for the IO (high or low) and toggles the signal while the state machine is active.

Table 3-5 summarizes control and operation of the RF control outputs.

**Table 3-5. RF External Control Signal Summary**

| Signal Name | Pin Number | Function 0 [00] Default | Function 1 [01] TX Active[1] | Function 2 [10] RX Active[2] | Comment |
|---|---|---|---|---|---|
| ANT_1 | 56 | GPIO42 | Active | NA | ANT_1 can be made active low for RX only |
| ANT_2 | 57 | GPIO43 | Active | NA | ANT_2 can be made active low for TX only |
| TX_ON | 52 | GPIO44 | Rest = 0<br>Active = 1 | Rest = 1<br>Active = 0 | TX_ON can be made active high for TX or active low for RX mode; but only one configuration can be programmed at a time. |
| RX_ON | 59 | GPIO45 | Rest = 1<br>Active = 0 | Rest = 0<br>Active = 1 | RX_ON can be made active low on for TX or active high for RX mode; but only one configuration can be programmed at a time. |

1 Warm-up time is provided during the TX cycle for an external component before the PA ramps up to power.

2 Warm-up time is provided during the RX cycle for an external component before the receiver is active.

To more fully illustrate the use of the control signals, Figure 3-16 shows the available switching modes.

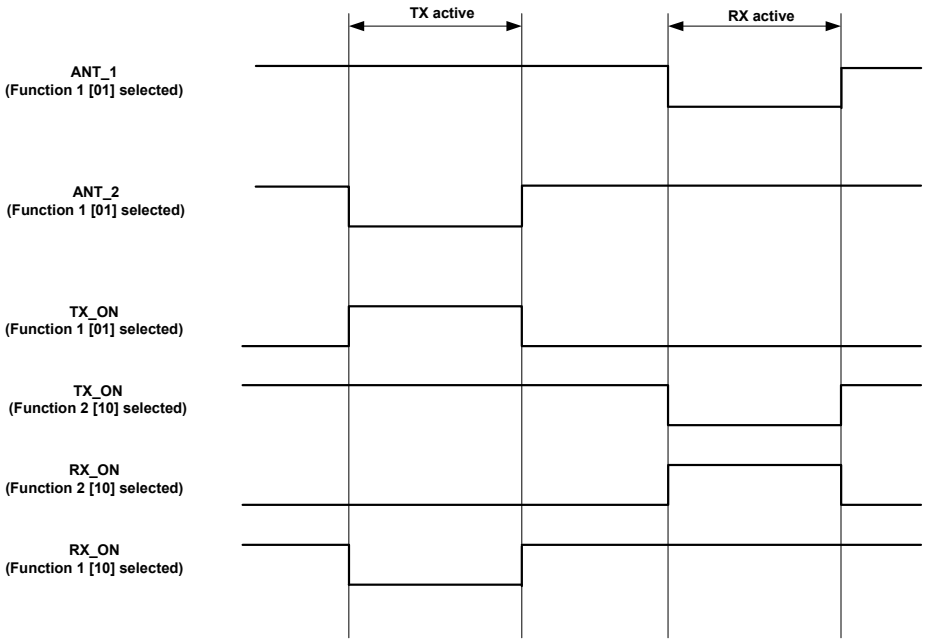



**Figure 3-16. RF Control Signal Available Modes**

In the figure, “TX active” or “RX active” refers to the time the TX or RX state machine is active in the radio. Each transmission is an independent event in that the VCO is enabled and locks, and then the onboard function is fully enabled.

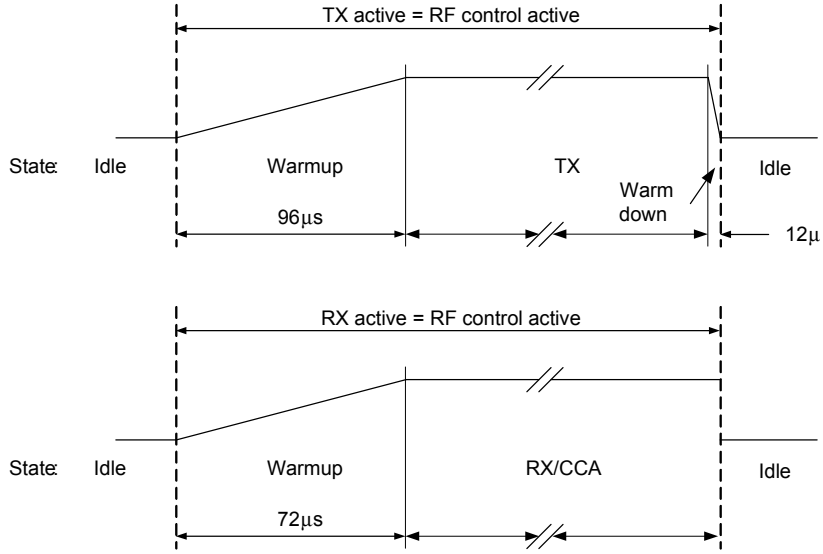



**Figure 3-17. RF Control Signal Timing**

Figure 3-17 relates the switching of the external control signals to the state of the radio:

- On TX the RF control signal is asserted as soon as the TX state machine is activated -
  - — As part of the warmup period, the VCO and TX circuitry require approximately the first 60 µs of the warmup period before the PAs are ramped-up to selected power.

— The first 60 µs is available as turn-on time for an eternal PA. Normally, this is much more than sufficient PA setup time which will typically stabilize in less than 5-10 µs.

— The RF control signal de-asserts when the TX state machine goes inactive after the power down of the PA.

- ON RX (CCA) the RF control signal is asserted as soon as the RX state machine is activated -
  — The RX circuitry requires a warmup of approximately 72 µs to be ready to receive.
  — The RF control signal de-asserts when the RX state machine goes inactive after the end of the receive sequence.

**NOTE**

- The use of the RF control outputs is initially configured through the Freescale BeeKit™ Wireless Connectivity Toolkit. See Section 3.9.2.5, "Programming RF Options"
- The RF control outputs can also be re-configured dynamically through an appropriate function call to a software base. See Section 3.9.2.5, "Programming RF Options"
- In low power modes, the RF GPIO do not retain their state and default to inputs which should be taken low in the low power state.

### 3.9.2.5 Programming RF Options

Programming of the RF options can be done at initialization and dynamically if required.

#### 3.9.2.5.1 Initial Configuration

The use and configuration of the RF modes and control signals can initially be set in the BeeKit™ Wireless Connectivity Toolkit via the "Platform Editor" of its "Project Configuration" tool (see Figure 3-18). Once configured these options will be retained when initialized from reset and recovering from a low power mode.

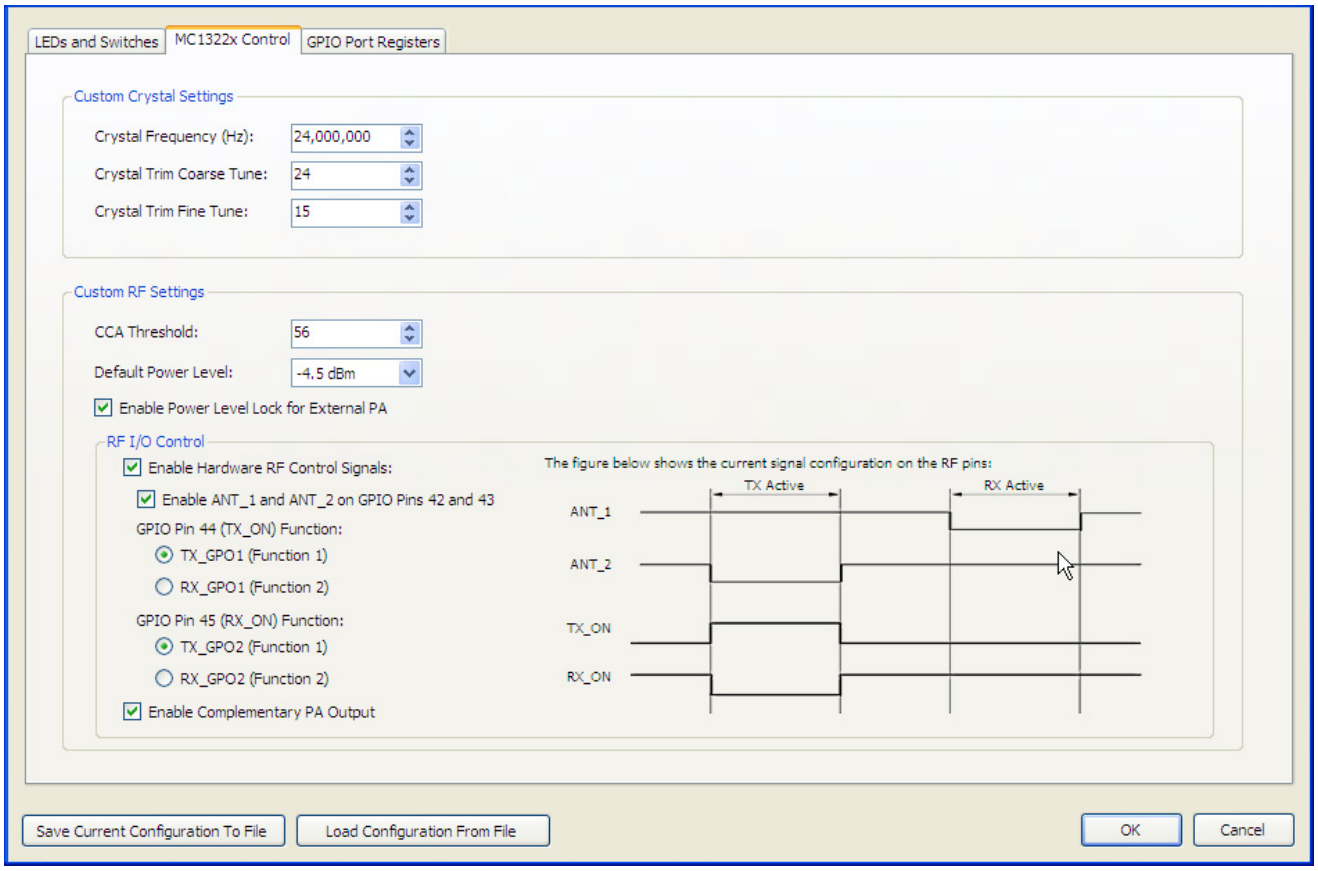



**Figure 3-18. BeeKit Project Configuration Tool**

Under the "MC1322x Control" page, the Custom RF Settings block includes several control boxes/buttons:

- "Enable Power Level Lock for External PA" - checking this box enables the Power Lock feature as described in Section 3.9.2.2, "TX Power Levels" (default is standard power levels).

**NOTE**

The editor will complain and not allow the box to be checked if the "Default Power Level" is not set to an allowed power lock TX level.

- "Enable Hardware RF Control signals" - checking this box enables the availability of the external RF control signals (defaults is no RF control signals enabled). Once this option is selected for use, additional selections can be made:
  - "Enable ANT_1 and ANT_2" - checking this box enables the switching of ANT_1 and ANT_2 (Function 1). Default is the signals disabled; standard GPIO.
  - GPIO Pin 44 (TX_ON) radio buttons - TX_ON will always be enabled; however, its function can be selected as either active high on TX (Function 1) or active low on RX (Function 2).
  - GPIO Pin 45 (RX_ON) radio buttons - RX_ON will always be enabled; however, its function can be selected as either active high on RX (Function 2) or active low on TX (Function 1).

**NOTE**

The application must provide for use of the GPIO[42:45] when the device is in low power mode because the GPIO revert to unpowered inputs during reset or low power)

- "Enable Complementary PA Output" - checking this box enables the separate complementary PA outputs (PA_P and PA_N) and disables the onboard PA common to the RX inputs (default is primary PA enabled).

#### 3.9.2.5.2 Dynamic Configuration

It may be desired to change the RF modes and control signals on a dynamic basis, such as to change control signal mode or TX power level mode. One example is where a PA may be enabled or disabled dynamically and use of the control signals would change in this instance. The Freescale-provided 802.15.4 MAC and 22xSMAC software provides function calls for this dynamic configuration. Table 3-6 lists the appropriate function call for each RF feature for both software bases.

**Table 3-6. API Function Calls to Configure MC1322x RF Operation**

| RF Feature | Function Call | Software Base[1] |
|---|---|---|
| Enable Power Lock | Asp_SetPowerLevelLockMode (bool_t enableLock) | 802.15.4 MAC |
| | uint8_t SetPowerLevelLockMode(bool_t state) | 22xSMAC |
| Set RF TX Power Level | Asp_SetPowerLevel (uint8_t powerLevel) | 802.15.4 MAC |
| | FuncReturn_t MLMEPAOutputAdjust (uint8_t u8Power ) | 22xSMAC |
| Enable Secondary PA | Asp_EnableComplementaryPAOutput (bool_t enable) | 802.15.4 MAC |
| | void SetComplementaryPAState(bool_t state) | 22xSMAC |
| Configure RF Control | uint8_t Asp_ConfigureRFCtlSignals (AspRfSignalType_t signalType, AspRfSignalFunction_t function, bool_t gpioOutput, bool_t gpioOutputHigh) | 802.15.4 MAC |
| | void ConfigureRfCtlSignals(RfSignalType_t signalType, RfSignalFunction_t function, bool_t gpioOutput, bool_t gpioOutputHigh) | 22xSMAC |
| Set Demodulator Type | void Asp_SetDemodulatorType(bool_t demDCDenable) | 802.15.4 MAC |
| | void SetDemulatorMode(DemTypes_t demodulator) | 22xSMAC |

[1] For the 802.15.4 MAC, refer to the *802.15.4 MAC PHY Reference Manual* and for the 22xSMAC, refer to the *MC1322x Simple Media Access Controller (SMAC) Reference Manual.*

## 3.10 Low Power Considerations

**NOTE**

The following paragraphs discuss using the MC1322x for lowest power and include comparisons of modes using greater or less current. The user is directed to the MC1322x Data Sheet (MC1322x.pdf) for exact current specifications.

### 3.10.1 Low Power Overview and Optimization

For battery operation, minimizing current draw for all conditions is of major concern. For the MC1322x, device usage is loosely divided into low power (sleep) options and normal run options.

The MC1322x allows for a wide variation of low power options, however, the most basic modes include:

- Reset - The MC1322x can be held in a reset state via the hardware reset pin. This has the absolute lowest power, but the longest recovery time.
- Hibernate - Has the lowest power (excluding reset), but does not have the reference oscillator running. Exit from Hibernate is via timers driven by low power oscillators (a default 2 KHz low accuracy oscillator or an optional 32.768 KHz crystal oscillator) or KBI asynchronous interrupt requests.
- Doze - Very similar to Hibernate, however, Doze keeps the reference oscillator running. It has the advantage of a low cost accurate time base ( no 32 KHz crystal required) while in sleep condition although at the cost of additional current.

**NOTE**

- Entering Hibernate or Doze mode is controlled via the CRM module as described in Chapter 5, “System Management (Including CRM)”.
- Section 3.3.2, “Using GPIO in Low Power Modes” describes GPIO in low power.

The typical user desires to minimize low power current while also maintaining a reasonable wakeup time. Although, the following paragraphs discuss many options, the **common optimal low power use model** is:

- Retain all RAM required to save the entire application image

**NOTE**

The default low power model for Freescale supplied software is all RAM retained.

- Use Hibernate mode
- If accurate wakeup timing is required, use the optional 32 kHz crystal oscillator for RTC timing

For run-time operation there are also options to help minimize current consumption:

- Clock control allows for varying CPU and bus clocks
- Peripherals can be enabled/disabled as required
- A bus-steal utility function allows for minimizing the MCU current while the radio is active

### 3.10.2 Controlling Low Power Current

Although reset is the lowest power option, this is typically only used when the MC1322x is a subsystem to some other controller. In standalone mode, where the device is the “system”, variations on sleep modes are used for lowest power. There are several programmable options that influence power (i.e., current draw) when low power operation is desired:

- Is the reference oscillator kept alive during sleep mode. This is the primary difference when designating sleep modes;
  a) Hibernate mode does not keep the reference oscillator alive.
  b) Doze mode does keep the reference oscillator alive. There is no appreciable lessening of startup time by keeping the oscillator running
- Amount of RAM contents retained during sleep mode.
- Is the MCU kept alive during sleep mode
- What is the required accuracy of the sleep time
- How fast is the required wakeup time. There is a trade-off between lowest power and faster wakeup time (from sleep to active radio)

#### 3.10.2.1 Hibernate Mode versus Doze Mode

Keeping the reference oscillator running during sleep operation is the only real difference between Hibernate and Doze modes:

- Hibernate has a significant advantage with lower current draw vs. Doze mode
- There is no advantage in startup time for Doze mode (even though the reference oscillator is running)
- The advantage of Doze mode is that a very accurate timebase for the RTC and wakeup timers can be kept without the use and cost of a 32.768 KHz crystal

#### 3.10.2.2 Retained RAM Contents

The MC1322x RAM is provided and controlled as a four blocks of 8Kbytes, 24Kbytes, 32Kbytes, and 32Kbytes, respectively (see Section 4.2, “Features”). There are factors to consider when retaining the RAM contents:

- Retaining an additional block of RAM has minimal impact on Hibernate or Doze current
- Retain only as much RAM as required for basic wakeup operation after low power mode if wakeup time is not an issue.
- RAM recovery time (loading from FLASH) has significant impact on wakeup recovery time. Approximately 2.86 ms are added to wakeup recovery time for every Kbyte of RAM that needs restored from FLASH

### 3.10.2.3 Retaining MCU Contents

The Sleep Control (SLEEP_CNTL) Register of the CRM (see Section 5.9.3, "Sleep Control (SLEEP_CNTL)") contains a MCU_RET control bit (default = off) that allows state retention of all the digital logic during sleep modes, although at a reduced voltage.

- If the MCU_RET condition is not set - MCU recovery from low power is the same as a cold start or reset condition, other than some amount of RAM contents have been retained (as programmed by user). The code starts executing from the bottom of RAM and the CPU exits from a reset condition.
  - Key registers/parameters may need to be saved in retained RAM or NVM before entering sleep mode
  - Applications code should be written to avoid problems with this condition
- If the MCU_RET condition is set - The digital logic (including the MPU) is not reset.
  - Program execution starts from the last program counter location, all CPU registers are intact
  - All program RAM contents must have been retained where the program is to start re-executing
  - The cost of MCU_RET is considerable additional current during the sleep mode condition.
  - Recovery time for applications software may be quicker because MCU initialization is minimized
  - The recovery flow must clear the wake-up conditions, similar to recovery without MCU retention

### 3.10.2.4 Required Accuracy of Sleep Time and RTC

Use of the low frequency oscillator options for sleep mode saves power and perhaps cost. Wakeup can be programmed via the wakeup timer or the RTC timer that are clocked by the default 2 KHz low accuracy oscillator or the optional 32.768 KHz crystal oscillator.

- The 2KHz oscillator is used for applications not requiring a high accuracy delay. Simple situations such as waking to take a sensor reading and report may be used with the 2KHz oscillator. The power-down timing delay is not critical and low cost is important.
- The optional 32.768KHz oscillator is useful for accurate timing delay or maintaining an accurate RTC on a continuous basis even through low power. Although a second low cost crystal is required for this oscillator, the RTC can run continuously as an accurate system clock and the wakeup timer now has a very accurate wakeup delay.

The alternative is to use the reference oscillator during low power (Doze mode) and use the reference clock/128 as the timer clock. The advantage is again a very accurate clock for RTC and wakeup delay without use of a second crystal. The penalty is a significant increase in low power current.

## 3.10.3 Wakeup or Recovery from Low Power Modes

Wakeup from low power consists of two parts which include the wakeup mechanisms and the recovery time.

### 3.10.3.1 Wakeup Mechanisms

The MC1322x wakeup mechanisms include:

1. Reset - If the device is in a low power mode and a hardware reset occurs, the active signal would override the low power condition and put the device in full reset. Upon release of the hardware reset signal, the device will wakeup and boot from a cold start condition.
2. Timer-based - There are two separate timers available that can generate a wakeup interrupt
   a) wakeup timer - This timer is a 32-bit counter that can be clocked from the 2kHz ring oscillator, the reference oscillator divided by 128, or the 32.768kHz oscillator. The source is selected as a sleep option. This counter is zeroed on entry into sleep and can be used for a very accurate and/or long sleep period. The time-out period is set via a register and the interrupt is generated when the counter and register value match.
   b) Real Time Clock (RTC) - This is a second 32-bit timer that can be clocked from the 2kHz ring oscillator or the 32.768kHz oscillator. The RTC runs continuously (also during normal operation) and can generate a periodic interrupt based on a register value. Once a given interrupt is generated, a new match value for the counter is generated internally, and the next interrupt will be generated based on the programmed delay count. This timer has the advantage that a wakeup can be based on a rolling, accurate timed-interval as opposed to a delay from entering sleep.
3. KBI input transition - Four KBI signals (inputs KBI_7:KBI_4) remain powered during sleep mode and a transition on the input can be enabled to generate an asynchronous interrupt request to the device. The transition allows an external event (such as a button push) to wakeup the device.

Any combination of these events can be used as sleep mode wakeup. Details of implementing these options are detailed in Chapter 5, “System Management (Including CRM)”.

### 3.10.3.2 Wakeup Recovery Time

The wakeup/reset recovery time can be an important parameter for some applications where it is required to receive or transmit a frame quickly. The recovery time is variable and is impacted by how much RAM must be restored, circuit start-up, and how much of the device must be re-initiated. Generally, there is a trade-off between sleep power and recovery time; the lower the sleep current draw, the longer the recovery time.

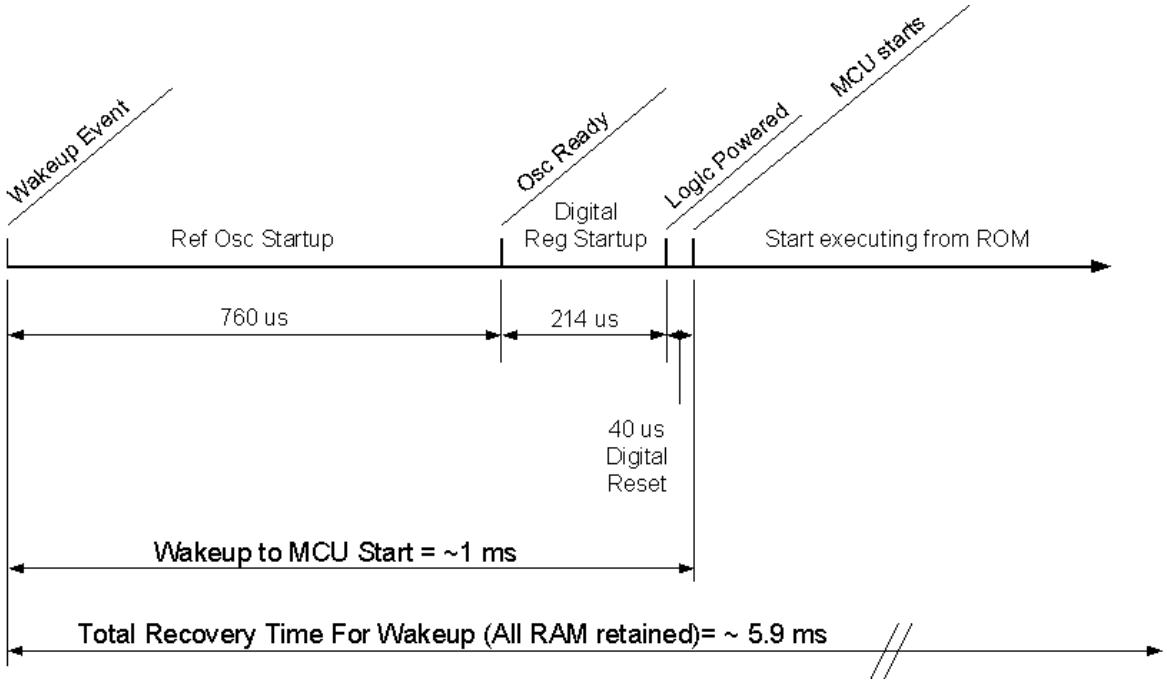



**Figure 3-19. Wakeup Recovery Time**

Table 3-19 illustrates a typical timeline for the recovery process. The recovery timeline starts with a wakeup event (or reset release) and includes the following simplified steps:

1. Startup time to CPU activity- The reference oscillator typically must start and then the digital voltage regulator must start. This time can vary from about 760 µs to1000 µs; note that there is no shortening of startup time if the 32 MHz oscillator is already running.
2. CPU clock starts and core starts executing boot code from ROM - as shown in Figure 3-23, the boot flow first tests whether the wakeup event is from a reset or low power state. The wakeup delay time is dependent on the mode:
   - From reset - RAM gets cleared, a valid FLASH image is assured, and then the entire application executable is transferred from FLASH to RAM. At this point, program execution gets transferred to RAM. This path will always require the longest recovery time due to the early steps as well as the loading of RAM -
     - The delay from reset release to start of application execution in RAM is dependent on the size of the object code image.
     - Figure 3-20 shows wakeup delay versus object code size. The delay time increases because the boot code must move a bigger image from serial NVM to RAM

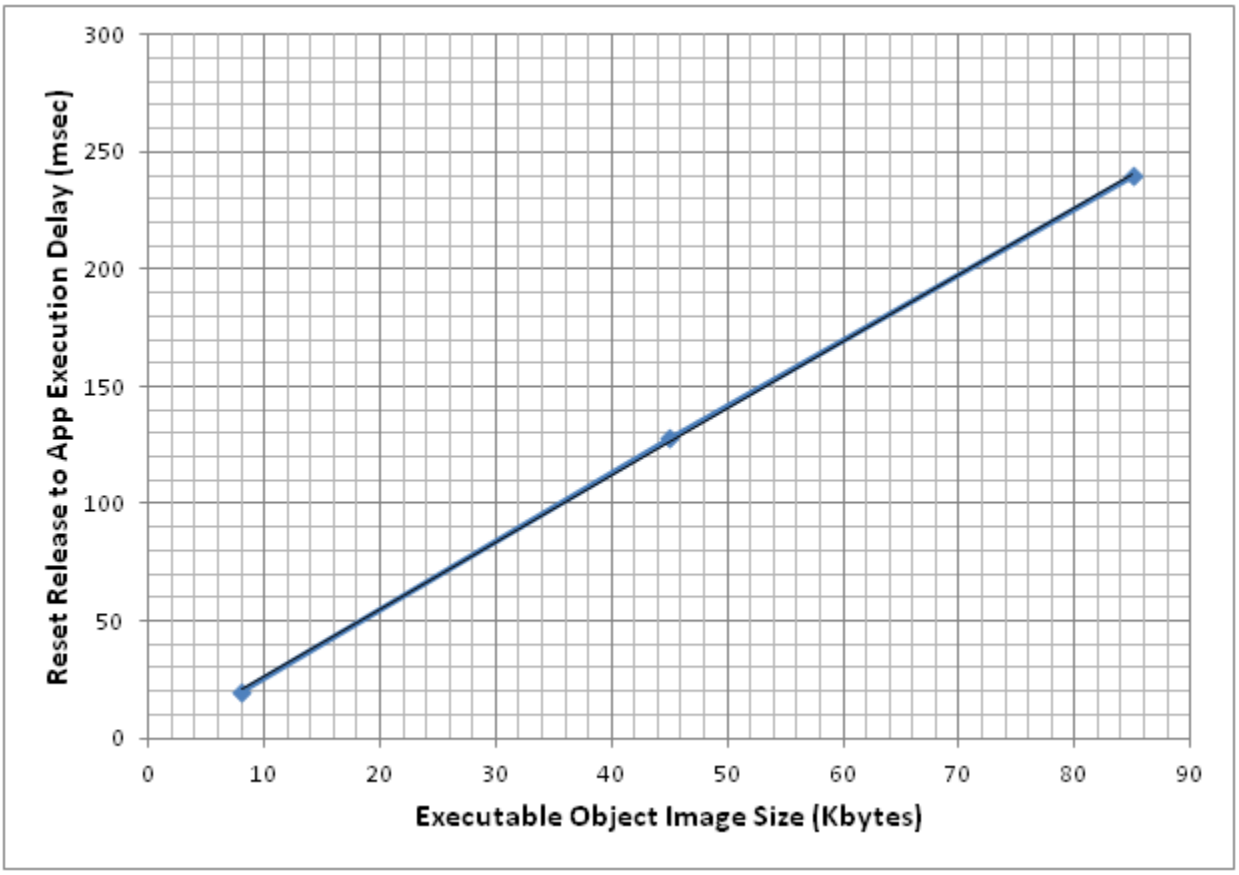



**Figure 3-20. Reset Wakeup Delay vs. Executable Object Size**

— From a low power wakeup event - if the boot code determines that wakeup is from low power versus reset, transfer of program execution to RAM is done without any transfer of program code from FLASH to RAM -

- The delay from wakeup event to start of application execution in RAM is independent of the size of the object code image. This delay is typically about 5.9 ms.
- Common practice is to retain all required RAM so that all applications code is available immediately without any recovery

**NOTE**

If lowest power is desired during Hibernate and wakeup is not from a reset condition, only 8 Kbytes of RAM can be retained. However, it is the responsibility of the application to restore the unsaved executable code to RAM.

3. Initialize transceiver, peripherals, and the software stack - once program execution starts in RAM (boot flow is exited), the transceiver and peripheral registers first get initialized. Following setup of the registers, the communications stack (typically based on the Freescale IEEE 802.15.4 MAC) gets restarted.
   — From start of application code to stack ready (send or receive frames) delay is typically about 121 ms. What MCU peripherals that are in use and must be initialized can impact this delay.
   — The application code must also determine cause the of the wakeup event and service it accordingly

In summary, the full wakeup recovery time is the sum of:

1. The clock/CPU startup time and boot recovery run - dependent on how much RAM must be initialized.
2. System initialization time, once the application is running from RAM.

The optimal trade-off of low power versus recovery time is normally using Hibernate mode with all required program RAM retained, and no MCU retention:

- Hibernate current typically equal to 2-5 µa - depending on retained RAM size
- Total recovery time to active network from wakeup event typically equal to about 127 ms - sum of the wakeup to RAM execution delay plus MCU and stack initialization delay

## 3.10.4 Run-time Current

Run-time current is the actual operating current of the MC1322x when in normal operation, not sleep mode. By understanding how operation is controlled, current usage can be predicted and minimized.

### 3.10.4.1 Operating Current Profile

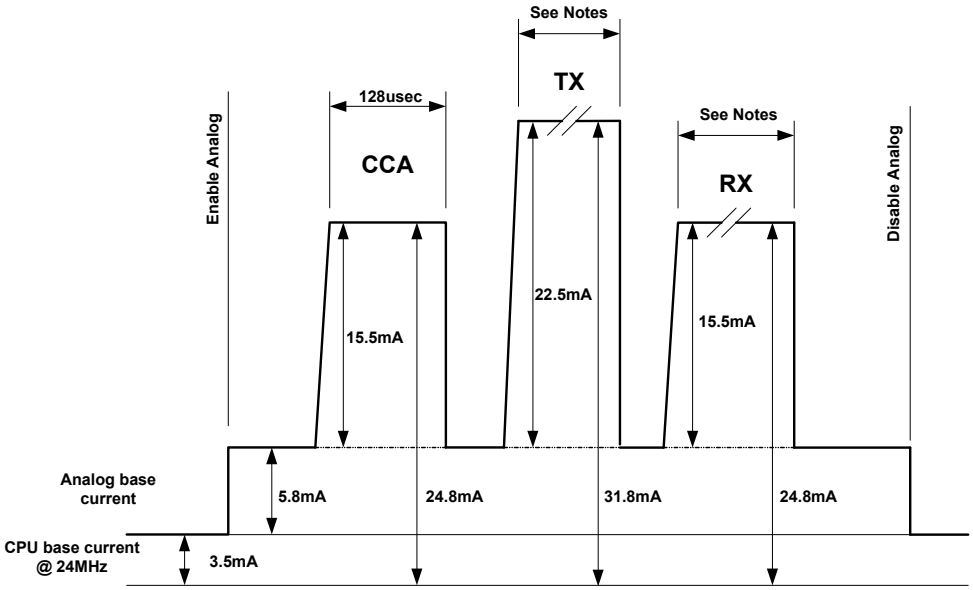
The MC1322x power distribution network is diagrammed in Figure 5-2. Some key attributes of the network include:

- All the digital logic is powered from the VBATT rail connection
  — The CPU current draw is most affected by the CPU clock rate
  — The MCU peripheral current draw is most affected by enabling/disabling required peripherals, especially the ADC module
- The NVM and Analog circuitry are individually regulated and are powered from the LREG_BK_FB rail connection.
  — The NVM regulator is typically only enabled to read/write the serial FLASH
  — The Analog regulator must be powered for any radio operation. Although the analog current increases with an actual TX, RX or CCA operation, there is also a quiescent analog current when the analog regulator is enabled

Figure 3-21 illustrates how the MC1322x operating current varies versus operational mode:

- The CPU base current is typically about 3.5 mA with a CPU/peripheral clock of 24 MHz
- The Analog regulator must be enabled (turning on the radio) before entering a radio cycle (CCA, TX or RX), and the additional analog base current is about 5.8mA

- Active radio modes cause the radio current to increase while one is active
  - CCA/RX adds about 15.5mA
  - TX adds about 22.5mA at nominal 0dBm power out.
- The duration of the CCA cycle is about 128us as determined by the IEEE 802.15.4 Standard
- TX and RX duration are dependent on the frame length (see Notes in Figure 3-21). Also, the RX duty cycle may be difficult to predict because the radio must be in receive in anticipation of a a frame, and this can be highly dependent on the use model.




**Notes:**
1. **CCA duration set by IEEE 802.15.4 Standard**
2. **A frame typically has 4 bytes of preamble, 1 byte of SFD, 1 byte of FLI, 2 bytes of FCS plus the payload data (125 bytes maximum). The 8 bytes of overhead equal 256us of delay, and each byte of payload data adds another 32usec.**
3. **The RX time is not deterministic. The receiver must be on before start of an expected frame and and will stay active for the entire length of a frame once reception starts.**

**Figure 3-21. MC1322x Operating Current Profile**

### 3.10.4.2 Controlling Run-time Operating Current

When the MC1322x is not in sleep mode there are strategies that can be used to minimize run-time power:

- Control CPU and peripheral clock rate - The CPU and peripheral clocks run at the same rate and are controlled by the System Control (SYS_CNTL) Register in the CRM (see Section 5.9.1, “System Control (SYS_CNTL)”). The clock can sometimes be slowed down if the application does

not require high throughput. Figure 3-22 illustrates typical device current versus CPU clock rate (radio inactive and peripheral inactive). Rate of current change is approximately 0.11 mA/MHz.

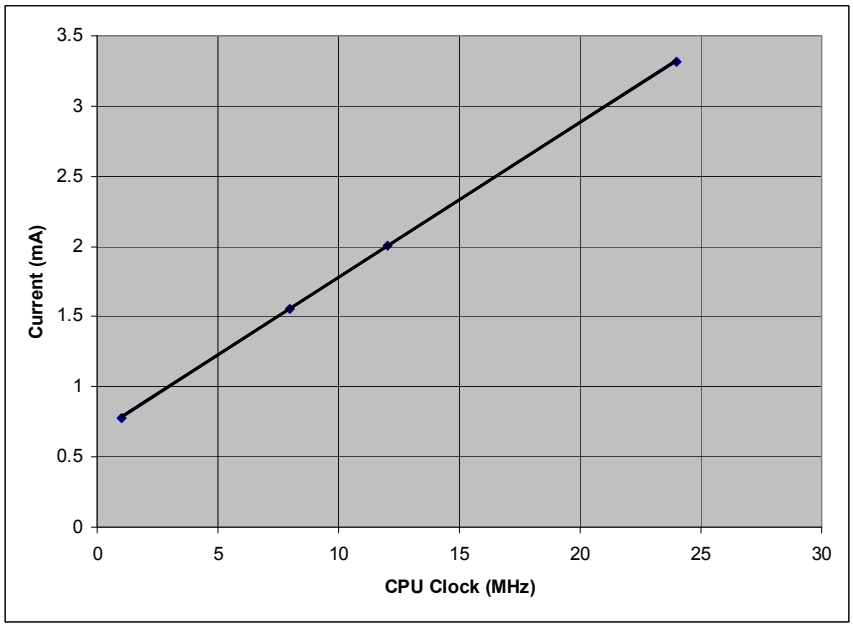



**Figure 3-22. Current versus CPU Clock Rate**

**NOTE**

- If the radio is active, i.e., an RX or TX sequence is to be in progress, there is a minimum 2 MHz required clock rate for the MACA and DMA to work properly.
- Reduced clock rate for radio operation is not recommended operation because the software may need to respond to an abnormal event; keeping the CPU clock at 24MHz for radio operation is suggested and use of bus-steal is recommended as a better choice.

- Enable bus-steal module for active radio sequences - The MC1322x has a unique bus-steal module (see Section 5.9.4, “Bus Stealing Control (BS_CNTL)”) to minimize MCU current drain during radio operation and save overall power. The MCU data bus has three bus masters that include the MACA DMA function (for moving packet data during a RX or TX), the CPU core, and the bus-steal module. The DMA works on a cycle-steal basis and has highest priority (it is also the reason there is a minimum required clock rate during an active radio sequence). Under normal operation, the CPU dominates the data bus usage and releases it as required by the DMA. If enabled, the bus-steal module is only active during a radio operation and “steals” clock cycles from the CPU. This allows the clock to run at a higher rate but not burn as much power because the CPU is halted for the programmed number of cycles on a cyclical basis. The CPU can also be halted to wait for an interrupt while the bus-steal is active.
- Enable peripheral functions as required - All peripherals can be enabled individually. Enable peripherals only as required. One caution is that a given peripheral may require a higher peripheral clock rate than would be set a minimum required rate during TX or RX.

## 3.11 MC13224 and MC13226 Device Differences

The MC13224V is the original device version and the MC13226V is the more recent version. The MC13226V varies only in ROM content and has a different device ID.

### 3.11.1 MC1322x Device

The MC1322x devices provide a version or identification code that designates the device type. The ID code is found in the MC1322x_ID Register (MACA_MC1322x_ID) at Address 0x8000_4018 located in the MACA Register memory map (see Section 9.7.6, "MC1322X_ID Register (MACA_MC1322x_ID)").

- The MC13224 ID code reads 0x0000_0009
- The MC13226 ID code reads 0x0000_0011

### 3.11.2 ROM Variations

These devices vary only in ROM content, where the MC13226V ROM is updated primarily to make more RAM space available for application code.

- The MC13226V must be used for ZigBee-2007 Profile 2 (Pro) applications because the MC13224 RAM space does not support this larger code footprint
- The MC1322x device ROM components are listed in Table 3-7. Differences in component location are highlighted
  - Three lesser used components including the ADC, LCD Font, and SSI drivers, were removed from ROM on the MC13226V to make room for BeeStack functions. These components are still available as library functions, but now compile into the RAM space
  - BeeStack functionality that previously resided in RAM on the MC13224V was moved to ROM in the MC13226V
  - Both devices support IEEE 802.15.4 full function devices with the exception of no beacon or GTS capability

**Table 3-7. MC1322x Device ROM Components**

| ROM Component | MC13224V | MC13226V |
|---|---|---|
| MAC 2003 | ROM | ROM |
| BeeStack Functions | **RAM** | ROM |
| ADC driver | ROM | **RAM** |
| ASM driver | ROM | ROM |
| CRM driver | ROM | ROM |
| GPIO driver | ROM | ROM |
| I2C driver | ROM | ROM |
| ITC driver | ROM | ROM |
| LCD Font | ROM | **RAM** |

**Table 3-7. MC1322x Device ROM Components**

| ROM Component | MC13224V | MC13226V |
|---|---|---|
| NVM driver | ROM | ROM |
| SSI driver | ROM | **RAM** |
| Timer driver | ROM | ROM |
| UART driver | ROM | ROM |
| Interrupt driver | ROM | ROM |

- An application generated for the MC13224 cannot run on the MC13226. The project must be upgraded using the Freescale BeeKit Wireless Connectivity Toolkit to the latest Codebase that supports the MC13226. See application note AN3819, *Methods for Upgrading Freescale BeeStack® Codebases*, for details on how to upgrade a BeeKit solution and its projects to a new Codebase

## 3.12 Bootloader

The bootloader is stored in the MC1322x onboard ROM. After exiting a reset, the CPU starts executing from the onboard ROM to boot the device.

**NOTE**

The bootloader is generally described in this section. Detailed information on procedures and data format are given in Appendix C, "Bootloader Reference".

### 3.12.1 Overview

Stored in the ROM, a very simple bootstrap program is executed out of reset. The most common mode of operation is that a valid target image has been stored in the onboard FLASH, the FLASH contents get transferred into RAM, and the actual application starts executing from RAM. This bootstrap will only load a configurable number of consecutive bytes from the FLASH into the RAM area beginning at address 0x0040_0000.

If the exit from reset has resulted from wakeup from a sleep mode (versus a hardware reset), the bootstrap will vector to the start of RAM and begin executing from there. The lowest page of RAM is always saved in low power operation.

If a valid FLASH image is unavailable, the bootstrap can load a configurable number of consecutive bytes from a secondary boot source into the beginning of the RAM area (0x0040_0000). The secondary boot source options are:

- Load the target image from UART1 port
- Load the target image from the SPI port (as slave) attached to a master device
- Initiate the SPI port as a master and load target image from external slave (serial FLASH)

**NOTE**

During boot, the JTAG/Nexus debug port gets enabled for all operations other than a secure mode (protected) FLASH image. The debug mode can always take command of the system, as would be common during development/debug operations.

- Load the target image from $I^2C$ (serial EEPROM)

### 3.12.2 Exception Vectors

The ARM7TDMI-S processor expects that its exception vectors reside at memory addresses 0x0000_0000 to 0x0000_001F (see Section 7.9.9, “Exception Vectors”). However, the ROM is mapped into the low memory address space (0x0000_0000 through 0x0001_3FFF), and as a result, the exception vectors also reside in ROM and are hard coded.

To allow the user to place exception service routines at his desired RAM location, the ROM exception vector entries redirect or jump program execution to the bottom of RAM (0x0040_0000 to 0x0040_001F), see Section 4.4, “Exception Vectors”. As an example, the normal undefined instruction vector (0x0000_0004) will jump to the RAM address 0x0040_0004, and the user should treat address 0x0040_0004 as his undefined instruction vector.

**NOTE**

The bootstrap will starting loading code into 0x0040_0000. The beginning of the code must be treated as exception vectors!

### 3.12.3 Bootstrap Flow

The Figure 3-23 shows the flow chart for the bootstrap. The flow provides recovery from low power modes as well as reset.

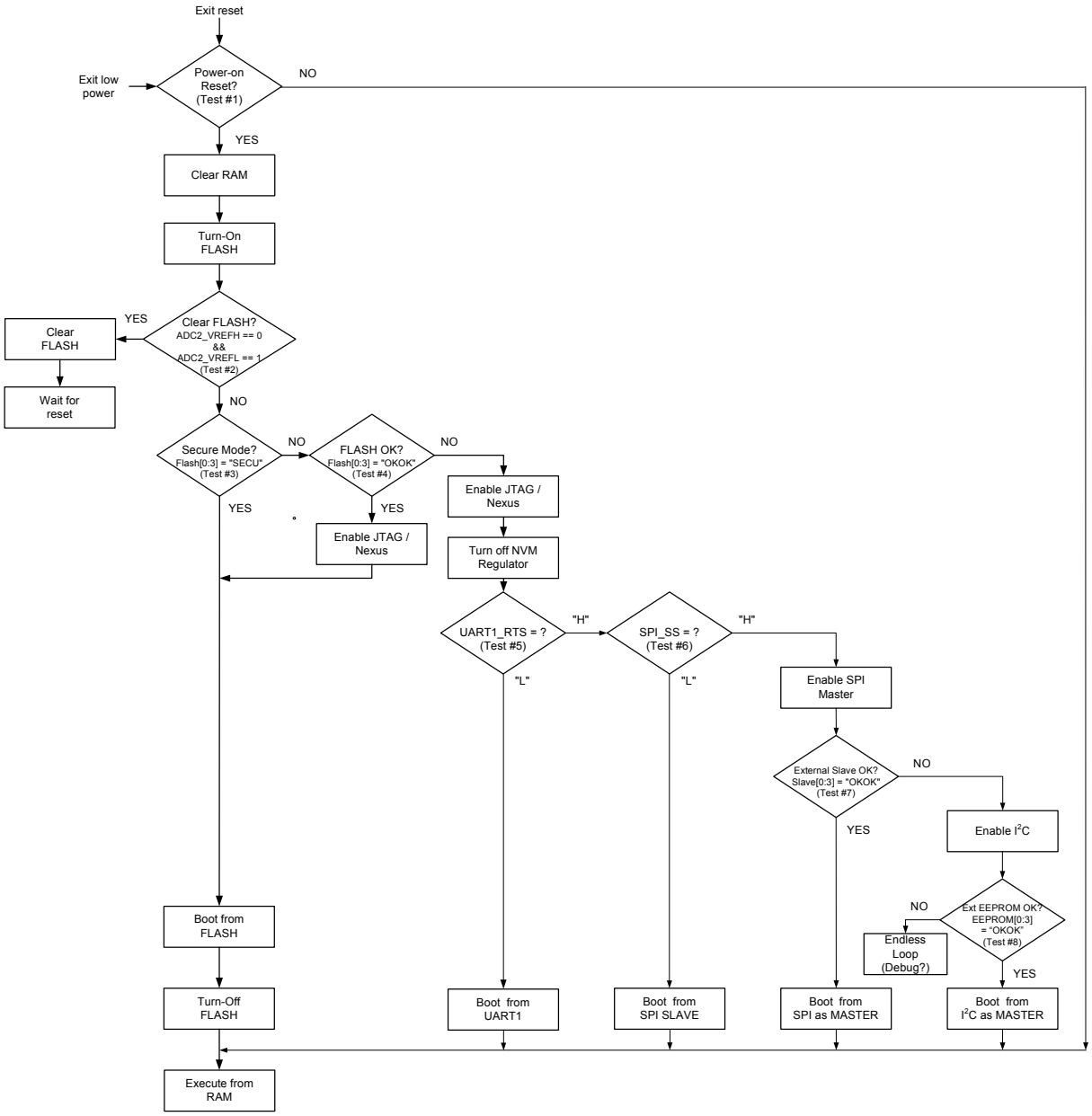



**Figure 3-23. ROM Bootstrap Flow Chart**

### 3.12.3.1 Flow Description

The bootstrap flow is entered as the result of exiting a low power mode or exiting reset:

1. TEST #1 - Is entry from a power-on reset (POR) condition?
   - NO: Proceed to execute from RAM - This action is the result of wakeup from Hibernate or Doze modes. The program jumps directly to the beginning of the RAM and starts executing from there. The lowest RAM sector PAGE0 (address 0x0400_0000) is always kept powered during Hibernate or Doze. It is the responsibility of the programmer to provide proper RAM and MCU recovery based on the power down options (such as how much RAM has been kept powered and how much has to be reloaded).
   - YES: Proceed with the boot flow. -
     - Clear RAM
     - Turn-on (power-up) the serial FLASH
     - Go to TEST #2
2. TEST #2 - Clear FLASH? (Two signals are tested for their input state to see if the condition is met, i.e., ADC2_VREFH == 0 && ADC2_VREFL == 1. See Section 3.12.5, “FLASH Erase or Recovery Mode”)
   - YES: Proceed to clear FLASH
     - Clear FLASH contents
     - Wait for reset
   - NO: Go to Test #3
3. Test #3 - FLASH in secure mode? (The contents of FLASH [0:3] are tested for the ASCII value “SECU”. See Section 3.12.6, “Secure Mode”)
   - YES: Proceed to boot from FLASH and execute. Do not enable debug to allow examination of FLASH
     - Load RAM from FLASH (boot). See Section 3.12.4, “Booting from FLASH”
     - Disable FLASH (turn off NVM regulator)
     - Jump to RAM and execute
   - NO: Go to Test #4
4. Test #4 - FLASH is OK? (The contents of FLASH [0:3] are tested for the ASCII value “OKOK”. The contents of the FLASH are valid for boot, but not secured.)
   - YES: Enable debug, proceed to boot from FLASH and execute.
     - Enable JTAG/Nexus
     - Load RAM from FLASH (boot). See Section 3.12.4, “Booting from FLASH”
     - Jump to RAM and execute
   - NO: Enable debug and continue
     - Enable JTAG/Nexus
     - Disable FLASH (turn off NVM regulator)
     - Go to Test #5

5. Test #5 - Boot from UART1? (The GPIO signal UART1_RTS is tested for its input state, i.e., UART1_RTS = ?)
   - LOW (yes)
     - Proceed to load RAM from UART1
     - Jump to RAM and execute
   - HIGH (no): Go to Test #6
6. Test #6 - Boot from SPI Slave Mode? (The GPIO signal SPI_SS is tested for its input state, i.e., SPI_SS = ?)
   - LOW: An external SPI Master is present
     - Place SPI port in Slave mode
     - Proceed to load RAM from onboard SPI Slave port
     - Jump to RAM and execute
   - HIGH: Attempt boot from external SPI Slave (typically external FLASH)
     - Place onboard SPI port in Master mode
     - Read four bytes
     - Go to Test #7
7. Test #7 - Is external SPI Slave present and are data correct? (The data are tested for the ASCII value “OKOK”)
   - YES: Proceed to boot from external FLASH and execute
     - Proceed to load RAM through onboard SPI Master
     - Jump to RAM and execute
   - NO: Attempt boot from external $I^2C$ EEPROM
     - Enable $I^2C$ as Master
     - Read four bytes data from external slave
     - Go to Test #8
8. Test #8 - Is external EEPROM present and are data correct? (The data are tested for the ASCII value “OKOK”)
   - YES: Proceed to boot from external EEPROM and execute
     - Proceed to load RAM through $I^2C$
     - Jump to RAM and execute
   - NO: Enter hold loop
     - Enter endless loop
     - Either reset or debug port is required to restore control

**NOTE**

- Interrupts are disabled during boot.
- After code is loaded into RAM, execution jumps to address 0x0400_0000 and starts from there.

- After exiting boot, GPIO pins are not all in the default state, but retain any setup values from boot. See Section 3.12.7, “GPIO Function Upon Boot Exit”
- If the device boots from an external source, it is normal procedure to write a valid application image to FLASH. It is the user’s responsibility to implement this function.

### 3.12.4 Booting from FLASH

After FLASH has been determined to be valid (through the “SECU” or “OKOK” bytes) the boot flow loads the subsequent four bytes (32-bit, little-endian), and uses that as a length field parameter of how many bytes to load into RAM. The boot flows then loads the next consecutive “length” number of bytes from FLASH into RAM starting with the first RAM address (0x0040_0000). Once the code is loaded into RAM, the FLASH is disabled (the NVM regulator is turned-off), and the bootstrap flow jumps to the first RAM address (0x0040_0000) and continues execution from there.

Table 3-8 illustrates a valid FLASH memory image.

**Table 3-8. MC1322x Valid FLASH Image Format**

| Byte Add (Dec) | Byte Value | Use |
|---|---|---|
| 0 | K/U | Validation Code |
| 1 | O/C | |
| 2 | K/E | |
| 3 | O/S | |
| 4 | Length[7:0] | Length Parameter |
| 5 | Length[15:8] | |
| 6 | Length[23:16] | |
| 7 | Length[31:24] | |
| 8 | xx | ARM Execution Code |
| 9 | xx | |
| 10 | xx | |
| ... | xx | |
| ... | xx | |
| $7_{dec}$ + $Length_{dec}$ | xx | |

## 3.12.5 FLASH Erase or Recovery Mode

In the course of software development or in the field, the FLASH contents may need to be erased and then updated with a new image. To accomplish this update, use the standard JTAG development tools. However, this may not be practical for products in the field.

An alternative to using the the JTAG port is to erase the FLASH through the boot process. This can be achieved as follows:

- ADC2_VREFH is enabled as an input with pull-up resistor, and ADC2_VREFL is enabled as an input with pull-down resistor.
- These pins are then tested for ADC2_VREFH == 0 && ADC2_VREFL == 1. The user forces this condition typically through jumpers, and this condition being true forces the FLASH to be erased.
  - — Figure 3-24 shows a typical hardware configuration with jumpers to erase FLASH.
  - — A similar function can be accomplished under control of a secondary host MCU for updating FLASH under control of the host MCU.
- When executing this option, wait a few seconds for the FLASH to be cleared, and then set ADC2_VREFH = 1 and ADC2_VREFL = 0 (non-recovery mode) before resetting the device.
- The FLASH is now cleared, and with a reset, the MC1322x will follow the boot flow.

**NOTE**

When the FLASH is erased, the top 4 Kbyte sector is left untouched. This sector is reserved for production use.

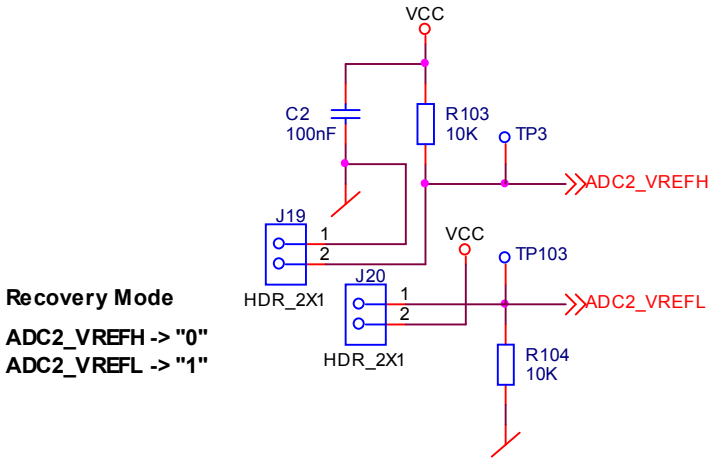



**Figure 3-24. Typical FLASH Erase Circuit**

Once the FLASH is erased, the boot flow can be used to load a new executable binary through one of the serial communication ports (see Figure 3-23), and in turn, this application can then be used to acquire and load a new binary image into FLASH.

**NOTE**

Freescale application note AN3860, *MC1322x Flash Loader Utility (Second Stage Loader)* describes this process and example software tools are provided. The user can modify these tools to suit their needs.

### 3.12.6 Secure Mode

In order to prevent the FLASH contents from being unnecessarily dumped, or otherwise altered through use of the JTAG/Nexus interface(s), the JTAG/Nexus pinouts are disabled when the device is powered up. Only if the bootstrap determines that the FLASH is not secured ("SECU" not found), will it enable the JTAG/Nexus interfaces; otherwise they remain disabled.

Once the device is secured, and the JTAG/Nexus interfaces remain disabled, they can only be re-enabled by forcing the FLASH to be cleared (i.e., enter recovery mode).

### 3.12.7 GPIO Function Upon Boot Exit

Depending on the boot option, certain GPIO are left in an altered mode (from default) when exiting boot. The following summary lists modified GPIO based on boot option.

- Boot from internal flash.
  - GPIO_38 (ADC2_VRefH) - GPIO mode, input, read from pad, pullup
  - GPIO_39 (ADC2_VRefL) - GPIO mode, input, read from pad, pulldown
- Boot from UART1.
  - GPIO_38 (ADC2_VRefH) - GPIO mode, input, read from pad, pullup
  - GPIO_39 (ADC2_VRefL) - GPIO mode, input, read from pad, pulldown
  - GPIO_14 (UART1_TX) - Function1 mode
  - GPIO_15 (UART1_RX) - Function1 mode, pullup
  - GPIO_16 (UART1_CTS) - Function1 mode
  - GPIO_17 (UART1_RTS) - Function1 mode, pullup
- Boot from SPI (master or slave).
  - GPIO_38 (ADC2_VRefH) - GPIO mode, input, read from pad, pullup
  - GPIO_39 (ADC2_VRefL) - GPIO mode, input, read from pad, pulldown
  - GPIO_17 (UART1_RTS) - GPIO mode, input, read from pad, pullup
  - GPIO_4 (SPI_SS) - Function1 mode, pullup
  - GPIO_5 (SPI_MISO) - Function 1 mode
  - GPIO_6 (SPI_MOSI) - Function 1 mode
  - GPIO_7 (SPI_SCK) - Function 1 mode
- Boot from $I^2C$ EEPROM
  - GPIO_38 (ADC2_VRefH) - GPIO mode, input, read from pad, pullup
  - GPIO_39 (ADC2_VRefL) - GPIO mode, input, read from pad, pulldown
  - GPIO_17 (UART1_RTS) - GPIO mode, input, read from pad, pullup
  - GPIO_4 (SPI_SS) - Function 1 mode, pullup
  - GPIO_5 (SPI_MISO) - Function 1 mode
  - GPIO_6 (SPI_MOSI) - Function 1 mode
  - GPIO_7 (SPI_SCK) - Function 1 mode
  - GPIO_12 (I2C_SCL) - Function 1 mode
  - GPIO_13 (I2C_SDA) - Function 1 mode

### 3.12.8 Bootstrap version.

The Bootstrap ROM Software Version is located at address 0x0000_0020 in ROM and has four bytes.

# Chapter 4
# Memory

## 4.1 Introduction

The MC1322x ARM7 core has a register-based instruction set and the CPU registers are not located in the memory map. The on-board memory resources include:

- 128 Kbyte serial FLASH memory (that is mirrored into RAM)
- 96 Kbyte SRAM ($98304_{dec}$ bytes actual)
- 80 Kbyte ROM
- MCU peripheral registers and buffers
- Transceiver control registers

## 4.2 Features

- 96 Kbyte SRAM in 4 blocks; addressed contiguously
  - RAM0: 8 Kbytes, 2 Kwords (2048 x 32 bits)
  - RAM1: 24 Kbytes, 6 Kwords (6144 x 32 bits)
  - RAM2: 32 Kbytes, 8 Kwords (8192 x 32 bits)
  - RAM3: 32 Kbytes, 8 Kwords (8192 x 32 bits)
- All read or write accesses require a minimum of two system clock cycles
- Stall signal generated for read after write cycles
- SRAM have been divided into blocks to allow for power savings.
  - Block RAM0 is always powered.
  - While sleeping, 1, 2, or 3 of the remaining RAM blocks can be turned off.
  - While sleeping, powered RAM blocks are placed in a low voltage mode for data retention.
  - As more RAM blocks are turned on, more current is required. The disadvantage is that boot time can be longer as required to reload RAM from FLASH.
- 80 Kbyte ROM
  - 20 Kwords (20480 x 32 bits)
  - Contains bootstrap code, 802.15.4 MAC (no security) and drivers. The MAC software builds on the lower level hardware capability of the transceiver and MACA. All code except the bootstrap is “patchable”.
- Serial FLASH (NVM)
  - 128 Kbyte

- Accessed via a dedicated SPI module designated as the SPIF. Freescale provides an access driver.
- The FLASH erase, program, and read access are programmed through the SPIF port.
- Program code gets mirrored into SRAM for execution. The FLASH is accessed at boot time to load/initialize RAM. All CPU program access is from RAM or ROM.
- Highest 4 Kbyte sector is reserved for factory use.
- Write accessible under program control.
  - Use provided driver.
  - Can also be used for non-volatile parameter, look-up table, and optional program code storage

## 4.3 Memory Map

The MC1322x summary memory map is shown in Table 4-1. In this table only the base address of the various modules is shown. For a fully detailed memory map of peripheral registers, See Appendix A, "MC1322x Register Address Map". Also, in each module chapter the base address and register address table for the associated module is given.

**Table 4-1. MC1322x Top Level Memory Map**

| Base Address | Module | Size (Bytes) | Type |
|---|---|---|---|
| 0x0000_0000 | ROM | 80k | R |
| 0x0040_0000 | SRAM | 96k | R/W |
| 0x0040_0000 | RAM0 | 8k | R/W |
| 0x0040_2000 | RAM1 | 24k | R/W |
| 0x0040_8000 | RAM2 | 32k | R/W |
| 0x0041_0000 | RAM3 | 32k | R/W |
| 0x8000_0000 | GPIO | | R/W |
| 0x8000_1000 | SSI | | R/W |
| 0x8000_2000 | SPI Port | | R/W |
| 0x8000_3000 | Clock and Reset Module (CRM) | | R/W |
| 0x8000_4000 | MAC Accelerator (MACA) | | R/W |
| 0x8000_5000 | UART1 | | R/W |
| 0x8000_6000 | $I^2C$ | | R/W |
| 0x8000_7000 | TIMER | | R/W |
| 0x8000_8000 | Advanced Security Module (ASM) | | R/W |
| 0x8000_9000 | Modem Synthesizer Write Functions | | R/W |
| 0x8000_9200 | Modem Transmit Sequence Manager | | R/W |
| 0x8000_9400 | Modem Radio Receiver Functions | | R/W |

**Table 4-1. MC1322x Top Level Memory Map (continued)**

| Base Address | Module | Size (Bytes) | Type |
|---|---|---|---|
| 0x8000_9600 | Modem Radio Transmitter Functions | | R/W |
| 0x8000_9800 | Modem Radio Frequency Synthesizer | | R/W |
| 0x8000_9A00 | Modem Tracking Oscillator Controller | | R/W |
| 0x8000_A000 | Radio Analog Write Functions | | W |
| 0x8000_A000 | Radio Analog Read Functions | | R |
| 0x8000_B000 | UART2 | | R/W |
| 0x8000_C000 | Reserved | | - |
| 0x8000_D000 | ADC Module | | R/W |
| 0x8002_0000 | Interrupt Controller | | R/W |

## 4.4 Exception Vectors

The ARM architecture places the exception vector addresses at the bottom of the memory map, i.e., start address 0x0000_0000. With the MC1322x, the on board ROM is located at address page 0x0000_0000 and does not allow the user to change vector contents. To overcome this issue, the ROM redirects the vector table to the base of the RAM at address 0x0040_0000. Table 4-2 shows the exception vector addresses with both the original address and the redirected user address in RAM.

**Table 4-2. MC1322x ARM Exception Vectors**

| ROM Address | Exception | RAM Address | Notes / Usage |
|---|---|---|---|
| 0x0000_0000 | Reset (boot start) | 0x0040_0000 | Start RAM address after boot. |
| 0x0000_0004 | Undefined instruction | 0x0040_0004 | Supported |
| 0x0000_0008 | Software interrupt | 0x0040_0008 | Supported |
| 0x0000_000C | Abort (Prefetch) | 0x0040_000C | Supported[1] |
| 0x0000_0010 | Abort (Data) | 0x0040_0010 | Supported[1] |
| 0x0000_0014 | Reserved | 0x0040_0014 | Reserved |
| 0x0000_0018 | IRQ | 0x0040_0018 | Supported |
| 0x0000_001C | FIQ | 0x0040_001C | Supported |

[1] Supported for RAM access and UART modules; not supported by all other peripherals

**NOTE**

Address 0x0000_000 is still the reset vector in ROM, and execution starts here in ROM for the boot process. Address 0x0040_0000 becomes the start address of execution from RAM after boot (reset or restart from a low power condition).

## 4.5 ROM and RAM

The MC1322x executes all program out of the 80 kbyte ROM or the 96 kbyte application SRAM. The ROM is factory programmed and contains boot code, drivers, utilities, and different communication protocol services. After recovery from a reset condition, execution always starts from address 0x0000_0000.

The RAM is broken into 4 pages for the purpose of allowing different options for power-down. Page RAM0 is 8192 bytes and is always powered under a sleep condition. RAM1, RAM2, and RAM3 pages are 24576, 32768, and 32768 kbytes respectively. These can be kept powered or un-powered in sleep mode.

The RAM is loaded from the serial FLASH at boot time and any powered-down page must be re-loaded after wake-up from sleep (Hibernate or Doze). Recovery time can vary greatly as a result. After wake-up, execution always starts at address 0x0040_0000. The recovery routine for reloading powered-down SRAM must reside in page RAM0.

Figure 4-1 shows the MC1322x ROM and SRAM memory map.

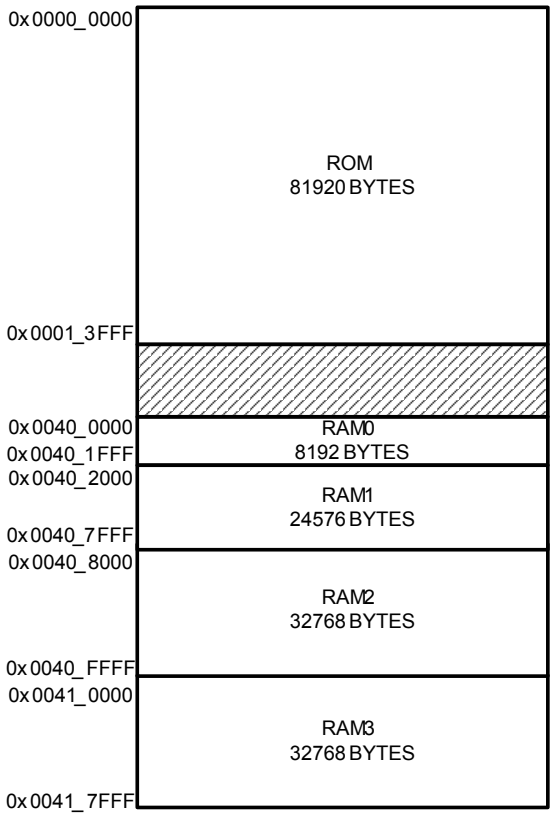



**Figure 4-1. Memory Map (Mem Module, 80 kbyte ROM and 96 kbyte RAM)**

## 4.6 Serial FLASH (NVM)

The MC1322x non-volatile memory (NVM) is the a 128 Kbyte serial FLASH. The serial FLASH is accessed via a dedicated SPI module designated as the SPIF. The FLASH erase, program, and read capability are all programmed through the SPIF port. The valid FLASH contents are accessed at boot time to load/initialize RAM for program execution. The FLASH is also accessed to re-initialized/restore RAM that had been un-powered during sleep.

With 96 kbytes of RAM and 128 kbytes of FLASH, there is 32 Kbytes of excess capacity for other useful purposes than the main program code. The top 4 kbyte sector is reserved for factory use. The additional 28kbytes is sufficient capacity for non-volatile data store and optional application profiles if desired.

Some of the characteristics of the serial FLASH include:

- SPI serial interface with 13 MHz maximum data clock
- Accessed via a dedicated SPI module designated as the SPIF.
- 15 mA erase, program, or read current
- 5 µA maximum standby current
- 75 ms sector or block erase duration
- 150 ms chip erase duration
- Accessible under program control

### 4.6.1 FLASH Access

A driver for FLASH access (erase, program, and read) is available for integration with the user's application. See Appendix B, "MC1322x Software Driver Utilities".

### 4.6.2 FLASH Security

The FLASH is accessed during the boot process. The FLASH must contain valid data and it also can be made secure for the boot process as detailed in Section 3.12.6, "Secure Mode".

The JTAG/Nexus interfaces are disabled when the MC1322x is powered up. Only if the bootstrap determines that the FLASH is not secured, will it enable the JTAG/Nexus interfaces; otherwise they remain disabled.

The first four bytes of the FLASH are reserved for a validation word that is used to verify that the FLASH contains valid data, as well, to verify if the FLASH is secure. If the validation word is "OKOK" the data are valid, but not protected. If the validation word is "SECU" the data are valid AND secure.

Once the device is secured and the JTAG/Nexus interfaces remain disabled, they can only be re-enabled by forcing the FLASH to be cleared (i.e., enter recovery mode).

### 4.6.3 FLASH Recovery (Erase)

As detailed in Section 3.12.5, “FLASH Erase or Recovery Mode”, the FLASH can be erased via the boot process to be “recovered” to a known clear condition. Also, erasing parts or all of FLASH through program control is possible.

## 4.7 Peripheral Register Access

The MCU peripherals control and status registers vary in register width and allowed access type.

- 32-bit write access is always allowed.
- Some registers are 32-bit write access only. Reference the individual register descriptions for limitations as to 8-bit or 16-bit write access.
- 8-bit, 16-bit, and 32-bit read accesses are always allowed. Any access will always provide 32-bits of read data, and all bits/fields not present in the register will default to “0” on the data bus.

# Chapter 5
# System Management (Including CRM)

## 5.1 Management Overview

This chapter details the management of the PiP including reset, power management, wake-up from low power, clock control, and KBI interface. Management is controlled by the Clock/Reset/Power Module (CRM) which is a dedicated unit designed to handle all top level clock, reset and power functions for the MC1322x. The CRM controls the voltage regulators on MC1322x for power management. The CRM also contains a small block called the Sleep Module that runs when the rest of the device entirely powered down and it keeps time during sleep and wake modes.

### 5.1.1 Management Features

- Manages system reset and available software initiated system reset
- Controls clock gating for power savings
- Power management of internal regulated voltage sources including buck regulator
- Real Time Clock (RTC)
- Sleep mode management
  - — Two types of sleep: Hibernation or Doze
  - — Controlled power down
  - — Programmable degree of power-down
  - — Critical programmed values are retained during sleep
- Wake-up mode management
  - — Chip is gracefully powered up.
  - — Clocks are automatically turned on.
  - — wake-up available by programmable wake-up and RTC timers.
  - — wake-up available by external interrupts from KBI pads.
- Watchdog (COP) surveillance timer
- Bus stealing mode to keep ARM quiet during idle periods.
- Management of ring oscillator and optional 32.768 KHz oscillators
- Management of reference oscillator
- Management of MCU core and peripheral clocks

## 5.1.2 System Reset

The MC1322x provides complete management of system reset and recovery/initialization from reset. There are several sources of reset:

- Hardware reset signal RESETB - The reset input is an active low, asynchronous signal. Holding RESETB low causes the lowest power mode for the device (typically less than 300 nA) where all circuitry is disabled except for the CRM Sleep Module. There is no onboard pull-up resistor attached to RESETB to further reduce current. Upon release of RESETB to high, the device recovers to a run condition.
- Power-on reset (POR) - In a wireless node where the MC1322x is the main controller, RESETB can be tied to VBATT so that the device powers-up as VBATT becomes valid. To guarantee a proper start-up procedure, there is a power-on reset circuit that holds the equivalent of an active reset signal until the VBATT voltage reaches a minimum threshold voltage. The POR will then be released to allow the device start-up sequence to proceed.

**NOTE**

The external RESETB is tied into the POR such that after RESETB is released, the POR circuit must also release to allow start-up to proceed.

- Software reset - The CRM has the Software Reset Register (address 0x8000_3050) where writing to the register can cause a system reset. The affect is the same as exiting a POR and starting a start-up sequence.
- Waking from Hibernate or Doze modes without MCU state retention - One option for Hibernate or Doze modes is to NOT retain all the MCU, Modem, and Analog Control states. This is controlled by the MCU_RET Bit 6 of the Sleep Control Register (address 0x80003008). If the MCU retention bit is not set, then the wake-up from low power mode(s) resets the MCU.

**NOTE**

- After exit from an MCU reset or any other reset initiated start-up sequence, program execution begins at the reset boot vector address 0x0000_0000.
- After exit from a wake-up sequence with an MCU reset, program execution begins at address 0x0040_0000, the beginning of RAM.

## 5.1.3 System Clocks

This section provides a brief overview of the system clocks.

### 5.1.3.1 Clock Sources

The MC1322x has 3 possible clock sources, i.e., the reference oscillator (13 - 26 MHz with 24 MHz default), an onboard self-contained 2 kHz ring oscillator, and an optional 32.768 khz crystal oscillator.

- Reference oscillator - is the primary clock source for the device
  - — Default frequency is 24 MHz, although a frequency from 16 MHz to 26 MHz can be used.
  - — Normally a crystal is used with the onboard amplifier although an external source can be used.

— Frequency accuracy of +/-40 ppm for IEEE 802.15.4 compatibility must be maintained over all conditions.
— Section 3.6, “Reference Oscillator” describes detailed use of the oscillator.
— The frequency of the crystal oscillator can be trimmed by changing onboard load capacitance. Section 5.9.16, “Reference XTAL Control (XTAL_CNTL)” describes the control register that adjust capacitive loading.
— Is divided by 128 and used as the wake-up timer source during Doze mode.

- 2 kHz ring oscillator - is the default oscillator for the Real Time Clock (RTC) and the Sleep Module when in or waking-up from Hibernate or Doze. The specified default frequency accuracy of the oscillator over temperature is from 1 kHz to 4 kHz. However, onboard resources are available to trim the oscillator accuracy.
- Optional 32.768 KHz crystal oscillator - is available with the addition of an external 32.768 KHz crystal (see Section 3.7, “32.768 kHz Crystal Oscillator (use optional)”). This oscillator typically replaces the use of the ring oscillator if a low power, high accuracy source is desired for wake-up and the RTC.

### 5.1.3.2 Main System Clock Distribution

All main system clocks are derived from the reference oscillator. Figure 5-1 shows a simplified diagram of the MC1322x clock distribution. The diagram does not illustrate clock power-down options.

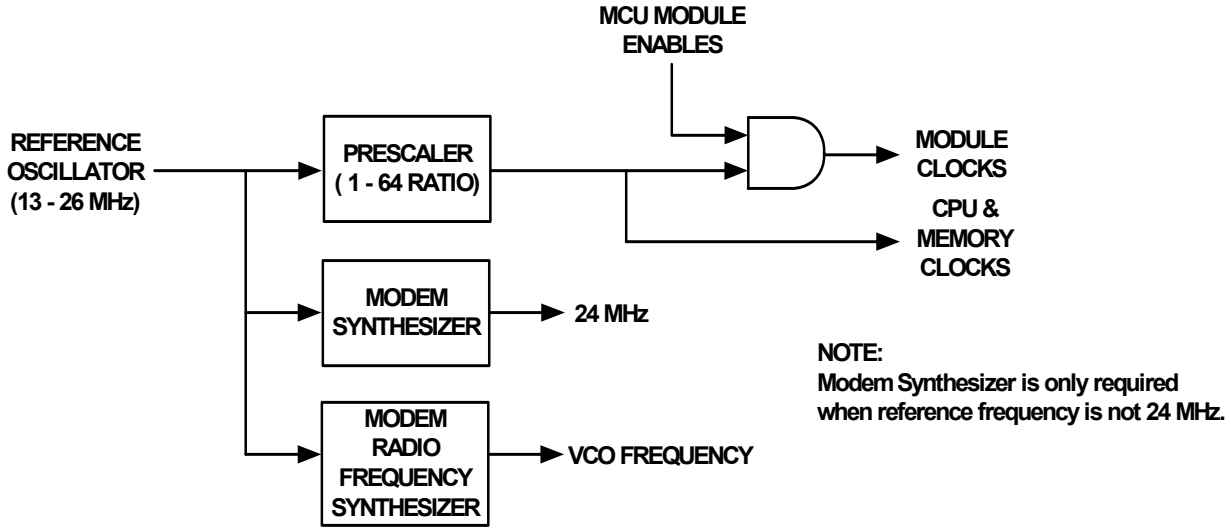



**Figure 5-1. MC1322x Simplified Clock Distribution**

The reference oscillator is normally 24 MHz, but can be from 13 - 26 MHz

- The CPU, memory, and MCU module clocks are all derived from a programmable divider (prescaler)
  — Prescaler divide ratio is set by the **xtal_clkdiv[5:0]** field of the System Control Register - divide ratios programmable from 1 to 64 (default = 1). Lower core clock rates reduce power, but also reduce performance.

— MCU module clocks are enabled individually by their respective module enables - the module clock enables are located within register sets for each module. The module source clocks are further modified by the module circuitry as to baud rates, count frequencies, etc.

- The Modem Radio Frequency Synthesizer generates the radio VCO frequency - it can use any reference frequency from 13 to 26 MHz. The radio channel frequencies are set by appropriately programming the synthesizer using the known reference frequency.
- The Modem Synthesizer
  - — The modem synthesizer is only used for the receiver and only outputs 24MHz.
  - — The receiver always requires a 24 MHz reference clock and the modem synthesizer is only used to generate this 24 MHz when the reference oscillator is NOT 24 MHz.
  - — The external PLL filter is used by the modem synthesizer and is only used with a non-24MHz reference clock.

## 5.1.4 System Power Distribution

Figure 5-3 shows a diagram of power distribution. There are a number of different linear regulators that supply different functional blocks on the device. Power can be enabled/disabled to the different blocks based on the operational mode.

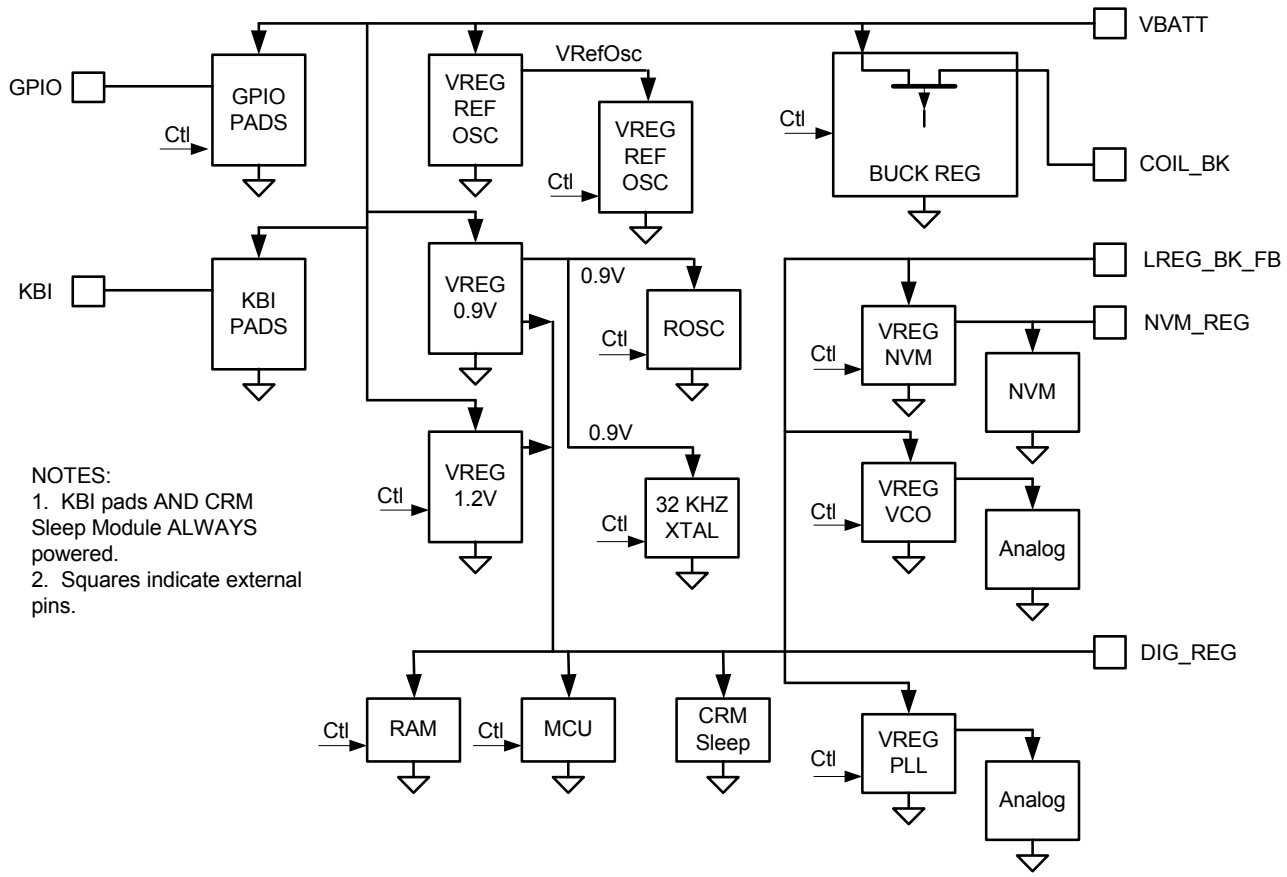



**Figure 5-2. MC1322x Power Distribution**

Consider the following:

- The KBI pads and CRM Sleep Module are always powered except when RESETB is active. The GPIO pads are not powered in reset and typically in low power modes. See Section 3.3.1, "GPIO Pin State During Reset, Default, and Low Power Modes" for a detailed description of IO in different modes.
- In sleep modes
  - The CRM Sleep Module maintains control
  - The different oscillators can be enabled (one at a time) depending on options
  - RAM0 page is always powered; RAM1, RAM2, and RAM3 can be optionally powered
- Power to the device can be connected in three separate ways, see Section 3.2.1, "Power Pin Descriptions" for details
- The BUCK REG is an optional switching regulator and is controlled via the CRM

### 5.1.5 KBI Interface Signals

The KBI signals are unique among the MC1322x GPIO signals:

- All eight KBI_7 : KBI_0 signals are powered at all times including low power modes. The KBI_7 : KBI_4 signals are always inputs during low power, and the KBI_3 : KBI_0 signals are always outputs during low power.
- The four KBI_7 : KBI_4 inputs may be used to generate asynchronous interrupt requests to the CPU during all modes of operation. The interrupt requests are programmable for sense and level or edge activation.
- The CRM controls these signals in low power, and the GPIO Controller controls these ports during normal operation

#### 5.1.5.1 KBI_7 : KBI_0 Signals in Low Power Mode

Dissimilar to the other GPIO, the eight KBI signals remained powered during low power modes. During low power operation, complete control of these signals always reverts to the CRM through the WU_CNTL Register (see Section 5.9.2, "Wake-up Control (WU_CNTL)".

- The KBI signals along with low power control circuitry are kept powered during low power modes
- KBI_3 : KBI_0 - These signals always revert to outputs whenever low power mode is enabled.
  - Default out of reset is all outputs in the high state.
  - Once the device is initiated (WU_CNTL Register), the outputs can be programmed to revert to either output high or output low in low power.
- KBI_0 - In addition to being a powered output, KBI_0 is unique in that it can be programmed as a wake-up signal to external devices. For this application, in sleep mode KBI_0 is an output low, but upon wake-up it transitions to high, and an external device can monitor the signal for a wake-up event.
- KBI_7 : KBI_4 - These signals always revert to inputs whenever low power mode is enabled.
  - Default out of reset is inputs with pull-downs enabled.

— Once the device is initiated (WU_CNTL Register), the use of the pull-ups/pull-downs is based on the selected sense of a potential interrupt request.

— These signals can be enabled to asynchronously wake-up the device and generate an interrupt request. See Section 5.1.5.2, “KBI_7 : KBI_4 Signals as External Interrupt Request Inputs”.

- Once low power is exited, these signal pads are controlled by the GPIO controller. These can also be used as general purpose IO, similar to any other device GPIO.

**NOTE**

If the application is using the KBI signals during low power operation, it is important that after wakeup the application expediently program the GPIO Controller to sustain the desired mode of operation of these IO in normal operation.

### 5.1.5.2 KBI_7 : KBI_4 Signals as External Interrupt Request Inputs

The KBI_7 : KBI_4 signals are the only device GPIO that can be used to generate external interrupt requests.

- The 4 independent interrupt requests are available in both normal and low power modes
- The control for the interrupt requests always resides in the CRM Module, WU_CNTL Register:
  — EXT_WU_IEN[3:0] are the external IRQ enable or mask bits
  — EXT_WU_POL[3:0] are the polarity select bits (active high or active low) for the IRQ sense. For low power mode, these also enable a pull-up for an active low select and a pull-down for an active high select.
- When in low power mode, the CRM retains complete control of the pads forcing them to inputs with programmable sense.
- When in normal mode (exit from low power), the GPIO Controller overrides and provides control. Default upon exiting reset is the pads as inputs with pulldowns enabled. If a different sense configuration is used in low power, then the GPIO Controller must be expediently initialized to support the alternate configuration in normal operation
- The interrupt requests may also be used to wakeup the device from low power.

## 5.2 Clock/Reset/Power Module (CRM) Overview

The following sections details functions and operations of the CRM. The CRM contains the Sleep Module with wake-up timer and Real Time Clock, status and control registers, clock controls, bus steal module, and the COP timer.

### 5.2.1 CRM Block Diagram

Figure 5-3 shows the CRM block diagram.

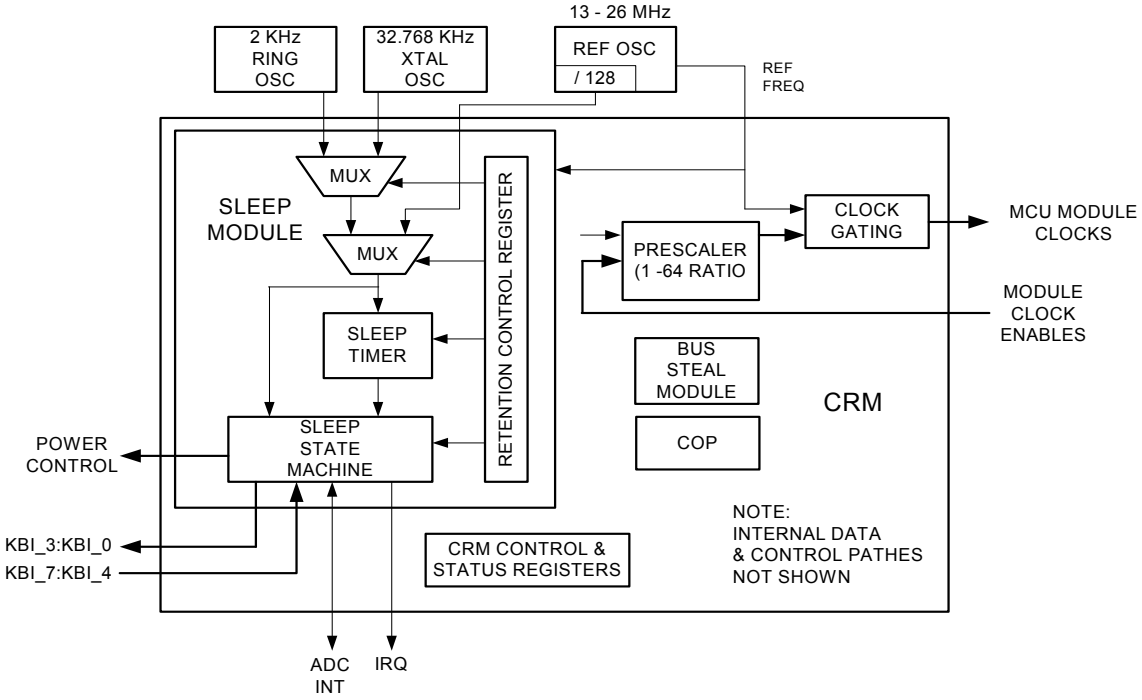



**Figure 5-3. CRM Block Diagram**

### 5.2.2 Signal Descriptions

Table 5-1 lists external signals associated with the CRM and gives their descriptions.

**Table 5-1. CRM External Signals**

| Signal Names | Direction | Active | Comments |
|---|---|---|---|
| RESETB | Digital Input | Low | Asynchronous system reset |
| KBI_0_HST_WK | Digital Input/Output (Output only for reset & low power) | High | 1) Can be used as a keyboard control output (can also be programmed as GPIO).<br>2) Can be used as a wake-up output when exiting sleep mode.<br>3) Retains power during reset and power-down. |
| KBI_3 - KBI_1 | Digital Input/Output (Output only for reset & low power) | High | 1) Can be used as a keyboard control outputs (can also be programmed as GPIO).<br>2) Retain power during reset and power-down. |

**Table 5-1. CRM External Signals (continued)**

| Signal Names | Direction | Active | Comments |
|---|---|---|---|
| KBI_7 - KBI_4 | Digital Input/Output (Input only for reset & low power) | Low | 1. When used as KBI inputs, the 4 signals can provide an asynchronous interrupt to exit sleep mode or as keyboard inputs (can also be programmed as GPIO).<br>2.) Retain power during reset and power-down. |
| XTAL_32_IN | Analog Input | - | Optional 32.768 kHz crystal oscillator input |
| XTAL_32_OUT | Analog Output | - | Optional 32.768 kHz crystal oscillator output |
| COIL_BK | Power Switch Output | - | Buck converter coil drive output |
| LREG_BK_FB | Power Input | - | 1) Voltage input to onboard regulators<br>2) Buck regulator feedback voltage |

## 5.2.3 Sleep Module

The sleep module of the CRM is always powered, integrated with the power-up sequence when exiting reset, and manages entering and exiting low power modes.

### 5.2.3.1 Modes of Operation

The Sleep Module assists in managing modes of operation.

- Reset active - this is the lowest power option for the MC1322x and only the Sleep Mode of the CRM is powered.
- Power Up from Reset

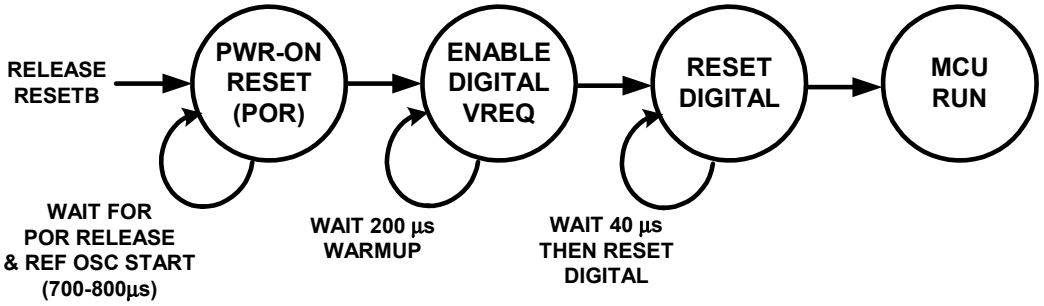



**Figure 5-4. Power-up from Reset State Diagram**

The Sleep module of the CRM performs power-up of the MCU after power on reset (POR). Figure 5-4 shows the sequence after release of RESETB. After POR, the reference oscillator is turned-on, and a wait occurs until the reference oscillator is ready (700usec). When the oscillator is ready, the full turn on of the Digital 1.2V supply can occur. A full digital reset is provided and then the reference clock (nominally 24MHz) is gated to the MCU core and the MCU core is fully functional.

- Normal Operation - In normal operation, the MCU core runs, and the MCU system clock runs at the frequency set by the prescaler (reference frequency divided by the prescaler ratio). Although not true modes, the device can run in various power configurations

  - — Run / Idle - the CPU is running and the transceiver is idling and MC1322x can quickly enable transceiver operations. The MCU core and peripheral are running with full capability.
  - — Transmit or Receive - similar to Run, this is not an actual CRM mode but is simply the MCU active and the transceiver active
    - – The MACA operation is independent of the CPU and uses DMA for moving data to/from RAM (stealing cycles away from the core). The MCUs effective run rate can be reduced lower via the CPU clock (as low as 2MHz for 802.15.4 mode)
    - – The Bus Steal Module (BSM) in the CRM can also be enabled to further reduce power
- Sleep Modes - There are two basic sleep modes, i.e., Hibernate and Doze

#### 5.2.3.2 Sleep Operation

Because low power operation is extremely important to battery operation. The MC1322x has extensive resources to manage power and provide low power or “sleep” modes. There are 2 types of sleep mode that MC1322x can enter: Hibernate and Doze.

There is only difference between modes is that Doze mode keeps the reference oscillator alive to supply the sleep timing clock. The reference frequency is divided by 128 and supplied to the counters. Alternatively, in Hibernate the ring oscillator is used as a default and if the 32 kHz oscillator is available it can be used as the alternative. The 32 kHz option provides an accurate time base at lowest current because it is used in Hibernate. Using Doze allows an accurate time base with lower cost (no 32 kHz crystal is used) but at the expense of higher current while in low power mode.

For both modes:

- Most of the system is powered off, except for the selected oscillator, timer support logic, and the RAM0 page (additional RAM can be retained).
  - — For Hibernate mode, the default oscillator is the ring oscillator. The 32 kHz crystal oscillator is an option if the crystal is present.
  - — For Doze mode the reference oscillator divided-by-128 is the clock source.
- RAM retention is programmable
  - — Memory is powered by 0.9V, but is not accessible (it just retains state).
  - — Four combinations of RAM retention are possible. Each of the pages RAM1, RAM2, and RAM3 can be also enabled.
  - — Low power current is increased as more RAM retains its state
- All the states of the MCU, Modem, and analog control can be retained.
  - — If the MCU State Retention bit (MCU_RET) is set, the states of this logic are retained
  - — Low power current is increased as the logic retains its state.
- The states of the GPIO can be retained
  - — In low power mode, the GPIO normally are disconnected from the top voltage rail except for the KBI signals. Setting the DIG_PAD_EN bit causes the GPIO to stay powered in their programmed state. The KBI signals still revert to CRM KBI control.
  - — The DIG_PAD_EN will be ignored if the MCU State retention bit is not set.

- Wake-up from sleep is possible from three sources
  - An external wake-up signal from KBI inputs KBI_7:KBI_4
  - The internal wake-up timer
  - The Real Time Clock time-out

Another option of auto ADC operation during sleep is available only in hibernate mode.

**Table 5-2. Power Modes and Clocks**

| | | CLOCKING | | | | | |
|---|---|---|---|---|---|---|---|
| **MCU Mode** | **MAX IDD Limit** | **CPU** | **RAM** | **ROM** | **Periph** | **Digital Modem** | **CRM/ Sleep** |
| | | | 2kx32<br>6kx32<br>8kx32<br>8kx32 | 20kx32 | | 24MHz from XTAL or PLL | 2kHz/ 32kHz/ Ref Osc |
| Reset | 0.3uA | OFF | OFF | OFF | OFF | OFF | OFF |
| Hibernate | 1uA[1] | OFF | OFF | OFF | OFF | OFF | 2kHz or 32kHz |
| Doze | 23uA | OFF | OFF | OFF | OFF | OFF | Ref osc/ 128 |
| Idle | 0.9mA | Ref osc | Ref osc | Ref osc | Ref osc | OFF | Ref osc |
| Run | 5mA | Ref osc | Ref osc | Ref osc | Ref osc | ON | Ref osc |
| Receive | 20mA | Ref osc | Ref osc | Ref osc | Ref osc | ON | Ref osc |
| Transmit | 20mA | Ref osc | Ref osc | Ref osc | Ref osc | ON | Ref osc |

[1] Amount of RAM retention programmable - MAX IDD limit assumes ONLY the 2kx32 is ON

### 5.2.3.3 Sleep Module Clock Structure

The Sleep Module clock structure is shown in Figure 5-5.

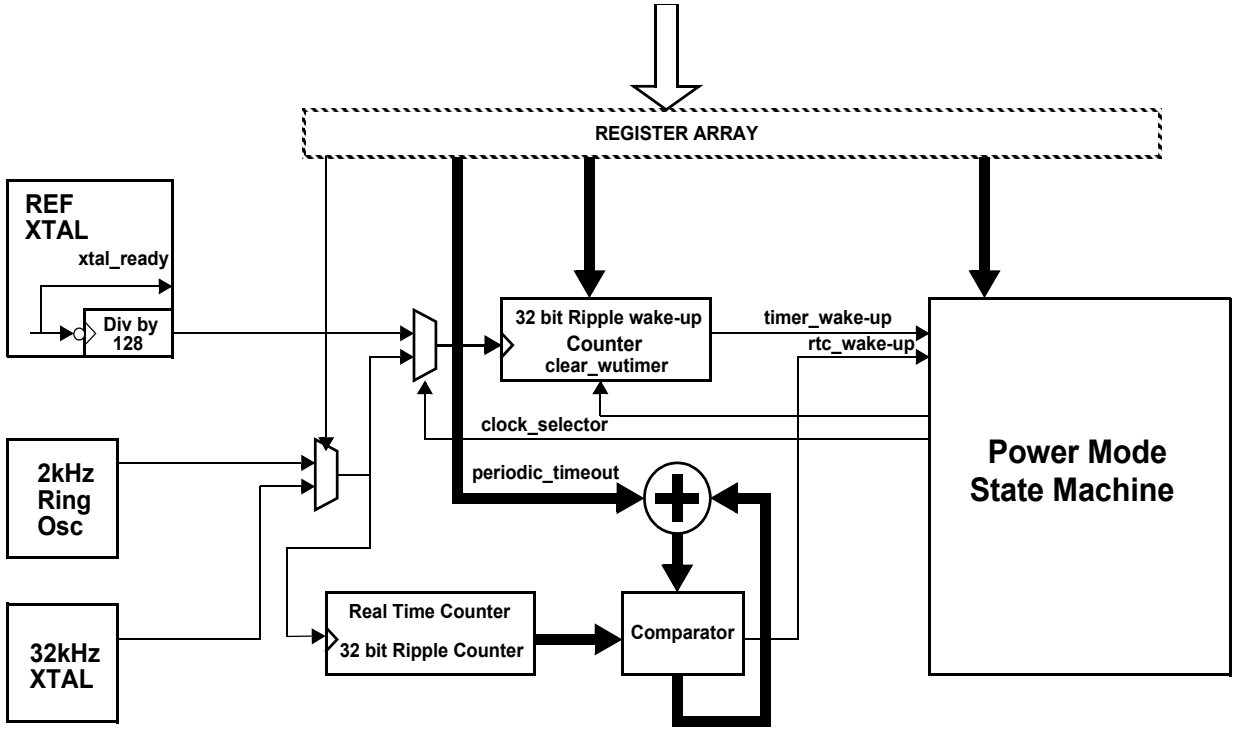



**Figure 5-5. Sleep Module Clock Structure**

The clock sources include a 2 kHz ring oscillator, an optional 32.768 crystal oscillator, and the reference oscillator frequency divided-by 128. The RTC is only driven from the ring oscillator or the 32.768 kHz crystal oscillator (if present and enabled). The wake-up timer is driven from the available low power clock source, i.e., ring oscillator or crystal oscillator in Hibernate, and alternatively, the reference oscillator divided-by 128 in Doze.

#### 5.2.3.3.1 2kHz Ring Oscillator

The 2kHz ring oscillator is the default clock source for the Sleep Module when coming out of reset or Hibernate for the Real Time Clock (RTC) and the wake-up counter. The specified default frequency accuracy of the oscillator over temperature is from 1 kHz to 4 kHz. However, onboard resources are available to trim the oscillator accuracy.

The accuracy of the ring oscillator for wake-up is typically unimportant. However, it may be desired to have a more accurate wake-up period from Hibernate. The CRM provides The Calibration Control Register (CAL_CNTL), Calibration XTAL Count Register (CAL_COUNT) and Ring Oscillator Control Register (RINGOSC_CNTL) for managing the ring oscillator.

The Ring Oscillator Control Register (see Section 5.9.15, “Ring Oscillator Control (RINGOSC_CNTL)”) provides:

- Ring Oscillator Enable bit - The default is the oscillator enabled exiting POR reset. If the 32.768 kHz oscillator is used the ring oscillator must be disabled via this bit.
- Frequency tuning (ROSC_CTUNE[3:0] and ROSC_FTUNE[4:0] fields) - The frequency of the oscillator can be adjusted by changing the loading capacitance. The ROSC_CTUNE[3:0] field (default 0x6) changes the loading in 1 pF steps and the ROSC_FTUNE[4:0] field (default 0x17) changes the loading in 160 fF steps.

The Calibration Control Register (see Section 5.9.13, “Calibration Control Register (CAL_CNTL)”) and Calibration XTAL Count Register (see Section 5.9.14, “Calibration XTAL Count (CAL_COUNT)”) provide the means for managing calibration.

- Calibration Control Register - provides the control for the calibration process. All fields typically would get written at one time
  - — Calibration Enable bit (CAL_EN) - Default is zero or off. If the bit gets written to one, the calibration process starts. Writing a zero while the calibration process is in progress will abort the process.
  - — Calibration Interrupt Enable bit (CAL_IEN) - Default is disabled. This will enable an interrupt from the CAL_DONE status when calibration cycle is complete.
  - — Calibration Time-out Value field (CAL_TIMEOUT[15:0]) - The value in this field determines the number of ring oscillator counts in the calibration sequence. A maximum value equates to about 32.768 sec. assuming the oscillator is near 2 kHz.
- Calibration XTAL Count Register - provides a full 32-bit count value of the number of reference oscillator clocks during the calibration cycle.

The user can use the calibration cycle to tune the ring oscillator frequency. Enable the calibration cycle (at the same time setting the calibration time-out period). After the cycle is complete, read the XTAL count register and compare the result with the expected number of reference oscillator cycles. The oscillator loading can then be adjusted to correct the frequency. Repeat the process as needed.

#### 5.2.3.3.2 32.768 kHz Crystal Oscillator

The 32.768 kHz option oscillator requires use of an external crystal (see Section 3.7, “32.768 kHz Crystal Oscillator (use optional)”. At the expense of the crystal, an accurate RTC timebase and wake-up timer can be provided. The CRM provides a control bit in the System Control Register (SYS_CNTL) and the 32kHz XTAL Control Register (XTAL32_CNTL) for managing the oscillator.

The System Control Register (see Section 5.9.1, “System Control (SYS_CNTL)”) includes the XTAL32_EXISTS control Bit 5.

- This bit is used to switch the internal clock circuitry AFTER the 32 kHz clock is available
- This bit should only be set to one after the 32 kHz oscillator is enabled and verified as running.

The 32kHz XTAL Control Register (see Section 5.9.17, “32kHz XTAL Control (XTAL32_CNTL)”) provides:

- Buffer Gain Control (XTAL32_GAIN[1:0] field) - This 2-bit field allows changing the buffer gain. Default is maximum gain and it is recommended hat this field be left default.
- 32kHz Crystal Enable (XTAL32_EN bit) - Setting this bit to one enables the 32 kHz oscillator.

#### 5.2.3.3.3 Switching from Default Ring Oscillator to 32 kHz Oscillator

Switching to the 32.768 kHz oscillator should consider the following:

Ring Oscillator Control Register (see Section 5.9.15, "Ring Oscillator Control (RINGOSC_CNTL)") provides:

- The ring osc should be disabled first by clearing the Ring Oscillator Enable bit in the Ring Oscillator Control Register.
- The XTAL32_EN bit is then set in the 32kHz XTAL Control Register
- The 32 kHz may require up to 5 sec. to start. The CPU must wait to verify clock startup
- It is suggested to verify 32 kHz startup
  - Enable a timer for a 1 second time-out and interrupt
  - Wait for timer interrupt
  - Read RTC_COUNT value
  - Loop and repeat; exit loop after RTC value has been observed to change validating that the 32 kHz oscillator is clocking the RTC. (Engaging bus_steal can save power during the verification loop time)
- Set the XTAL32_EXISTS bit in the SYSTEM_CNTL register

**NOTE**

Only a hard or soft reset can turn the XTAL32 off.

#### 5.2.3.3.4 Reference Oscillator Divided-by 128

The reference oscillator is kept alive in Doze low power mode. This has the disadvantage of higher current draw than Hibernate, but has the advantage that it allows for an accurate wake-up delay.

- During Doze the prescaler oscillator frequency (typically 187.5 kHz) drives the wake-up timer. As a result, the wake-up timer has a very accurate clock source for use in establishing the sleep period.
- The RTC continues to be clocked by the ring oscillator (or 32 kHz if enabled) during Doze
- The divide-by 128 clock cannot be used for an accurate RTC as the RTC can only be clocked form the ring oscillator or the 32 kHz oscillator.

**NOTE**

Typically, use of Doze mode has no advantage if the 32 kHz oscillator is available. The 32 kHz oscillator provides an accurate timebase for wake-up from Hibernate and does not require the higher current draw of Doze mode due to the reference oscillator running.

### 5.2.3.4 Wake-up Operation

Exiting reset, Hibernate or Doze modes, MC1322x will jump to Idle mode where the MCU will be fully functional and operates from the reference oscillator (the frequency is determined by the programmable prescaler; default is divide-by-1 or full speed). Communication with all mapped registers is possible including the digital modem and RF analog control. The transceiver is completely off. The Sleep module of the CRM gracefully controls this transition.

**NOTE**

- If wake-up is from Idle or Doze modes AND the MCU_RET bit was set before entering the low power mode, then upon wake-up the CPU will start executing from the last program counter location (all processor registers including the program counter are retained during sleep). For consistent operation after wake-up, the needed portions of RAM must also be retained. The RAM retention bits must be programmed accordingly before entering low power.
- The CRM STATUS Register must always be serviced after wake-up as part of the wake-up recovery routine.

The Hibernate mode is initiated by setting the HIB bit in the Sleep Control Register. Software then must wait for the sleep time sync (SLEEP_SYNC) bit, do maintenance of timers like in MACA, and clear this status bit. Clearing this bit will power down the system almost immediately. Waiting for the SLEEP_SYNC can take up to 1 full cycle of the Hibernate clock. When awakened, read the CRM Status register to see if it was a wake-up from Hibernate.

The Doze mode is initiated by setting the DOZE bit in the Sleep Control Register. Software then must wait for the sleep time sync (SLEEP_SYNC) bit, do maintenance of timers like in MACA, and clear this status bit. Clearing this bit will power down the system almost immediately. Waiting for the SLEEP_SYNC can take up to 1 full cycle of the Doze clock. When awakened, read the CRM Status register to see if it was a wake-up from Doze.

#### 5.2.3.5 Power Mode State Machine

The Power Mode State Machine properly guides MC1322x into and out of the sleep modes of Hibernation or Doze. This circuit controls the reference oscillator and power regulators.

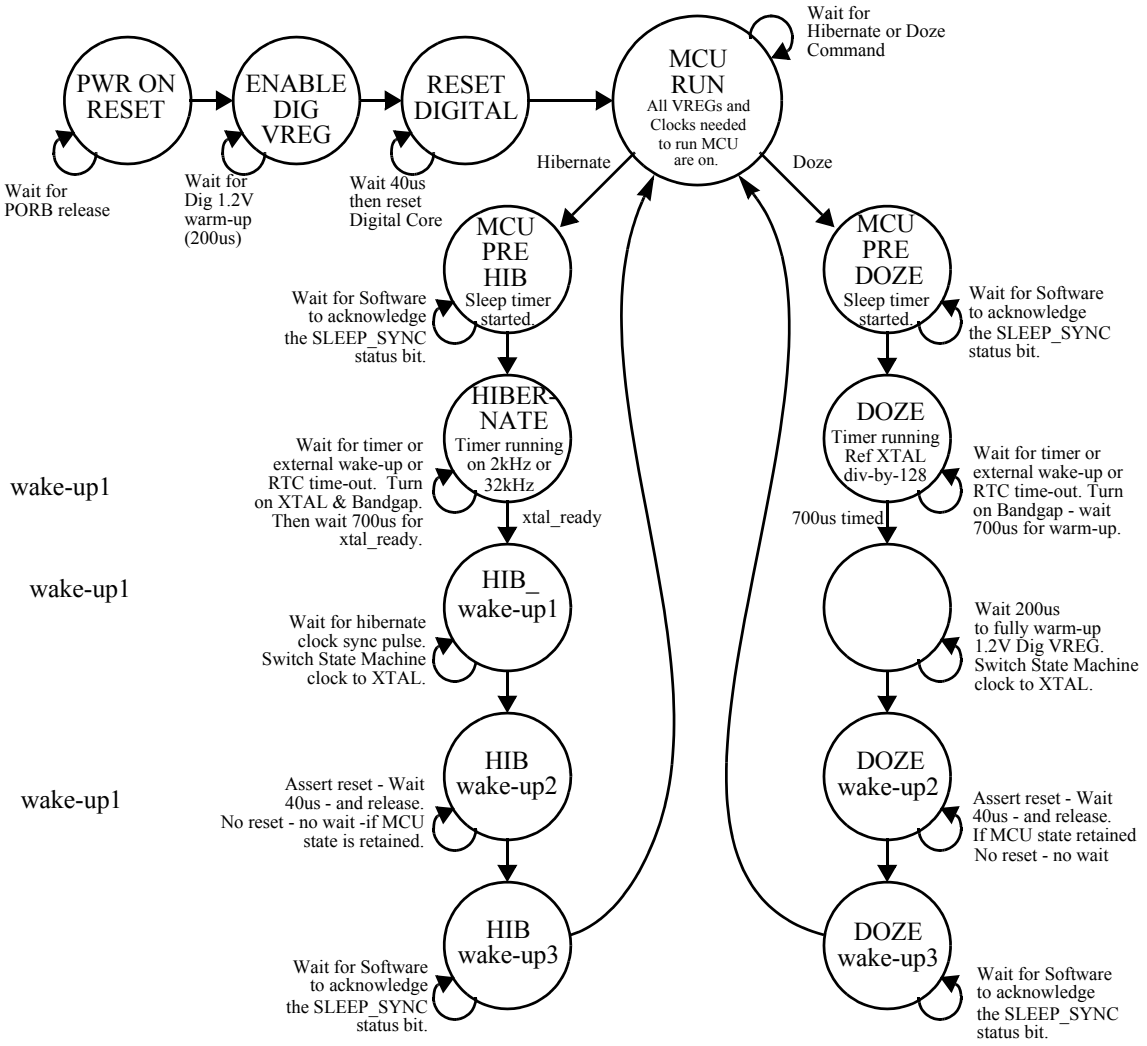



**Figure 5-6. Power Mode State Diagram**

Figure 5-4 shows the sequence for entering MCU RUN state from a “cold start” reset. There are two other main “loops” in the Power Mode State Machine, i.e., the Hibernate loop and the Doze Loop. These are entered from the MCU RUN state and are initiated via a hibernate or Doze command from the Sleep Control (SLEEP_CNTL). On assertion of the HIB or DOZE bits, a sleep mode sequence will be initiated. If HIB, then proceed to MCU_PRE_HIB; else if DOZE, then go to MCU_PRE_DOZE.

**NOTE**

The SLEEP_SYNC (Sleep Time Synchronizer) status bit is described in the following loop descriptions. This status bit coordinates system timing and clocks during entering/exiting low power modes. Its use is detailed in Section 5.9.7, “Status (STATUS)” register, Bit 0. Software should use this bit for sleep and wake-up procedures.

## Hibernate Loop

- **MCU_PRE_HIB** - This state is the transition state between "MCU RUN/Idle" mode and "Hibernate" mode. Upon entering this state, all clocks and power are fully on. MCU_PRE_HIB state stages the turn-off of clocks and power. Upon exit, all non-doze power supplies and clocks will be off.
  - Start the wake-up timer (from zero) and set the SLEEP_SYNC bit.
  - Wait for the SLEEP_SYNC bit to be cleared by software.
  - Freeze all MCU clocks and power down the system, except for the sleep timers and logic
- **HIBERNATE** - This state is the actual hibernate state. On entering this state, the Sleep Module is running solely on the Hibernate Clock.
  - Wait for wake-up - wake-up timer time-out, an external wake-up signal, or an RTC time-out.
  - On wake-up event, start wake-up - start main voltage reference, reference oscillator regulator and reference oscillator.
  - Wait for 700 µsec. for reference oscillator ready.
- **HIB_wake-up1** - this is the first state for wake-up sequence after the reference oscillator is ready.
  - Warm-up and turn-on the digital 1.2V regulator
  - Synchronize and switch the state machine clock to the reference oscillator
  - The MCU and all the RAMs will be connected to the digital 1.2V regulator, but no clocks will be running to MCU.
  - Turn-on the pad ring (apply VBATT to GPIO buffers)
- **HIB_wake-up2** - this state provided to allow MCU reset if the MCU retention has NOT been enabled.
  - If MCU_RET bit is not set - apply a 40usec reset.
  - If the MCU_RET bit is set - no digital reset
  - Enable MCU clock.
- **HIB_wake-up3** - wait state before entering RUN condition
  - Set the SLEEP_SYNC status bit -which indicates the precise clock increment of the sleep timer
  - Wait for the SLEEP_SYNC bit to be cleared by software - on clearing the SLEEP_SYNC bit, the state machine turns on the MACA clock and returns to the MCU_RUN state.

## Doze Loop

- **MCU_PRE_DOZE** - This state is the transition state between "MCU RUN/Idle" mode and "Doze" mode. Upon entering this state, all clocks and power are fully on. MCU_PRE_DOZE state stages the turn-off of clocks and power. Upon exit, all non-doze power supplies and clocks will be off with the exception of the reference oscillator
  - Start the wake-up timer and set the SLEEP_SYNC bit.
  - Wait for the SLEEP_SYNC bit to be cleared by software.
  - Freeze all MCU clocks and power down the system, except for the sleep timers (including reference oscillator) and logic

- **DOZE** - This state is the actual doze state. On entering this state, the Sleep Module is running solely on the Doze Clock (reference oscillator ÷ 128)
  - Wait for wake-up - wake-up timer time-out, an external wake-up signal, or an RTC time-out.
  - On wake-up event, start wake-up - start main voltage reference
  - Wait for 700 µsec.
- **DOZE_wake-up1** - this is the first state for wake-up sequence after the reference oscillator is ready.
  - Warm-up and turn-on the digital 1.2V regulator
  - The MCU and all the RAMs will be connected to the digital 1.2V regulator, but no clocks will be running to MCU.
  - Turn-on the pad ring (apply VBATT to GPIO buffers)
- **DOZE_wake-up2** - this state provided to allow MCU reset if the MCU retention has NOT been enabled.
  - If MCU_RET bit is not set - apply a 40usec reset.
  - If the MCU_RET bit is set - no digital reset
  - Enable MCU clock.
- **DOZE_wake-up3** - wait state before entering RUN condition
  - Set the SLEEP_SYNC status bit - which indicates the precise clock increment of the sleep timer
  - Wait for the SLEEP_SYNC bit to be cleared by software - on clearing the SLEEP_SYNC bit, the state machine turns on the MACA clock and returns to the MCU_RUN state.

### 5.2.3.6 Wake-up Timer

The wake-up timer is a 32-bit counter that is only used in Hibernate or Doze low power modes. The wake-up timer is clocked by the clock source available during the programmed sleep mode (see Figure 5-5):

- Hibernate enables the ring oscillator or optionally the 32.768 crystal oscillator source
  - — With the 2 kHz ring oscillator, maximum sleep time is ~24.85 days
  - — With the 32.768 kHz oscillator, maximum sleep time is ~36.4 hours
- Doze enables the reference osc ÷ 128 (typically 187.5 kHz) - maximum sleep time is ~6.36 hours

The CRM provides the following interfaces to the wake-up counter:

- TIMER_WU_EN Bit (wake-up Control Register, Bit 0) - the wake-up timer enable bit activates wake-up timer option from Doze or Hibernate.
- TIMER_WU_IEN Bit (wake-up Control Register, Bit 16) - the wake-up timer interrupt enable bit activates the interrupt request from a wake-up timer event.
- HIB_WU_EVT Bit (Status Register, Bit 1) and DOZE_WU_EVT Bit (Status Register, Bit 2) - if either the wake-up timer or the RTC cause a wake-up event, the appropriate status bit will be set.
- Wake-up Timeout Register (WU_TIMEOUT) Register - sets the value for the timeout counter that triggers the time-out event.
- Wake-up Count Register (WU_COUNT) - reads the full 32-bit present value of the wake-up counter.

To use the wake-up timer to cause a wake-up event to exit low power mode:

1. Write wake-up Timeout Register with the desired value - the value must be based on the desired delay and the activated low power clock source.
2. Set the TIMER_WU_EN Bit to enable the wake-up timer.
3. Set the TIMER_WU_IEN Bit if an interrupt request is desired.
4. Enable the low power mode (after any other options are set) by setting either the HIB control bit (Bit 0) or the DOZE control bit (Bit 1) in the Sleep Control Register.

After wake-up from sleep, the HIB_WU_EVT status will be set if caused by the wake-up timer.

### 5.2.3.7 Real Time Counter (RTC)

The RTC is a 32-bit counter that can be used to maintain a continuous timebase (with time “tick”) and also provide a periodic wake-up from sleep. The RTC is continuously clocked by the selected low frequency clock source (see Figure 5-5), i.e., the ring oscillator or the 32.768 crystal oscillator (maximum time for the 32.768 oscillator is ~36.4 hours).

The CRM provides the following interfaces to the RTC:

- RTC_WU_EN Bit (wake-up Control Register, Bit 1) - the RTC timer enable bit activates the RTC time-out option. This can be used simply as a periodic interrupt (for a time “tick”) or as wake-up from Doze or Hibernate.

- RTC_WU_IEN Bit (wake-up Control Register, Bit 16) - the RTC time-out interrupt enable bit activates the interrupt request from a RTC timer event.
- RTC_WU_EVT Bit (Status Register, Bit 3) - if the RTC has a time-out event, this bit will be set.
- HIB_WU_EVT Bit (Status Register, Bit 1) and DOZE_WU_EVT Bit (Status Register, Bit 2) - if either the wake-up timer or the RTC cause a wake-up event, the appropriate status bit will be set.
- RTC Periodic Timeout Register (RTC_TIMEOUT) Register - sets the period between time-out events. As soon as the timeout period occurs, the next timeout point will be calculated from the present RTC count and saved.
- RTC Count Register (RTC_COUNT) - reads the full 32-bit present value of the wake-up counter.

#### 5.2.3.8 External Wake-Up Event

A transition on any of the KBI_7 : KBI_4 signals can be enabled to asynchronously wake-up the device. Any or all 4 inputs can be enabled. The input transition can be programmed as to be level or edge sensitive to enable wake-up and generate an interrupt request to the CPU if desired.

Use of the KBI signals is detailed in Section 5.1.5.2, “KBI_7 : KBI_4 Signals as External Interrupt Request Inputs” and Section 5.9.2, “Wake-up Control (WU_CNTL)”.

## 5.3 Managing Low Power Mode

The MC1322x has a number of options for low power operation (see Section 5.2.3.2, “Sleep Operation”). In addition to the two primary hibernate and doze modes, there are options for RAM retention, MCU state retention, GPIO pad state retention, auto ADC operation, and wake-up source. The options used and the interface to the CRM impact management of entering and recovery from low power operation.

### 5.3.1 Entering Low Power Mode

Management of entry into low power mode should consider:

- The user must consider impact of low power options (see Section 3.10, “Low Power Considerations”) on recovery from low power. Especially important is -
  - — Amount of RAM retained - when returning from low power mode, the recovery time will be greatly impacted by how much RAM content needs reloaded.
  - — MCU retention - when returning from low power with MCU retention enabled, the CPU will start from the condition or state at which it stopped and will continue. If retention is disabled (advantageous for lower power), the CPU will start executing from the bottom of RAM as in a start-up condition. Software must be able to recover properly, and good practice may be to save an image of system status (in retained RAM) before entering sleep.
- Options must be set prior to entry into the low power state (via setting either the HIB or DOZE bit in the Sleep Control Register
- The Sleep Module state machine sets the SLEEP_SYNC bit (see Section 5.2.3.5, “Power Mode State Machine”) as the first step to enter low power mode. This function is to allow optional management of the MACA timer. SOFTWARE MUST CLEAR THE SLEEP_SYNC BIT TO FULLY ENTER LOW POWER MODE.

### 5.3.2 Recovery from Low Power Mode

Recovery from low power mode should consider:

- Similar to entering sleep, the Sleep Module state machine sets the SLEEP_SYNC bit as part of exiting sleep. This function is to allow optional management of the MACA timer. SOFTWARE MUST CLEAR THE SLEEP_SYNC BIT TO FULLY EXIT LOW POWER MODE.
- All enabled interrupts must be serviced as part of the recovery process.
- The options selected, the RAM retained, and the system state when entering sleep mode will impact how the recovery process is managed and the length of the recovery time.
- The Auto ADC mode requires special handling for recovery.

### 5.3.3 Auto ADC Mode

The MC1322x supports a version of Hibernate that allows the CRM Sleep Module to wake-up the system periodically and allow the ADC module to take samples independent of CPU activity. Conditions include:

- This mode is enabled by the AUTO_ADC control (Wake-up Control Register, Bit 3)
- Can be used in hibernate mode only (ring oscillator or 32 kHz oscillator is clock source).
- MCU state retention must be enabled (MCU_RET bit set). DIG_PAD_EN need not be set.

The suggested implementation flow is shown in Figure 5-7.

- The AUTO_ADC and MCU_RET must be set before entering Hibernate.
- The RTC Timer is programmed for a periodic wake-up period that initiates the ADC activity.
- The ADC must be programmed with
  - Desired sample mode
  - Threshold(s) to which an input level will be compared
  - Interrupt enabled if the threshold is tripped.
- The wake-up timer is programmed with a value large enough to cover a desired number of periodic wakeups. In this manner, the wake-up timer serves as a watchdog timer in case the ADC activity never trips a programmed threshold.

In observing Figure 5-7, after all proper initialization the hardware flow proceeds as follows:

1. **Start Hibernate** - Hibernate is entered.
2. **Sleep Time-out Event?** - The device will sleep until awakened
   - Wake-up timer - Go to #7 Normal MCU Wake-up (the wake-up timer is set for a much longer delay).
   - RTC periodic timeout - Go to #3 Wake-up MCU with CPU halted
3. **Wake-up MCU with CPU Halted** - The MCU_RET has kept the MCU powered. The CRM Sleep Module starts the reference oscillator. The bus steal module is used to hold the CPU in-active.
4. **Start ADC Module** - The Sleep module starts the ADC module which does one programmed sequence.
5. **ADC Interrupt? -** Threshold exceeded; interrupt set?

— Yes - Go to #8 Unhalt CPU

— No - Got to # 6 Re-enter Hibernate

6. **Re-enter Hibernate** - ADC signals Sleep Module that it has completed. Re-enter hibernate (stop reference osc); go to #2 Sleep Timeout.
7. **Normal MCU Wake-up** - Sleep module exits hibernate in a normal manner with wake-up timer status set**.**
8. **Unhalt CPU** - The ADC has asserted an interrupt request, the CPU is unhalted (bus steal module releases it), and the CPU will run.

**NOTE**

The CPU MUST clear the _AUTO_ADC, respond to the ADC interrupt, and then put the system back to sleep.

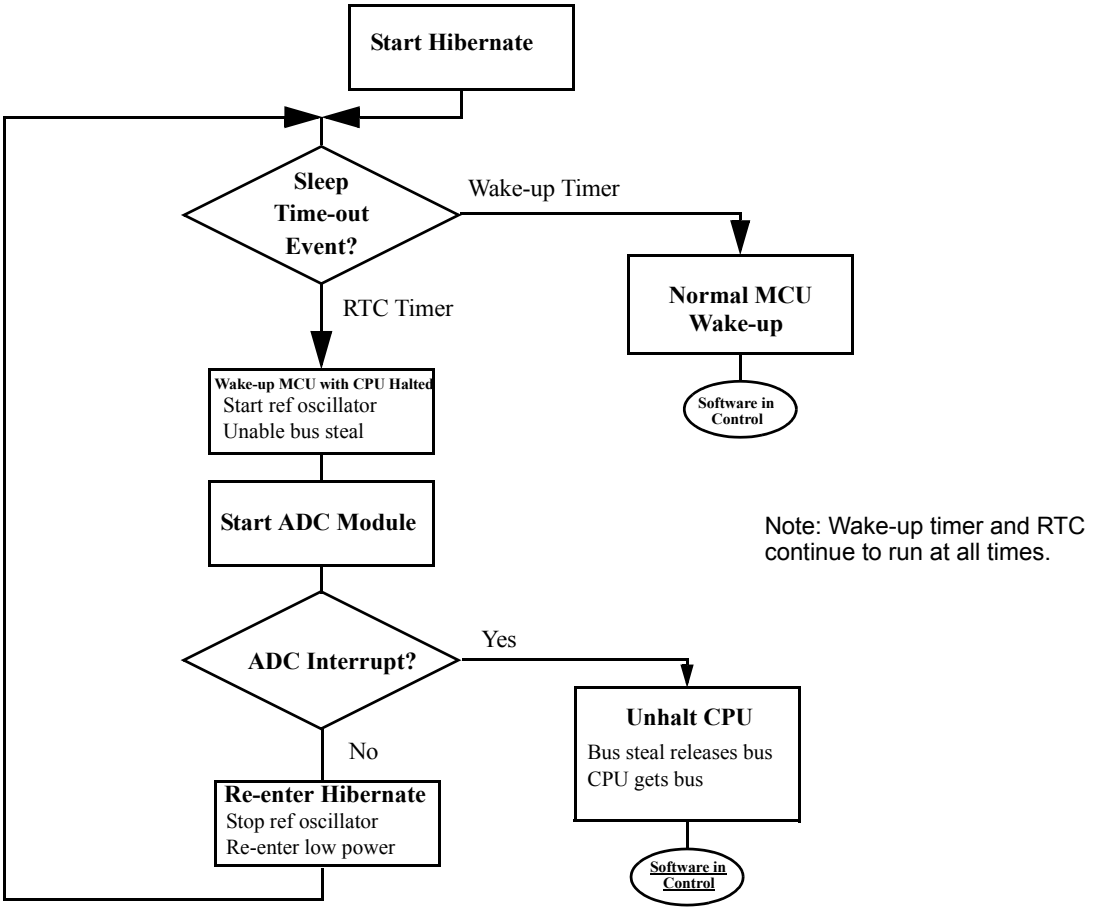



**Figure 5-7. Auto ADC Flow Diagram**

## 5.4 Managing Reference Oscillator

Use of the reference oscillator is detailed in Section 3.6, “Reference Oscillator”. The CRM provides control of the reference oscillator via the Reference XTAL Control Register (see Section 5.9.16, “Reference XTAL Control (XTAL_CNTL)”). The reference oscillator uses a crystal of frequency from 13 - 26 MHz, although 24 MHz is typical and normally recommended.

To minimize system cost, all required crystal capacitive loading is provided onboard the MC1322x. The user, however, must trim the capacitance via the control register to meet the required frequency tolerance of ±40 ppm over temperature.

**NOTE**

It is recommended that the software trim the crystal oscillator frequency to ±10 ppm at room temperature to ensure the larger tolerance over temperature. Supply voltage variation to the device has little affect as the oscillator has its own onboard voltage regulator.

When exiting a POR reset condition, the default loading to the crystal is minimal (XTAL_CTUNE[4:0] = 0x00 and XTAL_FTUNE[4:0] = 0x00). Typically the crystal will require added capacitance to have the frequency within spec; the default frequency will be too high. Typical practice has found that once a hardware implementation has been characterized (with a given crystal), the settings for XTAL_CTUNE[4:0] = 0x00 and XTAL_FTUNE[4:0] can be pre-determined and used as initialization parameters.

**NOTE**

It is considered best practice that frequency accuracy be measured as part of a product’s final test procedure during manufacture.

When wakening from hibernate, the CRM has retained the programmed loading and starts the oscillator with the programmed loading. It need not be restored as part of the wake-up process.

## 5.5 Bus Steal Function

The bus steal function (see Section 5.9.4, “Bus Stealing Control (BS_CNTL)”) is primarily intended to lower average power (current) while allowing the CPU to run. The CPU bus has three possible masters, i.e., the CPU, the DMA for TX/RX, and the bus steal function. If enabled, the bus steal will “steal” clock cycles from the CPU causing the CPU to stay in a halted state to lower average device current. The bus steal function can be used in several modes:

- During a modem function only - if the BS_EN control bit is set, the bus steal will only steal cycles from the CPU during an active modem function. During this period, the DMA is functional and always gets the bus when needed as it has highest priority. The bus steal is inactive all other times. This mode helps limit current draw during peak radio/modem activity.
- Full time (manual enable) - if the BS_MAN_EN control bit is set, the bus steal will be functional continually until disabled. This allows a lower “idle” where little CPU activity is needed. If the DMA is active, it still has highest priority.

- Wait for Interrupt Request - if the BS_EN is active and the WAIT4IRQ control bit is active, the bus steal function is active until an interrupt request goes active; then the bus steal is released so the CPU runs at full bus speed
- During Auto ADC Mode - for AUTO_ADC mode, the bus steal function is used to halt the CPU while the ADC is active on a periodic basis. See Section 5.3.3, “Auto ADC Mode”.

The bus steal does not totally shutdown the CPU, it steal cycles on a duty cycle determined by the ARM_OFF_TIME[5:0] control field. Also, the DMA has the highest priority to control the bus, so that the bus steal will release the bus on request from the DMA when data needs transferred.

Figure 5-8 shows the bus steal state diagram.

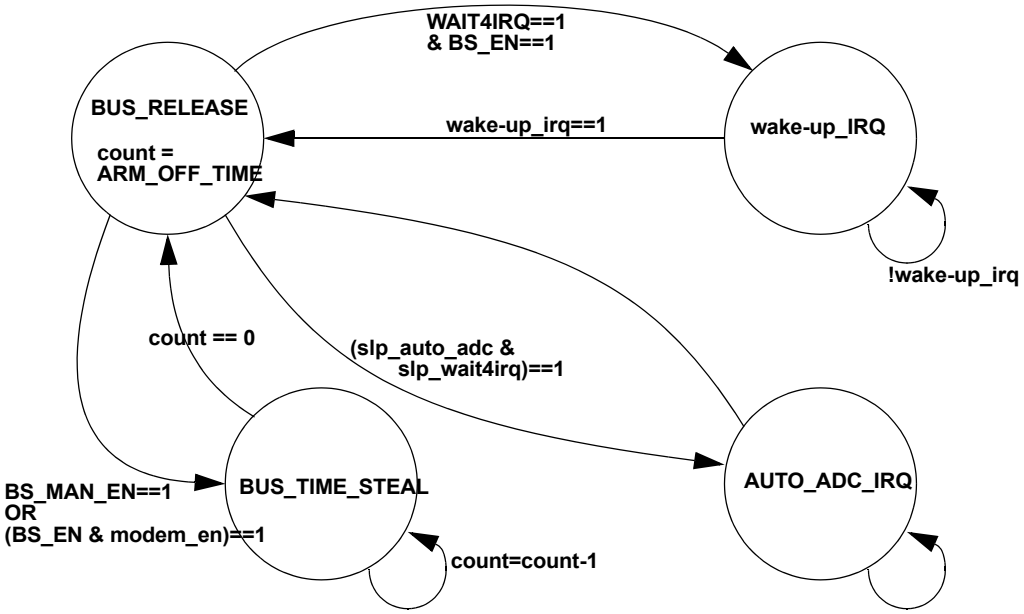



**Figure 5-8. Bus Steal State Diagram**

## 5.6 Computer Operating Properly (COP) or Watchdog Timer Module

The COP timer module can force a system reset or generate an interrupt if an application fails to maintain the timer through expected servicing. To prevent the COP time-out, the software must reset/service the COP timer periodically before the timer times out. If the application program gets lost and fails to reset the COP before it times out, a system reset is generated to force the system back to a known starting point, or alternatively, an interrupt can be generated to the CPU

Important features:

- The watchdog is by default disabled exiting device reset, and must be enabled for use.
- Programmable time out period (between 87ms and 11sec, in 128 steps).
- The watchdog is disabled for ARM debug mode
- The COP time-out can cause a complete reset POR power-up sequence or interrupt
- Programmable response to COP time-out (interrupt or reset, default is interrupt)
  - — Cause a complete reset POR power-up sequence

  - Cause interrupt request
- COP does not run in sleep modes

The CRM provides the COP Control Register (COP_CNTL) and the COP Service Register (COP_SERVICE) for managing the oscillator.

The COP Control Register (see Section 5.9.5, “COP Control (COP_CNTL)”) includes:

- COP_TIMEOUT[6:0] field - sets the time-out value of the COP timer. The period can range from 0.087 to 11.187 seconds with a nominal 24 MHz reference clock
- CPO_COUNT[6:0] field - this is the present value of the COP timer. Every count represents 0.087 seconds (24 MHz clock)
- Three COP control bits -
  - COP_EN - COP enable bit
  - COP_OUT - selects desired COP time-out result, i.e., reset or interrupt request
  - COP_WP - locks present state of COP control. Can only be over-ridden with a system reset.

The COP Service Register (see Section 5.9.6, “COP Service (COP_SERVICE)”) is written to service/reset the COP timer.

- The value 0xc0DE5AFE must be written to reset the counter
- Writing the correct value also clears the interrupt request if that option is enabled (COP_OUT = 1)

Use of the COP reset option allows a higher level of system integrity for a deployed system, in that there is an extra level recovery is possible. Use of the interrupt option can be useful for helping debug software.

Concerning low power COP usage:

- If MCU retention is not used (MCU_RET = 0), the software does need to disable the COP before MC1322x enters sleep mode. This is because all registers will be reset (including the COP_CNTL) when exiting low power mode.
- If MCU retention is used (MCU_RET = 1), software must carefully maintain the COP because the COP counter will be frozen during sleep mode and will start counting again upon wake-up. The software must ensure that the COP does not time out before MC1322x has had a chance to wake up and reset the COP counter.

## 5.7 Interrupts

The CRM Module has a single interrupt request signal CRM_IRQ that is connected to the interrupt controller. The CRM_IRQ can be generated from a number of possible sources which are logically OR'ed such that any of them can generate an interrupt request if enabled. Table 5-3 lists the CRM interrupt sources and their characteristics.

**Table 5-3. CRM Interrupt Sources**

| Item | Status Bit | Mask Bit | Source Description | Interrupt Clear Mechanism |
|---|---|---|---|---|
| 1 | HIB_WU_EVT / DOZE_WU_EVT | TIMER_WU_IEN | Hibernate or Doze wake-up event due to wake-up timer | Writing a 1 to the proper HIB_WU_EVT or DOZE_WU_EVT status bit |
| 2 | RTC_WU_EVT | RTC_WU_IEN | RTC timeout event (wake-up or periodic) | Writing a 1 to the RTC_WU_EVT status bit |
| 3 | EXT_WU_EVT[3:0] | EXT_WU_EN[3:0] | • An external wake-up event has occurred on the associated KBI pin during low power mode<br>• An external transition event has occurred while pin is programmed as GPIO input | Writing a 1 to the associated EXT_WU_EVT status bit |
| 4 | CAL_DONE | CAL_IEN | Calibration period is complete for ring oscillator calibration | Writing a 1 to the CAL_DONE status bit |
| 5 | COP_EVT | COP_OUT | COP timer has timed out | Writing a 1 to the COP_EVT status bit or writing the correct value to the COP_SERVICE register |

### 5.7.1 Wake-up Timer Time-out

The wake-up timer can be used along with the RTC and KBI inputs to exit either Hibernate or Doze mode. If the wake-up timer fires, the appropriate status bit, i.e., either HIB_WU_EVT or DOZE_WU_EVT will be set. An interrupt request will be generated for either if the TIMER_WU_IEN mask bit is set enabling the interrupt. The interrupt request is cleared by writing a one to the appropriate status bit.

### 5.7.2 Real Time Clock Time-out

The RTC timer can be used along with the wake-up timer and KBI inputs to exit either Hibernate or Doze mode. If the RTC timer fires, the RTC_WU_EVT status bit will be set. An interrupt request will be generated if the RTC_WU_IEN mask bit is set enabling the interrupt. The interrupt request is cleared by writing a one to RTC_WU_EVT status bit.

### 5.7.3 External wake-up Signals KBI_7:KBI_4

The GPIO designated as KBI_7, KBI_6, KBI_5, and KBI_4 can individually be programmed to generate an interrupt request on a level change or edge transition. If this occurs during low power mode, it can wake-up the MC1322x, and alternatively, the interrupt requests can be enabled during normal run mode.

If an external signal transition occurs the appropriate bit in the EXT_WU_EVT[3:0] status field will be set. An interrupt request will be generated if the mask bit in the EXT_WU_EN[3:0] field is set enabling the interrupt. The interrupt request is cleared by writing a one to the appropriate bit in the EXT_WU_EVT[3:0] status field.

### 5.7.4 Ring Oscillator Calibration Done

Calibration of the 2 kHz oscillator can take a significant amount of time. The CAL_DONE status signifies the completion of the calibration cycle. This event can initiate an interrupt request if enabled with the CAL_IEN bit. The interrupt request is cleared by writing a one to the CAL_DONE status.

### 5.7.5 COP Time-out

The COP timer can be used either for code debug or recovery from a “run-away’ code situation. When the COP_EVT status bit is set the COP timer has expired, and this status can generate an interrupt request instead of a system reset if enabled by the COP_OUT control bit. The interrupt request is cleared by either serving the COP_SERVICE register or by writing a one to the COP_EVT status.

## 5.8 CRM Register Memory Map

The CRM Module is programmed via a set of memory-mapped registers

- The base address is **0x80003000**.
- The register memory map for the module is listed in Table 5-4
- Table 5-4 also shows which registers are retained in sleep mode

**Table 5-4. CRM Module Register Memory Map**

| Address | Name | Access Type | Retained in Sleep Mode | Width (Bits) | Access Width |
|---|---|---|---|---|---|
| Base + 0x00 | System Control (SYS_CNTL) | R/W | Yes | 32 | All |
| Base + 0x04 | Wake-up Control (WU_CNTL) | R/W | Yes | 32 | All |
| Base + 0x08 | Sleep Control (SLEEP_CNTL) | R/W | Yes | 32 | All |
| Base + 0x0C | Bus Stealing Control (BS_CNTL) | R/W | No | 32 | All |
| Base + 0x10 | COP Control (COP_CNTL) | R/W | No | 32 | All |
| Base + 0x14 | COP Service (COP_SERVICE) | R/W | No | 32 | 32 only |
| Base + 0x18 | Status (STATUS) | R/W | No | 32 | All |
| Base + 0x1C | Module Enable Status (MOD_STATUS) | R | No | 32 | All |
| Base + 0x20 | Wake-up Count (WU_COUNT) | R | Yes | 32 | 32 only |
| Base + 0x24 | Wake-up Timer Compare (WU_TIMEOUT) | R/W | Yes | 32 | 32 only |
| Base + 0x28 | Real Time Count (RTC_COUNT) | R/W | Yes | 32 | 32 only |
| Base + 0x2C | RTC Periodic Wake-up Time-out (RTC_TIMEOUT) | R/W | Yes | 32 | 32 only |
| Base + 0x30 | Reserved | | | | |
| Base + 0x34 | Calibration Timer Set (CAL_CNTL) | R/W | No | 32 | 16/32 only |
| Base + 0x38 | Calibration XTAL Count (CAL_COUNT) | R | No | 32 | All |
| Base + 0x3C | 2kHz Ring Oscillator Control (RINGOSC_CNTL) | R/W | Yes | 32 | 16/32 only |

**Table 5-4. CRM Module Register Memory Map (continued)**

| | | | | | |
|---|---|---|---|---|---|
| Base + 0x40 | Reference XTAL Control (XTAL_CNTL) | R/W | Yes | 32 | 16/32 only |
| Base + 0x44 | 32kHz XTAL Control (XTAL32 CNTL) | R/W | Yes | 32 | All |
| Base + 0x48 | Voltage Regulator Control (VREG_CNTL) | R/W | No | 32 | All |
| Base + 0x4C | Reserved | | | | |
| Base + 0x50 | Software Reset (SW_RST) | R/W | No | 32 | 32 only |
| Base + 0x54 | Reserved | | | | |
| Base + 0x58 | Reserved | | | | |
| Base + 0x5C | Reserved | | | | |

# 5.9 CRM Registers and Control Bits

## 5.9.1 System Control (SYS_CNTL)

The SYS_CNTL register is used to control and configure the system for power supply, clock rate, and other device options. The register contents are retained in sleep mode.

| | SYS_CNTL | | | | | | | | **Base+0x00** | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | | | | | | | | | | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | | | XTAL_CLKDIV[5:0] | | | | | | | | XTAL32_EXISTS | JTAG_SECU_OFF | SPIF_1P8V_SEL | PADS_1P8V_SEL | PWR_SOURCE[1:0] | |
| TYPE | r | r | rw | rw | rw | rw | rw | rw | r | r | r | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |

**Table 5-5. SYS_CNTL Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 14-31 | Reserved | |
| 8-13 | **xtal_clkdiv[5:0]** — Reference oscillator divider (prescaler). This value determines the ratio of the prescaler that divides the reference oscillator frequency to the entire MCU (both CPU core and peripheral source clocks).<br>• It does not affect the rate of the Modem clock.<br>• Divide ratio = xtal_clkdiv[5:0] +1<br>• Maximum divisor is 64<br>• All baud rates and peripheral clocks must be determined from this base frequency | Default = 0x00; divide ratio = 1 (MCU clocks run at reference oscillator rate) |
| 6-7 | Reserved | |
| 5 | **XTAL32_EXISTS —** 32kHz crystal exists. Setting this bit selects the 32kHz oscillator to the Sleep Module mux<br>• This bit will be set to 1 after the 32kHz is operational after enabling the 32kHz XTAL (with XTAL32_EN during IDLE mode)<br>• A time of 1 to 5 second delay can occur before this oscillator is ready.<br>• On system reset, this 32kHz XTAL must be restarted. | 0 = Disable (default)<br>1 = 32 kHz enabled |
| 4 | **JTAG_SECU_OFF —** JTAG Security Disable. This bit gets set by the ROM boot code unless the FLASH image is secured. See Section 3.12.6, "Secure Mode". | 1 = The JTAG and NEXUS modules enabled.<br>0 = The JTAG and NEXUS modules disabled (default). |
| 3 | Reserved | |
| 2 | **PADS_1P8V_SEL** - Output Drive Selector for all Digital Pads. This bit selects whether the digital pads are configured for high drive or standard drive | 0 = Standard drive (default)<br>1 = High drive |
| 0-1 | **PWR_SOURCE[1:0]** — Selection of System Power Source. This value selects the type of supply that will be powering the system. This value will be retained until a hard or soft reset. On POR start-up, a VBATT battery supply is assumed unless over-programmed. On any subsequent wake-up from hibernate or doze, the system will immediately use the over-programmed supply configuration | 0x0 = Default (VBATT)<br>See Table 5-6 |

**Table 5-6. Power Supply Selections**

| PWR_SOURCE | Power Source | Detail |
|---|---|---|
| 2'b00 | VBATT | Any enabling of the Buck regulator will be ignored. (default) |
| 2'b01 | Buck Regulation | Enables buck regulator |
| 2'b10 | External regulated 1.8V | Enabling the Buck or the 1.8V regulator will be ignored. |
| 2'b11 | Reserved | N/A |

**NOTE**

If an external regulated 1.8V supply is selected instead of VBATT battery, then on any subsequent wake-up from hibernate or doze, the system will immediately use the correct voltage supply configuration.

- Using an external 1.8V supply will cause the enabling of the Buck Regulator to be ignored.
- If Bit 1 is set while the BUCK_EN is set, then the external 1.8V supply will not be enabled.

## 5.9.2 Wake-up Control (WU_CNTL)

The WU_CNTL register is used to enable/disable the wake-up events which can come from an external signal or from internal timers. For external wake-ups signals, the polarity and edge/level sense can be programmed for the wake-up signal, and for these or the timer wake-up events, an interrupt request can be generated if enabled.

The control fields EXT_WU_IEN[3:0], EXT_WU_EDGE[3:0], and EXT_WU_EN[3:0] are always active. This allows an interrupt request to be generated based on an input transition from a keyboard or pushbutton transition, even during run mode.

- These fields control the buffer pad logic during low power modes, but the GPIO Module controls the pad logic during run mode.
- During run mode, a pad must be programmed as a GPIO input and a pulldown or pullup enabled (as appropriate) via the GPIO Module to make use of the interrupt request capability.

The register contents are retained in sleep mode.

| | WU_CTRL | | | | | | | | **Base+0x04** | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | EXT_OUT_POL | | | | | | | | EXT_WU_IEN[3:0] | | | | | | RTC_WU_IEN | TIMER_WU_IEN |
| TYPE | r | r | r | r | r | r | r | r | rw | rw | rw | rw | r | r | rw | rw |
| RESET | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | EXT_WU_POL[3:0] | | | | EXT_WU_EDGE[3:0] | | | | EXT_WU_EN[3:0] | | | | AUTO_ADC | HOST_WAKE | RTC_WU_EN | TIMER_WU_EN |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 5-7. WU_CNTL Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 28-31 | **EXT_OUT_POL[3:0]** — External Wake-up Output Polarity (KBI 3, 2, 1, 0, respectively). These bits set the output state of the 4 external output drivers (i.e. KBI[3:0] ) during sleep. The state of the output drivers equal the states of EXT_WU_POL[3:0]. Once awake, the GPIO controller takes control of these pins. | 1 = External driver polarity high (Default)<br>0 = External driver polarity low. |
| 24-27 | Reserved | |
| 20-23 | **EXT_WU_IEN[3:0]** — External Wake-up Interrupt Enable (KBI 7, 6, 5, 4, respectively). These bits enable the 4 external wake-up interrupts (KBI[7:4]) respectively.<br>• If disabled, no any interrupt request is generated<br>• The status bit in CRM STATUS register will be asserted and must be cleared.<br>• The status/interrupt request will be cleared immediately upon servicing | 1 = External wake-up Interrupt request enabled<br>0 = External wake-up Interrupt request disabled (Default) |
| 18-19 | Reserved | |
| 17 | **RTC_WU_IEN** — Real Time Clock Wake-up Interrupt Enable. This bit enables the RTC timer compare interrupt request.<br>• If disabled, no any interrupt request is generated<br>• The status bit in CRM STATUS register will be asserted and must be cleared.<br>• The status/interrupt request will persist for 2 cycles of the low frequency hibernate clock after servicing.<br>**Note:** This may require that this interrupt be disabled in the RTI for these 2 cycles or until the status reads as clear. | 1 = Enable RTC interrupt request<br>0 = Disabled. (Default) |
| 16 | **TIMER_WU_IEN** — Timer wake-up Interrupt Enable. This bit enables the sleep timer compare interrupt for hibernate or doze.<br>• If disabled, no interrupt, request is generated.<br>• The HIB_WU_EVT bit or DOZE_WU_EVT bit in CRM STATUS register will be asserted and must be cleared<br>• To service this interrupt request - a) service the status bit, b) disable the TIMER_WU_IEN until SLEEP_SYNC is cleared (the interrupt request will be active until the SLEEP_SYNC is cleared)., d) HIB_WU_EVT status bit can be cleared simultaneously with SLEEP_SYNC. | 1 = Enable Wake-up timer interrupt request<br>0 = Disabled. (Default) |
| 12-15 | **EXT_WU_POL[3:0]** — External Wake-up Polarity (KBI 7, 6, 5, 4, respectively). These polarity bits control the detection state of the external wake-up inputs (i.e. KBI[7:4] ).<br>• In low power mode, the KBI 7:4 always revert to inputs.<br>• A pullup is enabled if a low level/negative edge is selected (bit = 0)<br>• A pulldown is enabled if a high level/positive edge is selected (bit = 1), which is the default<br>• During run mode these bits still function, but do not control the pad direction or pullup/pulldown.<br><br>**Note:** EXT_WU_POL[x] should only be changed when EXT_WU_EN[x] is disabled otherwise a false wake-up may be activated. | 1 = Detect High level or positive edge. (Default)<br>0 = Detect Low level or negative edge. |

**Table 5-7. WU_CNTL Register Bit Descriptions (continued)**

| Bit Number | Description | Operation |
|---|---|---|
| 8-11 | **EXT_WU_EDGE[3:0]** — External Wake-up Edge or Level Sense (KBI 7, 6, 5, 4, respectively). These bits select whether external wake-up events will be edge or level detected.<br>• If level sense is selected, wake-up will occur whenever EXT_WU_EN[x] is set to 1 and the state of the external wake-up signal matches the EXT_WU_POL[x] bit. To clear the interrupt request, the external pin must be returned to the inactive state and then the status bit, EXT_WU_STAT can be cleared.<br>• If edge sense is selected, then the wake-up will be asserted after an active edge is detected on the external wake-up pin. The status bit can be cleared at any time.<br>**Note:** The pulse width of the signal must be at least 2 clocks of whatever clock is running the edge detector. The clocks can be 2kHz or 32kHz in Hibernate or the high speed XTAL/128 (nominal 24MHz/128 = 187.5kHz) in Doze. | 1 = Edge sensitive.<br>0 = Level sensitive. (Default) |
| 4-7 | **EXT_WU_EN[3:0]** — External wake-up Enables (KBI 7, 6, 5, 4, respectively). These bits control which of the 4 external signals are enabled to wake-up the MC1322x from hibernate or doze.<br>**Note:** If none of the external wake-ups are enabled, a timer source must selected as the wake-up source. | 1 = External wake-up enabled<br>0 = External wake-up Disabled (Default) |
| 3 | **AUTO_ADC** — Enable the Autonomous ADC Mode.<br>• This bit enables the CRM to wake-up only the ADC to take samples and then go back to sleep.<br>• This wake/sleep process will happen independently of the CPU (i.e. the ARM will be in a "halt" state) unless the ADC generates an interrupt request.<br>• If the ADC does issue an interrupt request, the CPU MUST clear AUTO_ADC and then tend to the ADC. The CPU also needs to put the system back to sleep.<br>• This mode can only be used in hibernate with MCU State Retention mode MCU_RET=1(DIG_PAD_EN need not be set).<br>• It is suggested that this mode be used with the RTC Timer with a periodic wakeup.<br>• It is also suggested that the Wake-up Timer be programmed with a value large enough to cover the number of desired periodic wakeups. Here the Wake-up Timer serves as a watchdog timer in case the ADC samples never trip a programmed threshold. | 1 = Enable the Autonomous ADC Mode<br>0 = Disable the Autonomous ADC Mode (default) |
| 2 | **HOST_WAKE** — Enable Master Wake pin. This bit enables MC1322x to be a host wake-up device for a system.<br>• This bit enables pin KBI0/GPIO22 as a wake-up output.<br>• While sleeping, KBI0 will be 0 and upon wake-up, KBI0 will transition to 1.<br>• While set to 1, HOST_WAKE will take priority over any programming on GPIO22.<br>• Clearing HOST_WAKE to zero will allow the GPIO22 to be used as general purpose output again.<br>• The initial enabling of HOST_WAKE will cause the Host-Wake pin to go from 0 to 1 assuming that the GPIO22 had been programmed to 0. | 1 = Enable Host-Wake function/pin<br>0 = Disable Host-Wake Function/pin (default) |

**Table 5-7. WU_CNTL Register Bit Descriptions (continued)**

| Bit Number | Description | Operation |
|---|---|---|
| 1 | **RTC_WU_EN** — Real Time Clock Wake-up Enable. This bit enables the Real Time Clock (RTC) timer compare for exiting low power mode. | 1 = Enabled.<br>0 = Disabled. (Default) |
| 0 | **TIMER_WU_EN** — Timer wake-up Enable. This bit enables the wake-up timer compare for exiting low power mode. | 1 = Enabled.<br>0 = Disabled. (Default) |

## 5.9.3 Sleep Control (SLEEP_CNTL)

The SLEEP_CNTL register is used to control the various sleep options of the system including RAM retention, MCU retention, and GPIO pad retention. as well as, enable sleep mode (hibernate vs. doze). The register contents are retained in sleep mode.

| | SLEEP_CNTL | | | | | | | | Base+0x08 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | | | | | | | | | | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | | | | | | | | | DIG_PAD_EN | MCU_RET | RAM_RET[1:0] | | | | DOZE | HIB |
| TYPE | r | r | r | r | r | r | r | r | rw | rw | rw | rw | r | r | wr0 | wr0 |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 5-8. SLEEP_CNTL Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 8-31 | Reserved | |
| 7 | **DIG_PAD_EN** — Enable Digital (GPIO) Pad Power. This bit enables retention of power to the digital I/O pad ring (all pads associated with GPIOs) in sleep mode.<br>• The DIG_PAD_EN will only be acted on if MCU_RET is also set true.<br>• Typically, the pad ring is turned off for power savings in sleep.<br>**Note:** KBI_0 through KBI_7 are always powered. | 1 = Enable pad power retention<br>0 = Disable pad power retention (default) |
| 6 | **MCU_RET** — MCU State Retention. This bit enables all the states of the MCU, Modem, and Analog Control to be saved during sleep mode. | 1 = Enable MCU state retention<br>0 = Disable MCU state retention (default) |

**Table 5-8. SLEEP_CNTL Register Bit Descriptions (continued)**

| Bit Number | Description | Operation |
|---|---|---|
| 4-5 | **RAM_RET[1:0]** — RAM page Retention. These bits select the amount of RAM that is retained during sleep mode. RAM is selected by Page 0 (8kbytes), Page 1 (24kbytes), Page 2 (32kbytes) and Page 3 (32kbytes) | 2'b00: 8k bytes (Default - Page 0 only retained)<br>2'b01: 32k bytes total (Pages 0 & 1)<br>2'b10: 64k bytes total (Pages 0, 1, & 2)<br>2'b11: 96k bytes total (All pages) |
| 2-3 | Reserved | |
| 1 | **DOZE** — Put system into Doze Mode. Writing a 1 to this bit will start the power down sequence for the entire chip except for the doze timer which will be running at the REF XTAL frequency divided by 128.<br>• A read of this bit always returns zero.<br>• Once this bit is set, hardware synchronizes the Bus clock with the Doze clock - this could take up to 2 cycles of the Doze clock.<br>• During this synchronization period, software must poll the SLEEP_SYNC bit in the STATUS register. Once the SLEEP_SYNC bit is set, software can read and store the MACA timer and do any other Doze tasks that need completed before fully entering doze.<br>• Finally, the software must clear the SLEEP_SYNC bit. This allows final complete power-down of the MCU. | 1 = Enable start of Doze sequence<br>0 = Doze not enabled (default)<br>(Always reads as 0) |
| 0 | **HIB** — Put system into Hibernate Mode. Writing a 1 to this bit will start the power down sequence for the entire chip except for the hibernate timer which will be running at 2kHz from the internal ring oscillator or optionally 32kHz XTAL.<br>• A read of this bit always returns zero.<br>• Once this bit is set, the hardware will synchronize the Bus clock with the Hibernate clock - this could take up to 2 cycles of the Hibernate clock.<br>• During this synchronization period, software must poll the SLEEP_SYNC bit in the STATUS register. Once the SLEEP_SYNC bit is set, software can read and store the MACA timer and do any other Hibernate tasks that need completed before fully entering hibernate.<br>• Finally, the software must clear the SLEEP_SYNC bit. This allows final complete power-down of the MCU. | 1 = Enable start of Hibernate sequence<br>0 = Hibernate not enabled (default)<br>(Always reads as 0) |

## 5.9.4 Bus Stealing Control (BS_CNTL)

The CPU bus has three possible masters, i.e., the CPU, the DMA for TX/RX, and the bus steal function. If enabled, the bus steal module will "steal" clock cycles from the CPU causing the CPU to stay in a halted state to lower average device current. The bus steal function can be used in several modes (see Section 5.5, "Bus Steal Function"):

- During a modem function only - if the BS_EN control bit is set, the bus steal will only steal cycles from the CPU during an active modem function.
- Full time (manual enable) - if the BS_MAN_EN control bit is set, the bus steal will be functional continually until disabled.
- Wait for Interrupt Request - if the BS_EN is active and the WAIT4IRQ control bit is active, the bus steal function is active until an interrupt request goes active; then the bus steal is released so the CPU runs at full bus speed.
- During Auto ADC Mode - for AUTO_ADC mode, the bus steal function is used to halt the CPU while the ADC is active on a periodic basis. See Section 5.3.3, "Auto ADC Mode".

The bus steal does not totally shutdown the CPU, it steal cycles on a duty cycle determined by the ARM_OFF_TIME[5:0] control field. Also, the DMA has the highest priority to control the bus, so that the bus steal will release the bus on request from the DMA when data needs transferred.

This register controls the bus steal function, and the register contents are not retained in sleep mode.

| | BS_CNTL | | | | | | | | **Base+0x0C** | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | | | | | | | | | | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | | | ARM_OFF_TIME[5:0] | | | | | | | | | | | BS_MAN_EN | WAIT4IRQ | BS_EN |
| TYPE | r | r | rw | rw | rw | rw | rw | rw | r | r | r | r | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 5-9. BS_CNTL Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 14-31 | Reserved | |
| 8-13 | **ARM_OFF_TIME[5:0]** — Bus Steal Cycles. This value determines the number of clock cycles to steal away from the CPU. The maximum number of cycles to steal is 64 (value = 63). The bus is stolen for N+1 cycles (value); then the CPU gets the bus for 2 cycles. The bus is then stolen again for another N+1 cycles. | 0x00 = Default<br>····<br>0x3F = 64 cycles<br><br>See Figure 5-9. |
| 3-4 | Reserved | |
| 2 | **BS_MAN_EN** — Manual Bus Stealing Enable. The bit enables the bus stealing directly (without need for transceiver activity) through software.<br>• This bit has higher priority than BS_EN.<br>• Bus stealing is activated immediately when this bit is set (ARM_OFF_TIME should also be set). | 1 = Immediately enable the bus stealing.<br>0 = Disable the bus stealing (default)<br>**Note:** Disabling can take longer (based on ARM_OFF_TIME value) because the ARM is being stalled from executing code. |
| 1 | **WAIT4IRQ** — Wait for Interrupt Request. Setting this bit to 1 halts the CPU and makes it wait for an interrupt to be released.<br>• This bit control is only available when the BS_EN is true<br>• Read back of this bit will always be zero - bit is self clearing. | 1 = Halt CPU and wait for interrupt.<br>0 = Wait for interrupt in-active (default) |
| 0 | **BS_EN** — Bus Stealer Enable. This bit allows bus steal to automatically start and steal CPU cycles whenever the transceiver is active. The bus steal automatically stops when the transceiver operation is complete.<br>• ARM_OFF_TIME should also be set when enabling BS_EN<br>• This bit also enables a “Wait for Interrupt” (WAIT4IRQ) where the ARM is halted and needs an interrupt to be freed. | 1 = Enable the automatic bus steal<br>0 = Disable bus steal (default)<br>**Note:** Disabling can take longer (based on ARM_OFF_TIME value) because the CPU is being stalled from executing code. |

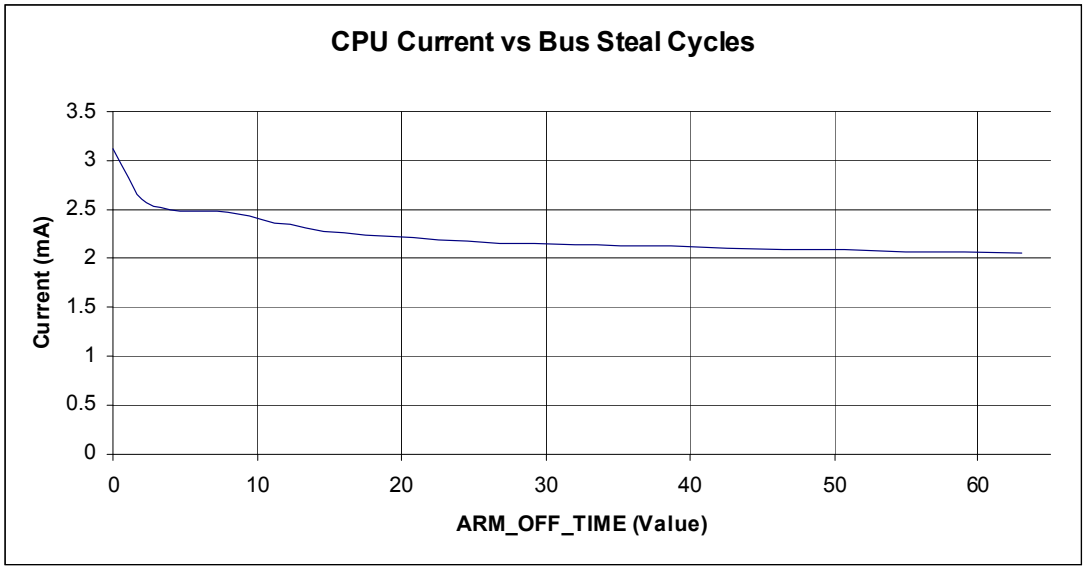



**Figure 5-9. Typical CPU Current vs. ARM_OFF_TIME (CPU clock = 24 MHz)**

## 5.9.5 COP Control (COP_CNTL)

The COP Control register configures the watchdog timer which allows recovery from a runaway code situation. The COP timer is clocked by the reference oscillator and the delay period is programmable from 87 ms to 11 seconds (with 24 MHz reference clock). The register contents are not retained in sleep mode.

| | COP CONTROL | | | | | | | | **Base+0x10** | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | | | | | | | | | | COP_COUNT[6:0] | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | | COP_TIMEOUT[6:0] | | | | | | | | | | | | COP_WP | COP_OUT | COP_EN |
| TYPE | r | rw | rw | rw | rw | rw | rw | rw | r | r | r | r | r | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 5-10. COP_CNTL Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 23-31 | Reserved | |
| 16-22 | **COP_COUNT[6:0]** — COP Counter Value. This read-only value is the running count in the COP timer. Every count of the COP timer is 87ms when running at 24MHz. | 0x00 = Default |
| 15 | Reserved | |
| 8-14 | **COP_TIMEOUT[6:0]** — COP Time-out Period. The value in this register determines the time-out period of the COP counter. The COP counter counts reference oscillator clocks (nominally 24MHz). | See Table 5-11. 0x00 = Default |
| 3-7 | Reserved | |
| 2 | **COP_WP** - COP Write Protect. Setting this bit disables a write to the COP CONTROL register. Once set, the register becomes read only, and this condition can only be cleared via a reset | 1 = Protection set; register is read only<br>0 = Protection not set; register is read/write (default) |
| 1 | **COP_OUT** - COP Output. This bit selects the result of a COP time-out; 1) interrupt request, or 2) system reset | 1 = COP time-out generates an interrupt request<br>0 = COP time-out generates a system reset (default) |
| 0 | **COP_EN** - COP Enable. The bit enables operation of the COP. THIS BIT CAN ONLY BE CHANGED WHEN COP_WP IS SET TO ZERO | 1 = COP enabled<br>0 = COP disabled (default) |

To determine the required COP_TIMEOUT[6:0] field setting for a desired time-out period, see Table 5-11. With a 24 MHZ reference frequency (typical usage) the time-out period can vary from 0.087 to 11.18 seconds.

**Table 5-11. COP Time-Out Values**

| Value (hex) | Timeout Period Formula | Time out Period @ 24MHz (sec) |
|---|---|---|
| 0x00 | $\frac{(\text{Value}+1)\times 2^{(21)}}{\text{RefXTALFreq}}$ | 0.087 |
| 0x01 | | 0.175 |
| 0x02 | | 0.262 |
| 0x03 | | 0.350 |
| | | .. |
| | | .. |
| 0x7F | | 11.18 |

The following additionally describes use of the COP_CNTL register:

- To read the COP time, the COP_COUNT[6:0] field can be read.
- COP_TIMEOUT[6:0] should be written before the COP is enabled.

**NOTE**

Once the COP has been enabled, the recommended procedure for changing the COP_TIMEOUT[6:0] is:

- Disable the COP
- Write to COP_TIMEOUT
- Re-enable the COP.

The COP counter is not reset by a write to COP_TIMEOUT. Changing TIMEOUT while the COP is enabled will result in a time-out period that differs from the expected value.

- COP_CNTL bits cannot be changed after COP_WP is set to 1 until a reset occurs.
- In Hibernate or Doze modes, the COP is disabled along with the entire MCU. However, if the state of the MCU is retained, then the COP counter will also be retained, although frozen during sleep. When the core awakens, the COP will continue counting where it was stopped before entering the sleep mode.

## 5.9.6 COP Service (COP_SERVICE)

The COP Service register must be periodically written (serviced) before the COP timer times out. Use 32-bit access (read and write) only. The register contents are not retained in sleep mode.

- Writing the value 0xC0DE5AFE resets the COP timer.
- If the interrupt output option is selected (COP_OUT == 1) then the correct service write will clear the interrupt.
- Reading the register always returns 0xC0DE5AFE.

| | COP SERVICE | | | | | | | | **Base+0x14** | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | COP_SERVICE[31:16] | | | | | | | | | | | | | | | |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | COP_SERVICE[15:0] | | | | | | | | | | | | | | | |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |

**Table 5-12. COP SERVICE Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 0-31 | **COP_SERVICE[31:0]** - COP Service Register. The register must be written with 0xC0DE5AFE to reset COP timer | Default read = 0xC0DE5AFE |

## 5.9.7 Status (STATUS)

The CRM Status register reports CRM events and status. Reading this register does not alter the values of the status bits, however, writing a 1 to certain status bits clears them (in Table 5-13, these bits are signified by notation “rw1c”). The register contents are not retained in sleep mode.

| | STATUS | | | | | | | | Base+0x18 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | | | | | | | | | | | | | VREG_1P5V_RDY | VREG_1P8V_RDY | VREG_BUCK_RDY | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | | | | | | COP_EVT | CAL_DONE | | EXT_WU_EVT[3:0] | | | | RTC_WU_EVT | DOZE_WU_EVT | HIB_WU_EVT | SLEEP_SYNC |
| TYPE | r | r | r | r | r | r | rw1c | r | rw1c | rw1c | rw1c | rw1c | rw1c | rw1c | rw1c | rw1c |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 5-13. STATUS Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 20-31 | Reserved | |
| 19 | **VREG_1P5V_RDY** — 1.5V Analog Voltage Regulator (for transceiver) Ready Status. This status bit will be 1 when the 1.5V Regulator has been given enough time to warm up.<br>• The warm up time is 600us at 26MHz if RX/TX and PLL have to be turned on or 200us if just the 1.5V Regulator for the RX/TX.<br>• There is no associated CPU interrupt.<br>• The 1.5V Regulator is enabled with the VREG_1P5V_EN bit in the VREG_CNTL register. | 1 = 1.5V Regulator is ready to use<br>0 = 1.5V Regulator is not ready to use (default) |
| 18 | **VREG_1P8V_RDY** — 1.8V NVM Voltage Regulator Ready Status. This status bit will be 1 when the 1.8V Regulator has been given enough time to warm up.<br>• The warm up time is 200us at 26MHz.<br>• There is no associated CPU interrupt.<br>• The 1.8V Regulator is enabled with the VREG_1P8V_EN bit in the VREG_CNTL register. | 1 = 1.8V NVM Regulator is ready to use<br>0 = 1.8V NVM Regulator is not ready to use (default) |

**Table 5-13. STATUS Register Bit Descriptions (continued)**

| Bit Number | Description | Operation |
|---|---|---|
| 17 | **VREG_BUCK_RDY** — Buck Voltage Regulator Ready Status. This status bit will be 1 when the Buck Regulator has been given enough time to warm up.<br>• The warm up time is 400us and 700us at 26MHz for regulation and bypass, respectively.<br>• There is no associated CPU interrupt.<br>• The Buck is enabled with the VREG_BUCK_EN bit in the VREG_CNTL register.<br>• This bit can also be used to determine when the buck has been properly bypassed in order to use the 1.8V NVM Regulator. Switching from Buck regulation to Buck bypass, while the NVM Regulator is on, is not allowed. If this condition occurs, then VREG_BUCK_RDY will be 0. The NVM Regulator (VREG_1P8V_EN) must be disabled first and then the Buck can be switched to bypass mode. | 1 = Buck is ready to use<br>0 = Buck is not ready to use (default) |
| 11-16 | Reserved | |
| 10 | **COP_EVT** - COP Event Status. This read-only bit is set to indicate a COP timeout event status<br>• It is usable is when the COP_OUT is enabled.<br>• It is cleared by writing the correct word to the COP_SERVICE register.<br>• If the COP_OUT is not set, a COP event will cause a complete reset of MCU and this status bit will be cleared. | 1 = COP event has occurred<br>0 = No COP event has occurred (default) |
| 9 | **CAL_DONE** — Calibration Done. This bit indicates that a ring oscillator calibration event is complete.<br>• Software should read the CAL_XTAL_CNT register to get the number of Reference oscillator cycles counted.<br>• The CAL_DONE bit is cleared by writing this bit position. | 1 = Calibration done.<br>0 = Calibration not done (default) |
| 8 | Reserved | |
| 4-7 | **EXT_WU_EVT[3:0]** — External wake-up Event Status. These status bits indicate that an event has occurred on any one of 4 external KBI input pins to wake-up the processor.<br>• These bits will only be set if enabled by their counterparts in the CRM WU_CNTL register.<br>• Writing a 1 to these bits will clear the status and clear the associated interrupt request, if also enabled in the WU_CNTL.<br>• Reading will not alter the contents.<br>• These 4 bits are ORed and sent to the ITC module. | 1 = External wake-up Event has occurred.<br>0 = No External wake-up Event (default) |

**Table 5-13. STATUS Register Bit Descriptions (continued)**

| Bit Number | Description | Operation |
|---|---|---|
| 3 | **RTC_WU_EVT** — RTC Timer Time-out Event Status. This status bit reflects an RTC periodic timer event.<br>• The RTC can cause a wake-up from sleep mode or just provide periodic timeouts.<br>• This bit will only be set if enabled by the RTC_WU_EN bit in the CRM WU_CNTL register.<br>• Writing a 1 to this bit will clear the status and clear the associated interrupt request if enabled.<br>• Reading will not alter the contents.<br>• With the assertion of this status, status bit, the value in the RTC_COUNT register can be read.<br>**Note:** This bit requires some special handling because of the asynchronous nature between the bus clock and the RTC clock (2kHz or 32kHz). When software clears the RTC_WU_EVT, the bit will be cleared in 2 cycles of the slower RTC clock. Software cannot go back to sleep until this bit is cleared. If commanded to go to sleep while the RTC_WU_EVT bit is still set, the chip will wake up immediately. | 1 = RTC Timer event has occurred.<br>0 = No RTC Timer event (default) |
| 2 | **DOZE_WU_EVT** — Doze Timer wake-up Event Status. This status bit reflects that the chip has awakened from doze.<br>• Wake-up from doze can be caused by the wake-up timer, the RTC or the external interrupt pads.<br>• This bit will only be set if enabled by the TIMER_WU_EN bit in the CRM WU_CNTL register.<br>• Writing a 1 to this bit will clear the status and clear the associated interrupt request if enabled.<br>• Reading will not alter the contents. | 1 = Timer wake-up Event has occurred.<br>0 = No Timer wake-up Event= |

**Table 5-13. STATUS Register Bit Descriptions (continued)**

| Bit Number | Description | Operation |
|---|---|---|
| 1 | **HIB_WU_EVT** — Hibernate Timer wake-up Event Status. This status bit reflects that the chip has awakened from hibernate.<br>• Wake-up from doze can be caused by the wake-up timer, the RTC or the external interrupt pads.<br>• This bit will only be set if enabled by the TIMER_WU_EN bit in the CRM WU_CNTL register.<br>• Writing a 1 to this bit will clear the status and clear the associated interrupt request if enabled.<br>• Reading will not alter the contents. | 1 = Hibernate wake-up Event has occurred.<br>0 = No Timer wake-up Event (default) |
| 0 | **SLEEP_SYNC** — Sleep Time Synchronizer. This status bit has 2 purposes:<br>• Either entering sleep mode and exiting wake-up.<br>• When entering sleep mode, this bit is enabled the exact point at which the sleep timer has begun timing from 0. At this point, the software can read the MACA timer value which has been frozen by hardware and store it for later recovery. Once the MACA value is stored, the software needs to then clear this bit by writing a 1. Clearing this status bit will immediately freeze all MCU clocks and power down the system, except for the sleep timers and logic. Make sure all sleep mode maintenance is done prior to clearing this bit. Reading will not alter the contents.<br>• Upon wake-up, software will need to immediately start polling this status bit. The bit is asserted at the exact point to which the sleep timer has counted; and the timer value can be fetched from the WU_COUNT register. This can be used to adjust the MACA timers with the precise amount of sleep time. The MACA can be loaded with the missing time and started.<br>• This bit and wake-up procedure should be used for any type of wakeup. | 1 = Sleep timer count has occurred.<br>0 = No counter change. (default) |

## 5.9.8 Module Enable Status (MOD_STATUS)

This read-only Module Enable Status register simply reports which modules are enabled or disabled.

| | MOD_STATUS | | | | | | | | Base+0x1C | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | | | | | | | | | | | | | | | AIM_EN | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | NEX_EN | JTA_EN | | ADC_EN | SPIF_EN | SSI_EN | I2C_EN | RIF_EN | TMR_EN | UART2_EN | UART1_EN | GPIO_EN | SPI_EN | ASM_EN | MACA_EN | ARM_EN |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

**Table 5-14. MOD_STATUS Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 18-31 | Reserved | |
| 17 | **AIM_EN** — Analog Interface Module enable status | 1 = Enabled<br>0 = Not enabled |
| 16 | Reserved | |
| 15 | **NEX_EN** — Nexus Module enable status | 1 = Enabled<br>0 = Not enabled |
| 14 | **JTA_EN** — JTAG Module enable status | 1 = Enabled<br>0 = Not enabled |
| 13 | Reserved | |
| 12 | **ADC_EN** — Analog to Digital Converter Module enable status | 1 = Enabled<br>0 = Not enabled |
| 11 | **SPIF_EN** — SPI FLASH Module enable status | 1 = Enabled<br>0 = Not enabled |
| 10 | **SSI_EN** — Synchronous Serial Interface Module enable status | 1 = Enabled<br>0 = Not enabled |
| 9 | **I2C_EN** — $I^2C$ Module enable status | 1 = Enabled<br>0 = Not enabled |
| 8 | **RIF_EN** — Radio Interface Module enable status | 1 = Enabled<br>0 = Not enabled |

**Table 5-14. MOD_STATUS Register Bit Descriptions (continued)**

| Bit Number | Description | Operation |
|---|---|---|
| 7 | **TMR_EN** — Timer Module enable status - If any clock to any counter is enabled, this bit is set. | 1 = Enabled<br>0 = Not enabled |
| 6 | **UART2_EN** — Universal Asynchronous Receiver/Transmitter 2enable status | 1 = Enabled<br>0 = Not enabled |
| 5 | **UART1_EN** — Universal Asynchronous Receiver/Transmitter 1 enable status | 1 = Enabled<br>0 = Not enabled |
| 4 | **GPIO_EN** — General Purpose I/O Module enable status | 1 = Enabled<br>0 = Not enabled |
| 3 | **SPI_EN** — Serial Peripheral Interface Module enable status | 1 = Enabled<br>0 = Not enabled |
| 2 | **ASM_EN** — Advanced Security module enable status | 1 = Enabled<br>0 = Not enabled |
| 1 | **MACA_EN** — MAC Accelerator enable status | 1 = Enabled<br>0 = Not enabled |
| 0 | **ARM_EN** — CPU enable status | 1 = Enabled<br>0 = Not enabled |

## 5.9.9 Wake-up Count (WU_COUNT)

The WU_COUNT[31:0] register reflects the current count of the wake-up timer. Upon entering hibernate or doze mode, this counter will be reset, start at a count of zero and count up. Use 32-bit accesses only. The register contents are retained in sleep mode.

| | WU_COUNT | | | | | | | | Base+0x20 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Base+0x20** | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | WU_COUNT[31:16] | | | | | | | | | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | WU_COUNT[15:0] | | | | | | | | | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 5-15. WU_COUNT Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 0-31 | **WU_COUNT[31:0]**— Wake-up Timer Count. This is the running value of the CRM sleep timer that runs on the 2kHz, XTAL/128 or 32kHz (selectable by the XTAL32_EN bit in the XTAL32_CNTL register).<br>• Entering sleep mode resets this timer to zero.<br>• This value can used to adjust missing time in the MACA block. | 0x0000_0000 = Default from reset |

There are three clocks that can run the sleep timer: Timer usage is shown in Table 5-16.

- 2kHz ring oscillator
- 32kHz XTAL
- Reference oscillator divided-by 128 (nominally 24MHz/128).

**Table 5-16. Timer Usage**

| | 2kHz Ring Osc (Hibernate Mode) | 32kHz XTAL (Hibernate Mode) | XTAL/128 (Doze Mode) |
|---|---|---|---|
| Max Sleep Time | 24.8 days | 36.4 hours | 11.75 hours @ 13MHz<br>6.36 hours @ 24 MHz<br>5.875 hours @ 26MHz |

## 5.9.10 Wake-up Time-out (WU_TIMEOUT)

The WU_TIMEOUT[31:0] register sets the time-out compare value for the wake-up timer. Use 32-bit accesses only. The register contents are retained in sleep mode.

| | WU_TIMEOUT | | | | | | | | **Base+0x24** | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | WU_TIMEOUT[31:16] | | | | | | | | | | | | | | | |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | WU_TIMEOUT[15:0] | | | | | | | | | | | | | | | |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 5-17. WU_TIMEOUT Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 0-31 | **WU_TIMEOUT[31:0]** — Wake-up Time-out compare value. This value sets the sleep duration time for the wake-up timer for either hibernate or doze.<br>• The true sleep duration will be longer than the value set because several sleep clock cycles are necessary to wake-up the MCU.<br>• Upon wake-up, the software should pole the appropriate status bit in the STATUS register and when the status bit goes true, it will then need to read the WU_COUNT register to find out exactly how long it has actually been asleep for. | |

## 5.9.11 RTC Count (RTC_COUNT)

The RTC_COUNT[31:0] register reflects the current count of the free-running real time clock timer. The 2kHz ring oscillator, or optionally, the 32kHz crystal oscillator must be enabled for the counter to be running. This is an up-counter, it starts at 0 and will rollover at 0xFFFF_FFFF. It cannot be reset by software. No special consideration has been given to the switch from 2kHz to 32kHz and visa-versa. 32-bit accesses only should be used for accuracy. The register contents are retained in sleep mode.

| | RTC_COUNT | | | | | | | | **Base+0x28** | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | RTC_COUNT[31:16] | | | | | | | | | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | RTC_COUNT[15:0] | | | | | | | | | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 5-18. RTC_COUNT Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 0-31 | **RTC_COUNT[31:0]** — Real Time Clock (RTC) Count. This is the running value of the RTC timer. | |

## 5.9.12 RTC Time-out (RTC_TIMEOUT)

This register sets the incremental time-out compare value for the RTC timer. Use 32-bit accesses only. The register contents are retained in sleep mode.

| | RTC_TIMEOUT | | | | | | | | Base+0x2C | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | RTC_TIMEOUT[31:16] | | | | | | | | | | | | | | | |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | RTC_TIMEOUT[15:0] | | | | | | | | | | | | | | | |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 5-19. RTC_TIMEOUT Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 0-31 | **RTC_TIMEOUT[31:0]** — RTC Periodic Time-out. This value sets the incremental time at which an RTC interrupt will be periodically generated.<br>• An interrupt request based on an RTC time-out can be used while awake or can cause a wake-up event.<br>• As soon as the time-out period occurs, the next time-out point (based on current RTC tcount) is calculated in hardware and saved.<br>• A new time-out value can be written at any time, BUT the new time-out value will be calculated and effective only after the next RTC clock. | |

## 5.9.13 Calibration Control Register (CAL_CNTL)

The Calibration Control register allows the 2kHz Ring Oscillator to be calibrated against the Reference XTAL. Use 16-bit or 32-bit accesses only. The register contents are not retained in sleep mode.

| | CAL_CNTL | | | | | | | | **Base+0x34** | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | | | | | | | | | | | | | | | CAL_IEN | CAL_EN |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | CAL_TIMEOUT[15:0] | | | | | | | | | | | | | | | |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 5-20. CAL_CNTL Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 18-31 | Reserved | |
| 17 | **CAL_IEN** - Calibration Interrupt Enable. This bit enables an interrupt request signifying that the calibration is complete. If disabled, there will be no interrupt request, however, the event status bit (CAL_DONE) in STATUS will be asserted and will need to be cleared. | 1 = Interrupt request enabled<br>0 = Disabled. (Default) |
| 16 | **CAL_EN** — Calibration Enable. This bit enables the calibration sequence. This bit will be automatically cleared and the CAL_DONE bit in CRM STATUS will be set on completion of the calibration. Writing a zero to CAL_EN during the calibration will abort the process. | 1 = Calibration enabled.<br>0 = Calibration disabled (Default) or cause calibration to abort. |
| 0-15 | **CAL_TIMEOUT[15:0]** — Calibration Time-out Value. This value determines how long the calibration is to last. The maximum amount of calibration time allowed (i.e. using all 16 bits) is 32.768sec at 2kHz. This value can be set at the same time as the CAL_EN. | |

## 5.9.14 Calibration XTAL Count (CAL_COUNT)

This register contains the reference oscillator count value after a ring oscillator calibration sequence.

| | CAL_COUNT | | | | | | | | **Base+0x38** | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | CAL_COUNT[31:16] | | | | | | | | | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | CAL_COUNT[15:0] | | | | | | | | | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 5-21. CAL_COUNT Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 0-31 | **CAL_COUNT[15:0] and CAL_COUNT[31:16]** — Calibration XTAL Count. This value contains the number of reference oscillator cycles that occurred during the calibration period. This value is valid after the CAL_DONE status or it's associated interrupt is asserted. | |

## 5.9.15 Ring Oscillator Control (RINGOSC_CNTL)

This register controls the calibration sequence for the ring oscillator. Byte writes are not allowed for this register; use 16-bit or 32-bit access only. The register contents are retained in sleep mode.

| | RINGOSC_CNTL | | | | | | | | Base+0x3C | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | | | | | | | | | | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | | | | ROSC_CTUNE[3:0] | | | | | ROSC_FTUNE[4:0] | | | | | | | ROSC_EN |
| TYPE | r | r | r | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |

**Table 5-22. RINGOSC_CNTL Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 13-31 | Reserved | |
| 9-12 | **ROSC_CTUNE[3:0]** — Ring Oscillator Course Tune. This is the course tune adjustment for the ring oscillator. Each step changes loading by 1 pF | 0x6 = Default |
| 4-8 | **ROSC_FTUNE[4:0]** — Ring Oscillator Fine Tune. This is the fine tune adjustment for the ring oscillator. Each step changes loading by 160 fF | 0x17 = Default |
| 1-3 | **Reserved** | |
| 0 | **ROSC_EN** — Ring Oscillator Enable. The ring oscillator is enabled by default. This bit can disable the ring oscillator, but this should only be done when the 32kHz clock source is present (necessary for waking from hibernate mode). | 1 = Ring Oscillator ON (default)<br>0 = Ring Oscillator OFF |

## 5.9.16 Reference XTAL Control (XTAL_CNTL)

This register controls the loading to the reference oscillator. Byte writes are not allowed; use 16-bit or 32-bit access only. The register contents are retained in sleep mode.

| | XTAL_CNTL | | | | | | | | Base+0x40 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | | | | | | | XTAL_CTUNE[4:0] | | | | | XTAL_FTUNE[4:0] | | | | |
| TYPE | r | r | r | r | r | r | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | | | | | XTAL_IBIAS_SEL[3:0] | | | | | | | | | | | |
| TYPE | r | r | r | r | rw | rw | rw | rw | r | rw | rw | rw | r | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |

**Table 5-23. XTAL_CNTL Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 26-31 | Reserved | |
| 21-25 | **XTAL_CTUNE[4:0]** — Reference oscillator coarse tuning for load capacitance:<br>XTAL_CTUNE[4] - selects 4 pF load to crystal<br>XTAL_CTUNE[3:0] - values 0-7 select load capacitance equal to programmed number, i.e., value 7 selects 7 pF | 1) Default = 0x00<br>2) Recommended program value 0x15 |
| 16-20 | **XTAL_FTUNE[4:0]** — Reference oscillator fine tuning for load capacitance. XTAL_FTUNE[4:0] selects from 0-5 pF in 32 steps of approximately 156 fF | 1) Default = 0x00<br>2) Recommended program value 0x10 |
| 12-15 | Reserved | |
| 8-11 | **XTAL_IBIAS_SEL[3:0]** — Reference oscillator bias select. These bits select different bias current for the oscillator amplifier.<br>**Note: Recommended that default be used as normal mode.** | Default = 0x1F |
| 7-0 | Reserved | |

## 5.9.17 32kHz XTAL Control (XTAL32_CNTL)

This register controls the use of the 32 kHz crystal oscillator. The register contents are retained in sleep mode.

| | XTAL32_CNTL | | | | | | | | Base+0x44 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | | | | | | | | | | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | | | | | | | | | | | XTAL32_GAIN | | | | | XTAL32_EN |
| TYPE | r | r | r | r | r | r | r | r | r | r | rw | rw | r | r | r | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |

**Table 5-24. XTAL32_CNTL Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 6-31 | Reserved | |
| 4-5 | **XTAL32_GAIN[1:0]** - 32kHz XTAL Gain Selector. These bits select the gain for the 32kHz crystal oscillator. Maximum gain is 2'b11. Minimum gain is 2'b00.<br>**Note: RECOMMENDED TO LEAVE AS DEFAULT .** | 2b11 = Maximum gain (default) |
| 1-3 | Reserved | |
| 0 | **XTAL32_EN** — 32kHz XTAL Enable. This bit turns on the XTAL32 oscillator.<br>• 2kHz Ring Oscillator should be turned off first.<br>• Engaging the XTAL32 must be performed while awake.<br>• It is also recommended that the RTC register be monitored to determine that XTAL32 has started. One can consider going into Bus Stealer Mode and waiting for 32kHz to be engaged.<br>• The state is retained in and out of sleep modes.<br>• Only a hard or soft reset can turn the XTAL32 off.One might consider going into Bus Stealer Mode and waiting for 32kHz to be engaged.<br>• Sometime after setting XTAL32_EN to 1 (up to 5sec), the 32kHz crystal will be ready. | 1 = XTAL32 ON<br>0 = XTAL32 OFF (Cold start default) |

## 5.9.18 Voltage Regulator Control (VREG_CNTL)

The VREG_CNTL register controls operation of a number of the power supply circuits. The register contents are not retained in sleep mode.

**NOTE**

All times stated here are based on a crystal of 26MHz (quickest warm-up). Lower frequency crystals will result in longer warm-up times. For example, a crystal of 13 MHz will nearly double the warm-up time for all regulators.

| | VREG_CNTL | | | | | | | | Base+0x48 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | | | | | | | | | | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | | | | | BUCK_CLKDIV[3:0] | | | | VREG_1P8V_EN | VREG_1P5V_SEL[1:0] | | VREG_1P5V_EN[1:0] | | BUCK_BYPASS_EN | BUCK_SYNC_REC_EN | BUCK_EN |
| TYPE | r | r | r | rw | rw | rw | rw | rw | r/w | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

**Table 5-25. VREG_CNTL Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 13-31 | Reserved | |
| 12 | Reserved | |
| 8-11 | **BUCK_CLKDIV[3:0]** — Divider Code Word for Buck Regulator Clock. These bits select the integer clock divider from decimal 2 to 16.<br>• See Table 5-28 for usage.<br>• The code word needs to be properly selected so that the resultant Buck Clock will be as close as possible to 1.6MHz.<br>• The actual range of resultant Buck Regulator Clocks will be 1.50MHz to 1.70MHz. For example, with a 24MHz reference oscillator, the integer divider should be 15 and the BUCK_CLKDIV code word will be set to 4'b1111 (reset default). This will yield a Buck Clock of 1.6MHz.<br>• No clock will be generated if the Buck is disabled. | 0xF = Divisor is 15 (default) |

**Table 5-25. VREG_CNTL Register Bit Descriptions (continued)**

| Bit Number | Description | Operation |
|---|---|---|
| 7 | **VREG_1P8V_EN** — NVM 1.8V Voltage Regulator Enable. This bit enables the 1.8V Voltage Regulator that supplies the Non-Volatile Memory (NVM).<br>• This bit can be set only when MC1322x is awake, CPU running and not accessing the NVM.<br>• When first enabled: a) the voltage regulator first trickle charges the associated capacitor, b) after 200us, the full current drive capability of the 1.8V voltage regulator will be enabled.<br>**Note:** Do not start using the NVM before the 200us warm-up period - check the VREG_1P8V_RDY bit in the STATUS register for readiness. .<br>**Note:** If the system is configured with an External 1.8V Supply, then VREG_1P8V_EN bit will be ignored. In this power configuration, the external 1.8V supply is externally connected to the 1.8 voltage regulator output pin, i.e., the 1.8V regulator is completely physically bypassed. Ensure that the PWR_SOURCE control bits are properly set in SYS_CNTL register.<br>**Note:** If the Buck is enabled and then the 1.8V regulator is enabled, this will cause the 1.8V regulator to be bypassed since the Buck regulator output will already be approximately 1.8V. Thus, the VREG_1P8V_EN still needs to be set to 1 and 200us is still needed for warm up (check the VREG_1P8V_RDY status bit). In this mode, the regulator will not actually be turned, however, it will be bypassed and time is needed to charge the associated capacitor. Make sure that the PWR_SOURCE control bits are properly set in SYS_CNTL register. | 1 = NVM Regulator enabled<br>0 = NVM Regulator disabled (default) |
| 5-6 | **VREG_1P5V_SEL[1:0]** - Analog 1.5V Voltage Regulator Current Selector These bits control the amount of current sourced out of the voltage regulator.<br>• The selection is defined by theTable 5-27 .<br>• This regulator must be set to source 20mA or 40 mA before the transceiver is used<br>• 40mA is recommended for nominal or higher TX power settings.<br>**Note:** Common usage is to enable the 40mA option for normal operation so that power settings need not be changed between RX and TX operations | 0x1 = 20mA source (default) |
| 3-4 | **VREG_1P5V_EN[1:0]** - Analog 1.5V Voltage Regulator Enable field. These bits turn on and off the 1.5V voltage regulators that supply the analog radio functions.<br>• These bits can be set only when MC1322x is awake and running in the "IDLE" mode (transceiver not active).<br>• Table 5-26 shows the possible selections and the amount of time needed for warmup.<br>• Software can monitor the VREG_1P5V_RDY to determine when the regulator is fully on.<br>• These regulators put the transceiver in the "RUN" mode<br>• Typical usage is to program all regulators to on after exiting low power mode.<br>**Note:** If ONLY the RX/TX Regulator is on (VREG_1P5V_EN[1:0]=2'b01) and then all regulators are needed on, then the VREG_1P5V_EN[1:0] must first be programmed to 2'b00 (disabled) and then re-programmed to 2'b11. | 0x0 = Off (default) |

**Table 5-25. VREG_CNTL Register Bit Descriptions (continued)**

| Bit Number | Description | Operation |
|---|---|---|
| 2 | **BUCK_BYPASS_EN** — Buck Regulator Bypass Enable. Setting this bit causes the system to bypass the Buck Voltage Regulator.<br>• Enable when the VBATT falls below 2.5V.<br>• Bypassing the buck will require 700us in order to allow the battery to charge the buck capacitor. Once the 700us has elapsed, the 1.5V and 1.8V regulators can be enabled.<br>• Application software can poll the BUCK_RDY bit in the CRM STATUS register to know when to continue with enabling the 1.5V or the 1.8V regulator.<br>• The BUCK_EN takes priority over this BUCK_BYPASS_EN. However, if the buck is being bypassed, it does not make sense to enable the buck regulator.<br>• If the system is configured with an External 1.8V Supply (i.e. SYSTEM_CNTL setting PWR_SOURCE = 2'b10), the BUCK_BYPASS_EN bit will be ignored.<br>**Note:** After POR, but before the use of the NVM, the system will not know if it is configured to use the Buck regulator. Therefore, the system will assume that a buck regulator will power the chip, and therefore, software MUST bypass the buck regulator whether there is a buck or not. Again, this is only needed after POR, but before enabling the 1.8V regulator and the NVM.The NVM Regulator must be disabled before switching the Buck Regulator from regulation to bypass mode. Check the VREG_BUCK_RDY set to 1 for a successful switch. This is handled by default in the ROM-based boot loader. | 1 = Buck regulator bypass enabled<br>0 = Bypass disabled (default) |
| 1 | **BUCK_SYNC_REC_EN** — Buck Synchronous Rectifier Voltage Regulator Enable. This bit enables the synchronous rectifier of the Buck Regulator. It should always be set when the Buck regulator is enabled. | 1 = Synch rect regulator enabled<br>0 = Synch rect regulator disabled (default) |
| 0 | **BUCK_EN** — Buck Voltage Regulator Enable. Setting this bit to 1 enables the buck regulator.<br>• The 1.5V and 1.8V regulators (for analog and NVM blocks) are powered by the buck regulator through the LREG_BK_FB pin.<br>• The application software must allow 400us after the Buck is enabled to turn on either the 1.5V or the 1.8V Regulators.<br>• Software can poll the BUCK_RDY bit in the CRM STATUS register to know when to turn-on the 1.5V or the 1.8V regulators.<br>**Note:** If the system is configured with an External 1.8V Supply (i.e. SYSTEM_CNTL setting PWR_SOURCE = 2'b10), the BUCK_EN bit will be ignored.<br>**Note:** The Buck Regulator as a switcher is not efficient when VBATT is 2.5V or less. When VBATT <= 2.5V, the Buck Regulator must be bypassed, i.e., BUCK_BYPASS_EN bit must be set. This bypass will take 700us. Software can poll the VREG_BUCK_RDY bit in the STATUS register to know when to continue with turning on the 1.5V or the 1.8V regulator.<br>• If the Buck is enabled and then the 1.8V regulator is enabled, this will cause the 1.8V regulator to be bypassed since the Buck regulator output will already be approximately 1.8V. | 1 = Buck regulator enabled<br>0 = Buck regulator disabled (default) |

**Table 5-26. Analog 1.5 V Voltage Regulator Enable**

| VREG_1P5V_EN[1:0] | Regulator(s) Turned On | Time for Warm-up |
|---|---|---|
| 2'b00 | None | - |
| 2'b01 | RX/TX Regulator | 200us |
| 2'b11 | RX/TX plus PLL Regulators | 600us |
| 2'b10 | Not Legal | - |

**Table 5-27. 1.5V Voltage Regulator Current Selector**

| VREG_1P5V_SEL[1:0] | Current Sourced | Reason |
|---|---|---|
| 2'b00 | 4mA | Pre-Transceiver State |
| 2'b01 (default) | 20mA | Normal Transceiver |
| 2'b11 | 40mA | 6dB Transceiver |
| 2'b10 | Not Legal | Not Legal |

**Table 5-28. Buck Clock Settings**

| BUCK_CLKDIV[3:0] | Actual Integer Divisor | Frequency Range of REF XTAL in which to use Divisor (MHz) | Resultant Buck Clock Frequency (MHz) |
|---|---|---|---|
| 4'b0001 | Constant Phase | Reserve for Test | N/A |
| 4'b0010<br>to<br>4'b0111 | 2<br>to<br>7 | N/A<br>Reserve for Test | N/A |
| 4'b1000 | 8 | 13.00 - 13.59 | 1.625 - 1.699 |
| 4'b1001 | 9 | 13.60 - 15.19 | 1.511 - 1.688 |
| 4'b1010 | 10 | 15.20- 16.79 | 1.520 - 1.679 |
| 4'b1011 | 11 | 16.80 - 18.39 | 1.527 - 1.672 |
| 4'b1100 | 12 | 18.40 - 19.99 | 1.533 - 1.666 |
| 4'b1101 | 13 | 20.00 - 21.59 | 1.538 - 1.661 |
| 4'b1110 | 14 | 21.60 - 23.19 | 1.543 - 1.656 |
| 4'b1111 (default) | 15 | 23.20 - 24.79 | 1.547 - 1.653 |
| 4'b0000 | 16 | 24.80 - 26.00 | 1.550 - 1.625 |

## 5.9.19 Software Reset (SW_RST)

Writing to the Software Reset register causes a system reset, which provides a means to accomplish a system reset via software control. Only 32-bit accesses are used. The application must write the read return value to cause the reset.

| | SW_RST | | | | | | | | **Base+0x50** | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | SW_RST[31:16] | | | | | | | | | | | | | | | |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | SW_RST[15:0] | | | | | | | | | | | | | | | |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |

**Table 5-29. SW_RST Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 0-31 | SW_RST[31:0] — Software Reset. The Software Reset register is used to initiate a complete reset of the system.<br>• To initiate the reset, write the value 0x87651234 to the SW_RST register.<br>• The value must be written as a 32-bit access - sequential byte or 16-bit writes will not cause a reset.<br>• Reading this register returns 0x87651234. | 0x87651234 = Default |

# Chapter 6
# MC1322x IEEE 802.15.4 2.4 GHz ISM Band Transceiver

## 6.1 Introduction

The MC1322x Platform-in-Package (PiP) provides a complete onboard system solution for the IEEE 802.15.4 Standard from the RF Physical Layer requirements through a complete MAC Layer implemented via dedicated hardware as well as ROM-based code. Key characteristics of the MC1322x 802.15.4 implementation include:

- 2.4 GHz ISM Band operation
- Over-the-air data rate of 250 Kbs
- 16 Standard independent channels
- Programmable transmitter output power through MAC services
- Uses carrier sense multiple access with collision avoidance (CSMA-CA) protocol
- Provides Energy Detection (ED), Clear Channel Assessment (CCA), and Link Quality Indication (LQI) through MAC services
- Dedicated hardware (MACA and DMA) to off load 802.15.4 services from CPU
- Hardware acceleration for AES security services of combined counter with CBC-MAC (cipher block chaining message authentication code)

The MC1322x transceiver implementation consists of the 2.4 GHz radio, modem, and MAC Accelerator (MACA). These elements combined with the ROM-resident MAC services running on the CPU provide the complete 802.15.4 service.

This chapter first provides an overview and general description for the radio and the modem. The MACA is the primary control block and path for TX and RX data and is described in Chapter 9, “MAC Accelerator (MACA)”. After an understanding of the transceiver operation is given, a discussion is presented of how the transceiver is used and controlled through the onboard software services. Finally, the need to use a special PLL for the receive modem when the reference frequency is NOT the typical 24 MHz is explained.

**NOTE**

Because of the complete onboard solution available in this device (from the RF front-end to the ROM’ed code), the control and use of the transceiver has been highly abstracted. The impact to the user is that Freescale provides software interfaces to set power levels, set channel frequencies, set receive modes, assess ED levels, and other control functions. These interfaces are integral within the software codebases available through the Freescale BeeKit environment. As a result, actual control details for the transceiver are not provided to the user.

## 6.2 Transceiver Overview

The MC1322x transceiver is capable of complete standalone operation. The MACA provides highest level control to the radio and modem and has the DMA for passing data to/from the RAM. The CPU is released to do other parallel tasks and also eliminates any timing dependency required to move data between the transceiver and memory. Also, the DMA works on a cycle-steal basis from the CPU bus and as a result is very quick; data are transferred on a byte-by-byte as-required basis.

Figure 6-1 shows a simplified block diagram of the MC1322x IEEE 802.15.4 transceiver. As the diagram shows, the transceiver is composed of the RF synthesizer, radio (analog), modem and MACA blocks.

- The radio contains all the analog circuitry used to transmit and receive an RF signal. On transmission, the radio is provided the encoded serial data stream from the modem and uses it to modulate the transmitted carrier signal. On reception, the radio selects the desired channel signal, amplifies and filters it, and then digitizes it for demodulation in the modem.
- The RF synthesizer provides the RF reference frequencies required by the radio to modulate and receive the desired signal.
- The modem receives/passes actual frame data from/to the MACA. For TX, the modem provides symbol generation and serialization of the data that is passed to the radio. For RX the modem receives the demodulated digitized data from the radio, does correlation, symbol sync and detection, and passes the received frame data to the MACA.
- The MACA provides the top level of sequence control for TX, RX, and CCA operations. It manipulates frame data transfer between the MACA and RAM through the DDMA, and off loads general transceiver control from the CPU.

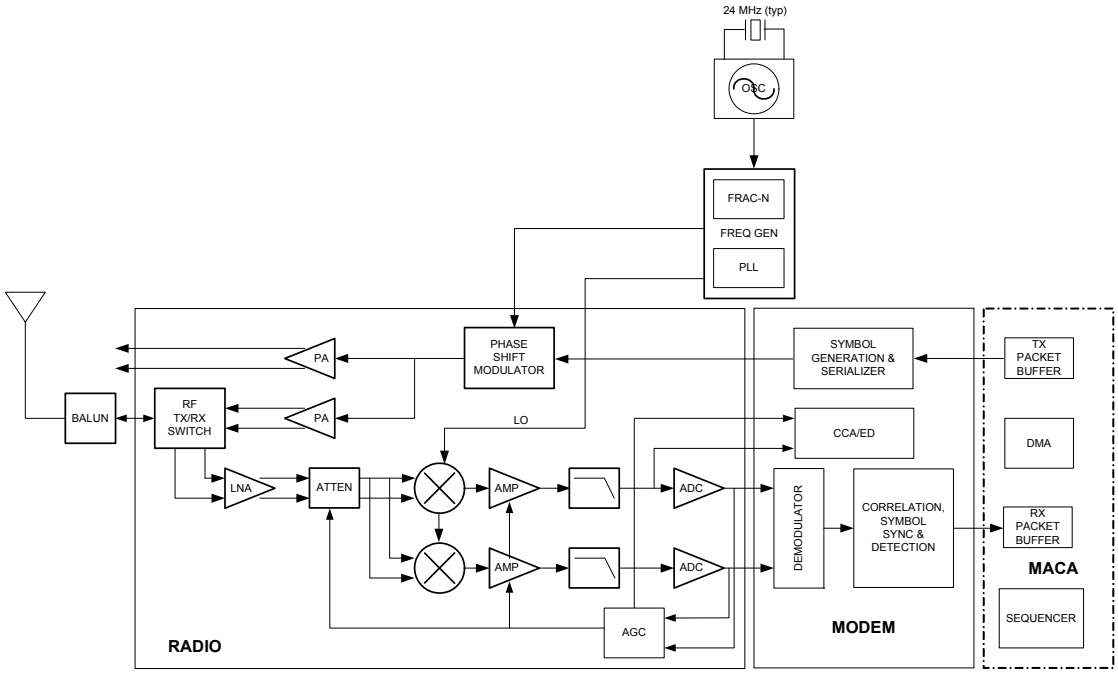



**Figure 6-1. MC1322x IEEE 802.15.4 Transceiver Block Diagram**

### 6.2.1 RF Synthesizer

Using the reference oscillator (typically 24 MHz) as a clock source, the RF synthesizer controls the activation and operation of the RF VCO. The VCO provides the 2.4 GHz frequency required for transmission or reception. The VCO frequency is the carrier that gets modulated for transmission, and on reception the VCO provides the LO (local oscillator) frequency used in the mixer. The synthesizer uses a fractional-N divisor architecture in conjunction with a PLL to frequency-lock the RF VCO to the reference clock.

Host CPU control of the RF synthesizer is done through software interfaces provided by the ROM resident MAC services. Programming of the VCO frequency is dependent on the desired 802.15.4 channel and the system reference clock frequency, and it controlled through application input. Table 6-1 lists the 802.15.4 2450 MHz Band channels.

**Table 6-1. IEEE 802.15.4 Standard 2450 MHz Band[1] Channels**

| Channel No | Frequency (MHz) |
|---|---|
| 11 | 2405 |
| 12 | 2410 |
| 13 | 2415 |
| 14 | 2420 |
| 15 | 2425 |
| 16 | 2430 |
| 17 | 2435 |
| 18 | 2440 |
| 19 | 2445 |
| 20 | 2450 |
| 21 | 2455 |
| 22 | 2460 |
| 23 | 2465 |
| 24 | 2470 |
| 25 | 2475 |
| 26 | 2480 |

1 The full frequency band is 2400-2483.5 MHz.

### 6.2.2 IEEE 802.15.4 Packet Structure

For reference, Figure 6-2 shows the IEEE 802.15.4 packet structure supported by the MC1322x. Payloads of up to 125 bytes are possible with the additional 2-byte FCS field. A four-byte preamble, a one-byte Start of Frame Delimiter (SFD), and a one-byte Frame Length Indicator (FLI) (value = 9-127) appear in front of the data load. A Frame Check Sequence (FCS) is calculated and appended to the end of the data.

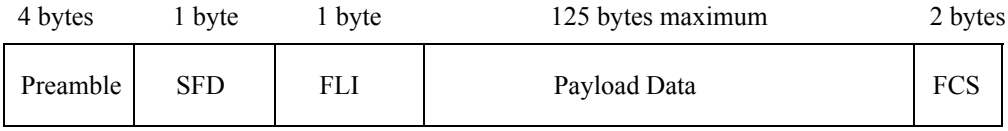



**Figure 6-2. MC1322x Packet Structure**

### 6.2.3 Transmit Path

The MACA provides the data path for both transmit and receive. For the transmit path, the MACA makes up the preamble, SFD, and FLI, and then fetches the payload data directly from memory with a simple DMA function. The DMA delivers the data on-demand on a byte-by-byte basis to the MACA FIFO designated as the TX packet buffer.

The MACA transfers data (at the bit-level) to the Tx modem. The modem transmitter block consists of an interface to the MACA that transforms the data to be transmitted based on the 802.15.4 standard into I/Q modulation signals for use by the radio analog phase shift modulator (PSM). The MACA transfers data already formatted with the correct 802.15.4 packet structure. The Tx Modem is responsible for the bit-to-symbol mapping, PN spreading, and PSM encoding. The encoded serial data is then passed from the modem to the radio.

In the radio transmit path the encoded data are used to modulate the carrier within the phase shift modulator (PSM) block. Then, the modulated signal is delivered to the RF analog power amplifier.

### 6.2.4 Receive Path

In the radio RF receive signal path, the RF input is first amplified in an LNA and then converted to parallel low IF In-phase and Quadrature (I & Q) signals through a single down-conversion stage. The IF signals are amplified, filtered, and the digitally sampled to convert the receive data to the digital domain.

The modem demodulator performs packet synchronization and data demodulation. It first demodulates the digitized data, and then the digital back-end performs Differential Chip Detection (DCD), a correlator “de-spreads” the Direct Sequence Spread Spectrum (DSSS) Offset QPSK (O-QPSK) signal, determines the symbols and packets, and detects the data.

The preamble, SFD, and FLI are parsed and used to detect the payload data and FCS. A two-byte FCS value is also calculated on the received data and compared to the FCS value appended to the transmitted data, generating a Cyclical Redundancy Check (CRC) result.

In the MACA, the data is accumulated in the Rx data packet buffer on a byte-by-byte basis. The payload data is passed directly to memory with a simple DMA function. The DMA delivers the data as it becomes available. The DMA always has highest bus priority.

The receiver demodulator includes a module called the Differential Chip Detector which has two modes of RX operation:

- Non-coherent Differential Chip Detection (DCD) Mode without automatic frequency control (AFC) - this is the default mode of operation in the Freescale applications. This is the more robust mode with the lowest current.
- Non-coherent Detection (NCD) Mode with AFC - This is a secondary mode of operation that can provide 3-4 dBm of increased sensitivity but increases the demodulator current drain about 3 mA.

The receiver block also supports ED level detection function for reporting CCA level and LQI for a received packet. The LQI level is calculated from several reported hardware levels and is reported as an 8-bit hex value from 0x00 to 0xFF compliant with the IEEE 802.15.4 Standard. Freescale's 802.15.4 MAC and 22xSMAC report LQI through function calls:

- The value 0x00 equates to about -100dBm
- The value 0xFF equates to about -15dBm

A simple conversion formula is:

Input Power (dBm) = ($LQI_{dec}$ / 3) - 100

Table 6-2 lists the function calls used in the MC13224 MAC codebases.

**Table 6-2. LQI Software Function CAlls**

| Codebase | Function Call (Primitive) |
|---|---|
| 802.15.4 MAC | MLME-SCAN.request |
| 22xSMAC | FuncReturn_t MLMELinkQuality (uint8_t * u8ReturnValue) |

## 6.2.5 Transceiver Timing Profiles

In the interest of looking at power requirements, Figure 6-3 shows simple models of the TX, RX, and CCA timing profiles (see also .Section 3.9.2.4, "RF Control Outputs"). The radio RF VCO and analog circuitry is not powered in an idle condition to save battery life. As a result, for any radio sequence there is a warmup time to allow the radio circuitry to stabilize. For TX, there also includes a 12 µs PA power ramp-up so as not to "splash" noise into the adjacent frequency channels.

The total "on" time for TX or RX is dependent on the packet length.

- For TX, the on time is consistent for a given packet length, knowing the number of data bytes as well as the packet overhead bytes. Also, there is an additional short 12 µs "warm down" time as the PA is powered down. Again, this is to prevent noise splatter as the PA turns off.
- For RX the time is more variable, because the receiver must be on in anticipation of the incoming packet, and once receiving the packet, the packet length will determine the additional on time.
- CCA on time is always the same based on 802.15.4 requirements.

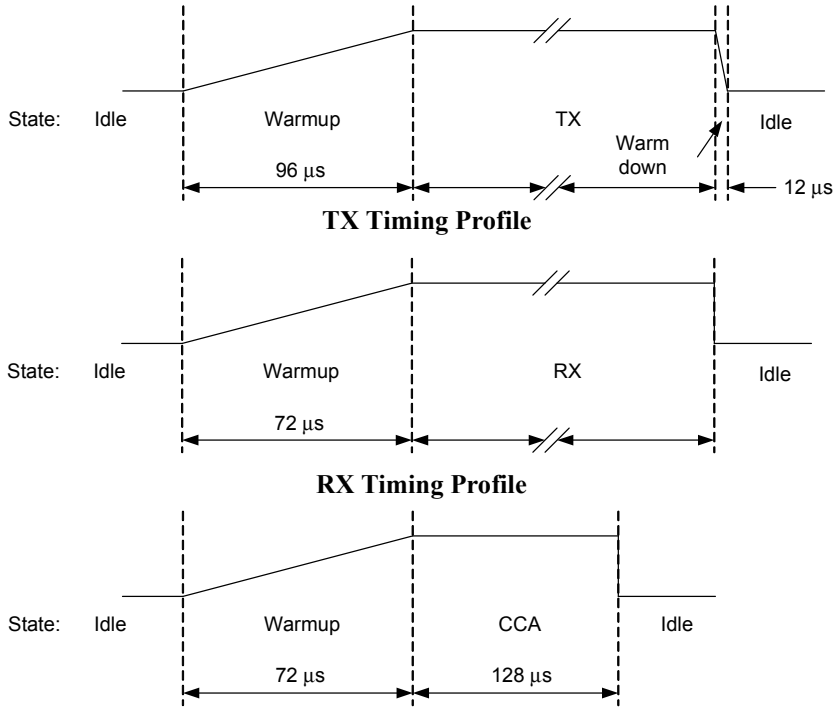



**Figure 6-3. MC1322x TX, RX, and CCA/ED Timing Profiles**

## 6.3 Transceiver Use and Control

The MC1322x provides a complete onboard set of IEEE 802.15.4 Standard functionality from the RF circuitry to the MAC sublayer services. This functionality is resident as dedicated hardware and dedicated code in ROM. Only the higher-level applications and communications services software for standards such as the ZigBee™ standard need to be resident in RAM.

Because of this complete, encapsulated concept, use of the transceiver and 802.15.4 services is done through the Freescale BeeKit™ Wireless Connectivity Toolkit. BeeKit is a standalone software application targeting Windows operating systems. BeeKit provides a graphical user interface (GUI) in which the users can create, modify, save and update wireless networking solutions. In addition to the graphical user interface, the BeeKit includes a comprehensive code base of wireless networking libraries, application templates and sample applications. This code base provides the networking software infrastructure which developers use when creating their own applications.

- The code base includes Freescale's ZigBee™ Protocol Stack (BeeStack™), pre-configured ZigBee application samples and templates.
- BeeKit also provides a path for inclusion of 802.15.4 MAC applications to the code base.
- System parameterization and setup are in configured in the BeeKit environment
- Initialization code is provided as part of the application build

- Channel selection, TX power selection, and RX mode selection are done through API calls in the services

In summary, the user is directed to the Freescale BeeKit™ toolkit and its documentation to understand how to configure and use the 802.15.4 transceiver and all the 802.15.4 services.

## 6.4 Modem Synthesizer

**NOTE**

This module is only used if the reference oscillator frequency is not the default of 24 MHz. The modem requires a clock of 24 MHz and this block generates that frequency if the reference oscillator is not the 24 MHz standard.

As described in Section 5.1.3.2, "Main System Clock Distribution" of Chapter 5, "System Management (Including CRM)" The modem synthesizer is part of the MC1322x clock distribution system. Figure 6-4 highlights the synthesizer subsystem.

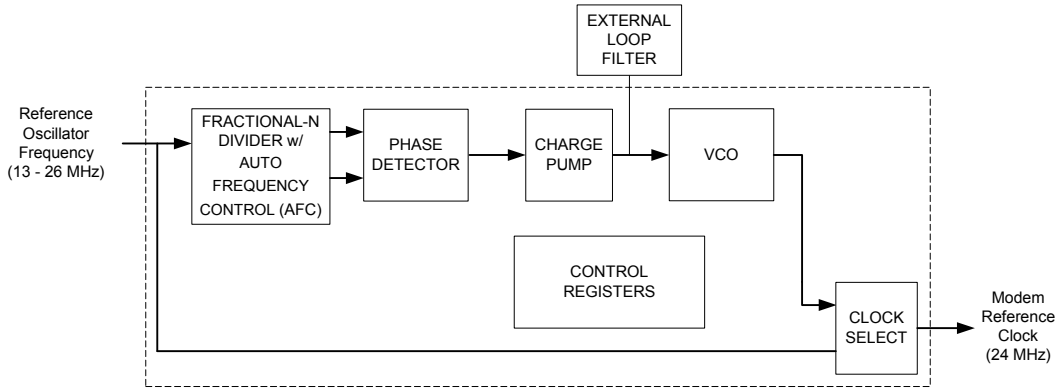



**Figure 6-4. Modem Synthesizer Block Diagram**

Features of the synthesizer include:

- Reference Clock — the main clock reference for the synthesizer is the reference crystal oscillator.
- Modem clock synthesizer PLL (the modem reference clock must always be 24 MHz in functional mode)
  - — A charge pump controls voltage-controlled oscillator (VCO).
  - — There is an external PLL loop filter which is a second order RC filter and is an integral part of the synthesizer loop. The charge pump and VCO are connected to the loop filter. This filter is discussed in Section 3.6.4, "Using a Reference Frequency Other Than 24 MHz".
  - — A fractional-N division (Σ–Δ Loop Divider) architecture along with digital automatic frequency control (AFC), resulting in less than 1Hz frequency resolution at the VCO
- Synthesizer Control Registers — The register map consists of a 256-word register field, of which only a subset is utilized. All registers are readable.
- Startup State Machine — The startup state machine is controlled by the **syn_en** register bit. This logic enables/disables the clocks in the Synthesizer, thereby placing the MC1322x modem in

battery-save. This logic optimally enables each analog component in the Synthesizer to reduce power consumption.

## 6.5 Modem Synthesizer Register Memory Map

The Modem Synthesizer module is programmed via a set of memory-mapped registers and their respective offset addresses are listed in Table 6-3.

**NOTE**

- Only 32-bit accesses to synthesizer registers are supported.
- Modem Synthesizer uses two base addresses

  Write base address is **0x8000_9000**

  Read base address is **0x8000_91C0**

**Table 6-3. Modem Synthesizer Register Memory Map**

| Address | Register Name | Access Type | Access Width |
|---|---|---|---|
| Base + 0x00 | Enable and Override Register (SYN_ENABLE) | R/W | 32-bit only |
| Base + 0x04 | Reserved | | |
| Base +0x08 | Reference Loop Divider Register (SYN_REFDIV) | R/W | 32-bit only |
| Base + 0x0C | VCO Loop Divider Register (SYN_VCODIV) | R/W | 32-bit only |
| Base + 0x0 - Base +0x28 | Reserved | | |

## 6.6 Register Descriptions

The following sections provide descriptions of the Synthesizer registers.

**NOTE**

The standard MC1322x initialization software disables the Modem Synthesizer and sets the MODEM_CLK_CTRL[1:0] field in the SYN_ENABLE Register to 01 for the reference oscillator as modem clock source.

### 6.6.1 Enable and Override Register (SYN_ENABLE)

The Enable and Override Register (SYN_ENABLE) is the primary control register for the modem synthesizer.

| **SYN_ENABLE** | | | | | | | | | | | | | **Addr Base+0x00** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | PLL_ACTIVE | SYN_EN | | | | | | | | | | | | | | |
| TYPE | rw | rw | r | r | r | r | r | r | r | r | rw | rw | rw | rw | rw | rw |
| RESET | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | | | | | | | MODEM_CLK_CTRL | | | | | | | | | |
| TYPE | r | r | r | r | r | r | rw | rw | r | r | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 6-4. SYN_ENABLE Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31 | **PLL_ACTIVE** - Control bit for SYN_EN control. This bit must be set to 1 to enable the SYN_EN control bit | 1 = Synthesizer is controlled by SYN_EN.<br>0 = Disables synthesizer and SYN_EN |
| 30 | **SYN_EN** — Enable bit for entire clock synthesizer. Writing a 1 to this bit enables the synthesizer startup state machine which starts and controls the synthesizer. **Note: PLL_ACTIVE must be asserted.** | 1 = Enables a synthesizer startup sequence.<br>0 = Disables the clock synthesizer. Synthesizer shutdown is immediate. Programming this bit to a 0 if it is already 0 will have no effect. |
| 29-22 | Reserved | Read only |
| 21-16 | Reserved | Read/write. Do not alter. |
| 9-8 | **MODEM_CLK_CTRL[1:0]** - Source and enable control for the modem clock. This 2-bit field controls the source for the modem reference clock. | 00 = The modem clock is forced to logic low.<br>01 = The modem clock is the reference oscillator (normal operation).<br>10 = Reserved (not used)<br>11 = The modem clock is the synthesizer 24 MHz VCO |
| 7-6 | Reserved | Read only |
| 5-0 | Reserved | Read/write. Do not alter. |

## 6.6.2 Reference Loop Divider Register (SYN_REFDIV)

The Reference Loop Divider Register (SYN_REFDIV) must be programmed for the synthesizer to provide 24 MHz to the modem if a reference frequency other than 24 MHz is used. This register should only be changed while the synthesizer is disabled (SYN_EN = 0). See Section 6.6.4, “Frequency Programming Example” for details on programming a desired frequency.

| SYN_REFDIV | | | | Reference Loop Divider | | | | | | | | | Addr Base+0x08 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | | | | | | | | | | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | | | | | | | | | PN | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

**Table 6-5. SYN_REFDIV Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-8 | **Reserved** | Read only |
| 7-0 | **PN[7:0]** - Integer value of the reference loop divider. This value determines the reference frequency. | Valid values are 1 through 255, where a “1” represents a feed through of the input reference clock and a “255” represents a divide by 255. See Section 6.6.4, “Frequency Programming Example” for details on programming a desired frequency. |

## 6.6.3 VCO Loop Divider Register (SYN_VCODIV)

The VCO Loop Divider Register (SYN_VCODIV) must be programmed for the synthesizer to provide 24 MHz to the modem if a reference frequency other than 24 MHz is used. This register should only be changed while the synthesizer is disabled (SYN_EN = 0). See Section 6.6.4, “Frequency Programming Example” for details on programming a desired frequency.

| SYN_VCODIV | VCO Loop Divider | | | | | | | | | | | | Addr Base+0x0C | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | RN | | | | | | | RAFC[24:16] | | | | | | | | |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | RAFC[15:0] | | | | | | | | | | | | | | | |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 6-6. SYN_VCODIV Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-25 | **RN[6:0]** — Integer value of the VCO loop divider. | Valid values are 1 through 127, where a “1” represents a divide by 1 of the VCO input clock and a “127” represents a divide by 127. An offset of 3 is present in this divider when fractionalization is enabled. When fractionalization is disabled, the value programmed in RN[6:0] becomes the actual divide modulo. |
| 24-0 | **RAFC[24:0]** — Fractional component of the divide value for the VCO loop divider. | This value feeds into the three accumulator sigma delta. See Section 6.6.4, “Frequency Programming Example” for details on programming a desired frequency. |

### 6.6.4 Frequency Programming Example

Multiple registers need to be programmed to synthesize the desired 24MHz modem frequency. These registers are shown below with a brief description.

**Table 6-7. Registers Requiring Programming Before Being Synthesized**

| Register Name | Description |
|---|---|
| RN[6:0] | Integer value of the VCO loop divider for Synthesizer |
| PN[7:0] | Integer value of the reference XTAL loop divider for Synthesizer |
| RAFC[24:0] | Fractional component of the divide value for the VCO loop divider |

Suppose a XTAL frequency of 26MHz is used to synthesize the VCO frequency of 24MHz. The equation shown in Figure 6-5 is used to calculate the values of **rn**, **pn**, and **rafc**.

$$Fvco = Fref \times \left( RN + 3 + \frac{RAFC}{2^{25}} \right)$$

**Figure 6-5. VCO Frequency Equation**

where,

*RN* = **rn**

*RAFC* = **rafc**

*Fref* = Reference Frequency (to the phase detector)

*Fvco* = VCO frequency to be synthesized (always 24 MHz)

A reference frequency (*Fref*) of 3.25MHz is selected by the designer. This yields a value of **pn** equal to 8 based on the equation shown in Figure 6-6.

$$Fref = \frac{Fxtal}{pn} = \frac{26MHz}{8} = 3.25MHz$$

**Figure 6-6. Reference Frequency Equation**

The desired VCO frequency of 24MHz is to be synthesized. Solving the equations shown in Figure 6-5 and Figure 6-6 results in the values shown in Figure 6-7.

$$24 = 3.25 \times \left( RN + 3 + \frac{RAFC}{2^{25}} \right)$$

$$\frac{24}{3.25} - 3 = \left( RN + \frac{RAFC}{2^{25}} \right)$$

$$4.384615385 = RN + \frac{RAFC}{2^{25}}$$

**Figure 6-7. Equation Resolution End Result**

This equation is then split into two equations to solve for the integer divide value (**rn**) and the fractionalization value (**rafc**) as shown in Figure 6-8.

$$rn = 4$$

$$rafc = \lfloor 0.384615385 \times 2^{25} \rfloor = 12905551$$

**Figure 6-8. Integer Divide Value and Fractionalization Value Equation**

*RAFC* is then converted to a hexadecimal base to yield a final value of 0xC4EC4E.

In summary, the final values to be programmed into **pn**, **rn**, and **rafc** are shown in Table 6-8.

**Table 6-8. Final Programming Values**

| Register | Final Value |
|---|---|
| pn | 0x08 |
| rn | 0x04 |
| rafc | 0xC4EC4E. |

# Chapter 7
# Central Processing Unit (CPU)

## 7.1 ARM7TDMI-S Processor Overview

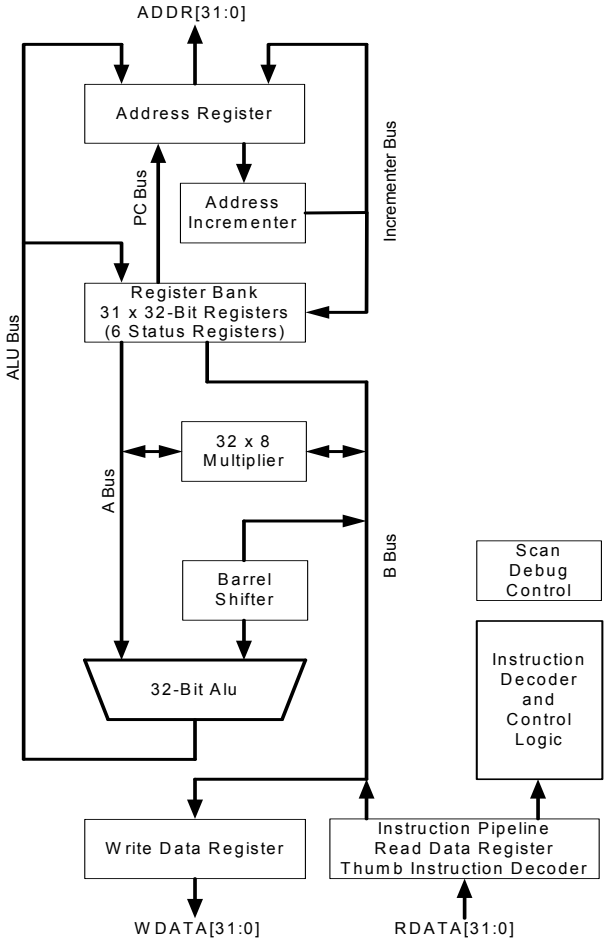



**Figure 7-1. ARM7TDMI-S 32-Bit CPU Core**

The MC1322x uses a ARM7TDMI-S processor (shown in Figure 7-1) which is a member of the ARM family of general-purpose 32-bit microprocessors that offers high performance for very low-power consumption and gate count. The ARM architecture is based on Reduced Instruction Set Computer (RISC) principles that give:

- High instruction throughput
- Excellent real-time interrupt response
- Low power

### 7.1.1 Memory Access

The ARM7TDMI-S CPU uses a single 32-bit data bus that carries both instructions and data. Only load, store, and swap instructions can access data from memory. The address bus is also 32 bits wide.

For data access on the data bus:

- The address is a byte address
- Data can be 8-bit bytes, 16-bit half words, or 32-bit words
- Words must be aligned to 4-byte boundaries
- Half words must be aligned to 2-byte boundaries

### 7.1.2 Memory Interface

The memory interface of the ARM7TDMI-S processor enables performance potential to be realized, while minimizing the use of memory. The MC1322x ARM7TDMI-S processor has three basic types of memory cycle:

- Internal cycle
- Nonsequential cycle
- Sequential cycle

### 7.1.3 Instruction Pipeline

The ARM7TDMI-S processor uses a pipeline to increase the speed of the flow of instructions to the processor. This enables several operations to take place simultaneously, and the processing, and memory systems to operate continuously.

A three-stage pipeline is used, so instructions are executed in three stages:

- Fetch
- Decode
- Execute.

The three-stage pipeline is shown in Figure 7-2.

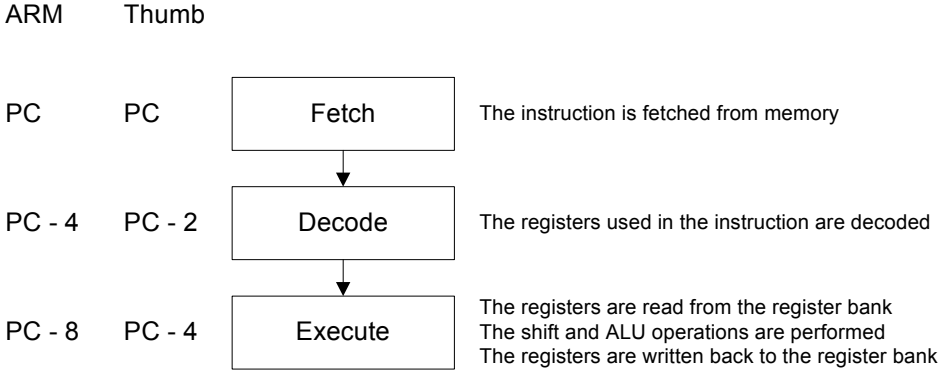



**Figure 7-2. The Instruction Pipeline**

The Program Counter (PC) points to the instruction being fetched rather than to the instruction being executed.

During normal operation, while one instruction is being executed, its successor is being decoded, and a third instruction is being fetched from memory.

## 7.2 ARM7TDMI-S Architecture

The ARM7TDMI-S processor has two instruction sets:

- The 32-bit ARM instruction set
- The 16-bit Thumb instruction set

The ARM7TDMI-S processor is an implementation of the ARM architecture v4T. For full details of both the ARM and Thumb instruction sets, see the *ARM Architecture Reference Manual*.

### 7.2.1 Instruction Compression

Microprocessor architectures traditionally had the same width for instructions and data. Therefore, 32-bit architectures had higher performance manipulating 32-bit data and could address a large address space much more efficiently than 16-bit architectures.

16-bit architectures typically had higher code density than 32-bit architectures, and greater than half the performance.

Thumb implements a 16-bit instruction set on a 32-bit architecture to provide:

- higher performance than a 16-bit architecture
- higher code density than a 32-bit architecture.

### 7.2.2 The Thumb Instruction Set

The Thumb instruction set is a subset of the most commonly used 32-bit ARM instructions. Thumb instructions are each 16 bits long, and have a corresponding 32-bit ARM instruction that has the same effect on the processor model.

Thumb instructions operate with the standard ARM register configuration, allowing excellent interoperability between ARM and Thumb states.

On execution, 16-bit Thumb instructions are transparently decompressed to full 32-bit ARM instructions in real time, without performance loss.

Thumb has all the advantages of a 32-bit core:

- 32-bit address space
- 32-bit registers
- 32-bit shifter and Arithmetic Logic Unit (ALU)
- 32-bit memory transfer.

Thumb therefore offers a long branch range, powerful arithmetic operations, and a large address space.

Thumb code is typically 65% of the size of the ARM code and provides 160% of the performance of ARM code when running on a processor connected to a 16-bit memory system. Thumb, therefore, makes the ARM7TDMI-S processor ideally suited to embedded applications with restricted memory bandwidth, where code density is important.

The availability of both 16-bit Thumb and 32-bit ARM instruction sets gives designers the flexibility to emphasize performance, or code size on a subroutine level, according to the requirements of their applications. For example, critical loops for applications such as fast interrupts and DSP algorithms can be coded using the full ARM instruction set and linked with Thumb code.

## 7.3 About the Programmer's Model

The ARM7TDMI-S processor core implements ARM architecture v4T, which includes the 32-bit ARM instruction set and the 16-bit Thumb instruction set. The programmer's model is described fully in the *ARM Architecture Reference Manual* Processor operating states

The ARM7TDMI-S processor has two operating states:

| | |
|---|---|
| ARM state | 32-bit, word-aligned ARM instructions are executed in this state. |
| Thumb state | 16-bit, half word-aligned Thumb instructions. |

In Thumb state, the Program Counter (PC) uses bit 1 to select between alternate half words. Transition between ARM and Thumb states does not affect the processor mode or the register contents.

### 7.3.1 Switching State

You can switch the operating state of the ARM7TDMI-S core between ARM state and Thumb state using the BX instruction. This is described fully in the *ARM Architecture Reference Manual*.

All exception handling is performed in ARM state. If an exception occurs in Thumb state, the processor reverts to ARM state. The transition back to Thumb state occurs automatically on return.

### 7.3.2 Memory Formats

The ARM7TDMI-S processor views memory as a linear collection of bytes numbered in ascending order from zero:

- bytes 0 to 3 hold the first stored word
- bytes 4 to 7 hold the second stored word
- bytes 8 to 11 hold the third stored word

The ARM7TDMI-S processor can normally treat words in memory as being stored in big-endian format or little-endian format. However, the processor on the MC1322x has been modified to support only little-endian format.

In little-endian format, the lowest-numbered byte in a word is considered the least-significant byte of the word, and the highest-numbered byte is the most significant. So byte 0 of the memory system connects to data lines 7 to 0. This is shown in Figure 7-3.

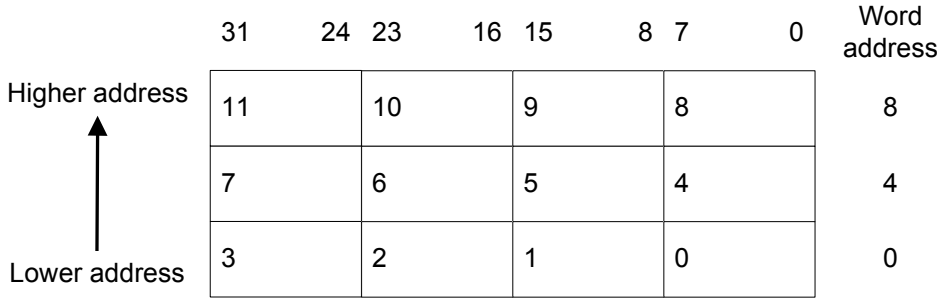



**Figure 7-3. Little-endian addresses of bytes within words**

## 7.4 Instruction Length

Instructions are either:

- 32 bits long (in ARM state)
- 16 bits long (in Thumb state)

## 7.5 Data Types

The ARM7TDMI-S processor supports the following data types:

- word (32-bit)
- half word (16-bit)
- byte (8-bit)

You must align these as follows:

- word quantities must be aligned to four-byte boundaries
- half word quantities must be aligned to two-byte boundaries
- byte quantities can be placed on any byte boundary

## 7.6 Operating Modes

The ARM7TDMI-S processor has seven operating modes:

- User mode is the usual ARM program execution state, and is used for executing most application programs.
- Fast interrupt (FIQ) mode supports a data transfer or channel process.
- Interrupt (IRQ) mode is used for general-purpose interrupt handling.
- Supervisor mode is a protected mode for the operating system.
- Abort mode is entered after a data or instruction prefetch abort.
- System mode is a privileged user mode for the operating system.
- Undefined mode is entered when an undefined instruction is executed.

Modes other than User mode are collectively known as privileged modes. Privileged modes are used to service interrupts, exceptions, or access protected resources.

## 7.7 Registers

The ARM7TDMI-S processor has a total of 37 registers:

- 31 general-purpose 32-bit registers
- 6 status registers.

These registers are not all accessible at the same time. The processor state and operating mode determine which registers are available to the programmer.

### 7.7.1 The ARM State Register Set

In ARM state, 16 general registers, and one or two status registers are accessible at any one time. In privileged modes, mode-specific banked registers become available. Figure 7-4 shows which registers are available in each mode.

The ARM state register set contains 16 directly-accessible registers, r0 to r15. An additional register, the Current Program Status Register (CPSR), contains condition code flags, and the current mode bits. Registers r0 to r13 are general-purpose registers used to hold either data or address values. Registers r14 and r15 have the following special functions:

Link register — Register 14 is used as the subroutine Link Register (LR).
r14 receives a copy of r15 when a Branch with Link (BL) instruction is executed.
At all other times you can treat r14 as a general-purpose register.
The corresponding banked registers r14_svc, r14_IRQ, r14_fiq, r14_abt, and r14_und are similarly used to hold the return values of r15 when interrupts and exceptions arise, or when BL instructions are executed within interrupt or exception routines.

Program counter — Register 15 holds the Program Counter (PC).
In ARM state, bits [1:0] of r15 are zero. Bits [31:2] contain the PC.
In Thumb state, bit [0] is zero. Bits [31:1] contain the PC.

In privileged modes, another register, the Saved Program Status Register (SPSR), is accessible. This contains the condition code flags, and the mode bits saved as a result of the exception that caused entry to the current mode.

See Section 7.8, “The Program Status Registers” for a description of the program status registers.

Banked registers have a mode identifier that shows to which User mode register they are mapped. These mode identifiers are shown in Table 7-1.

**Table 7-1. Register mode identifiers**

| Model | Mode identifier |
|---|---|
| User | usr |
| Fast interrupt | fiq |
| Interrupt | IRQ |
| Supervisor | svc |

**Table 7-1. Register mode identifiers (continued)**

| Model | Mode identifier |
|---|---|
| Abort | abt |
| System | sys |
| Undefined | und |

FIQ mode has seven banked registers mapped to r8–r14 (r8_fiq–r14_fiq). In ARM state, most of the FIQ handlers do not have to save any registers. The User, IRQ, Supervisor, Abort, and undefined modes each have two banked registers mapped to r13 and r14, allowing a private stack pointer and LR for each mode. Figure 7-4 shows the ARM state registers.

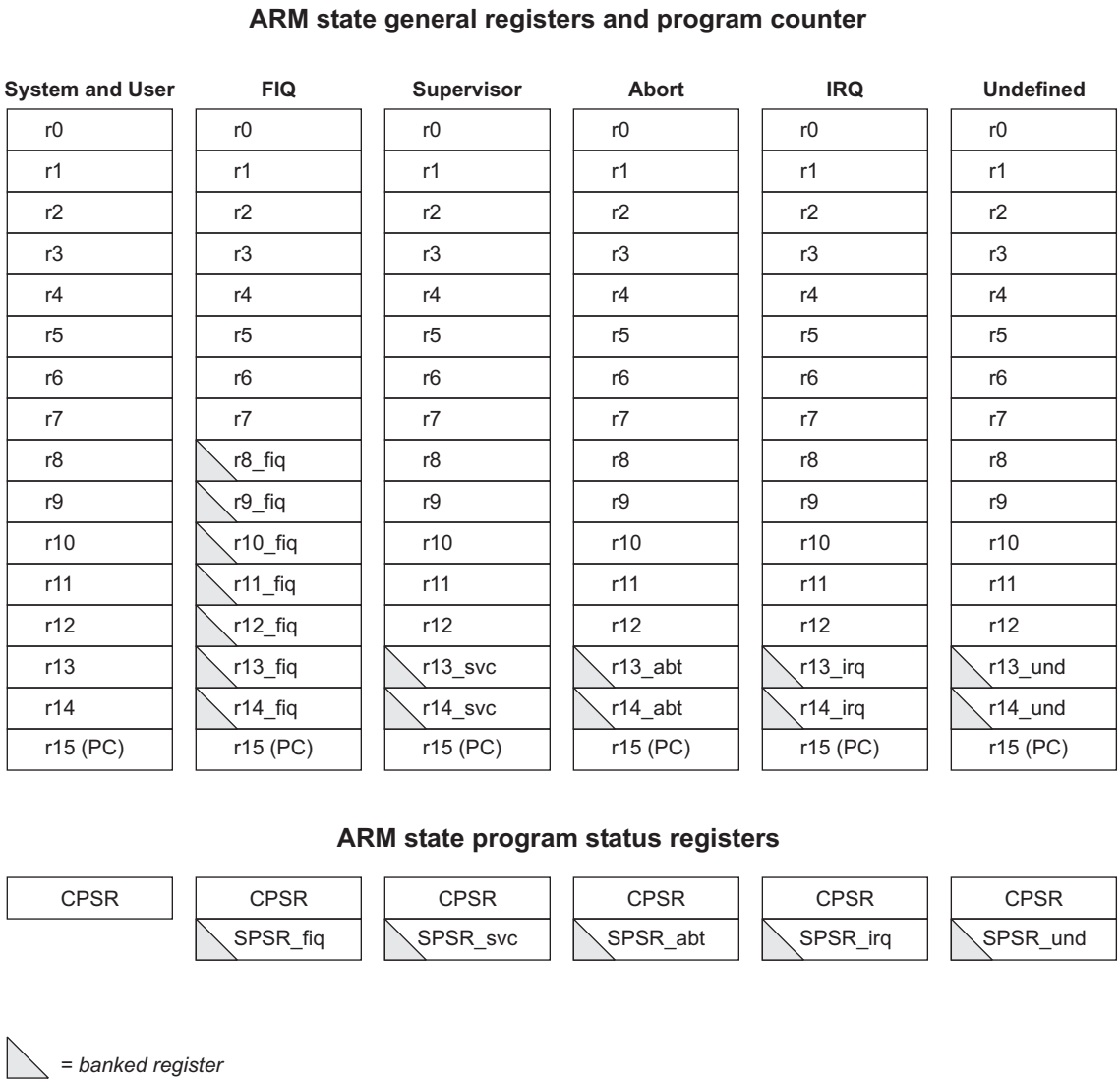



**Figure 7-4. Register organization in ARM state**

## 7.7.2 The Thumb State Register Set

The Thumb state register set is a subset of the ARM state set. The programmer has direct access to:

- eight general registers, r0–r7
- the PC
- a Stack Pointer (SP)
- a Link Register (LR)
- the CPSR.

There are banked SPs, LRs, and SPSRs for each privileged mode. This register set is shown in Figure 7-5.

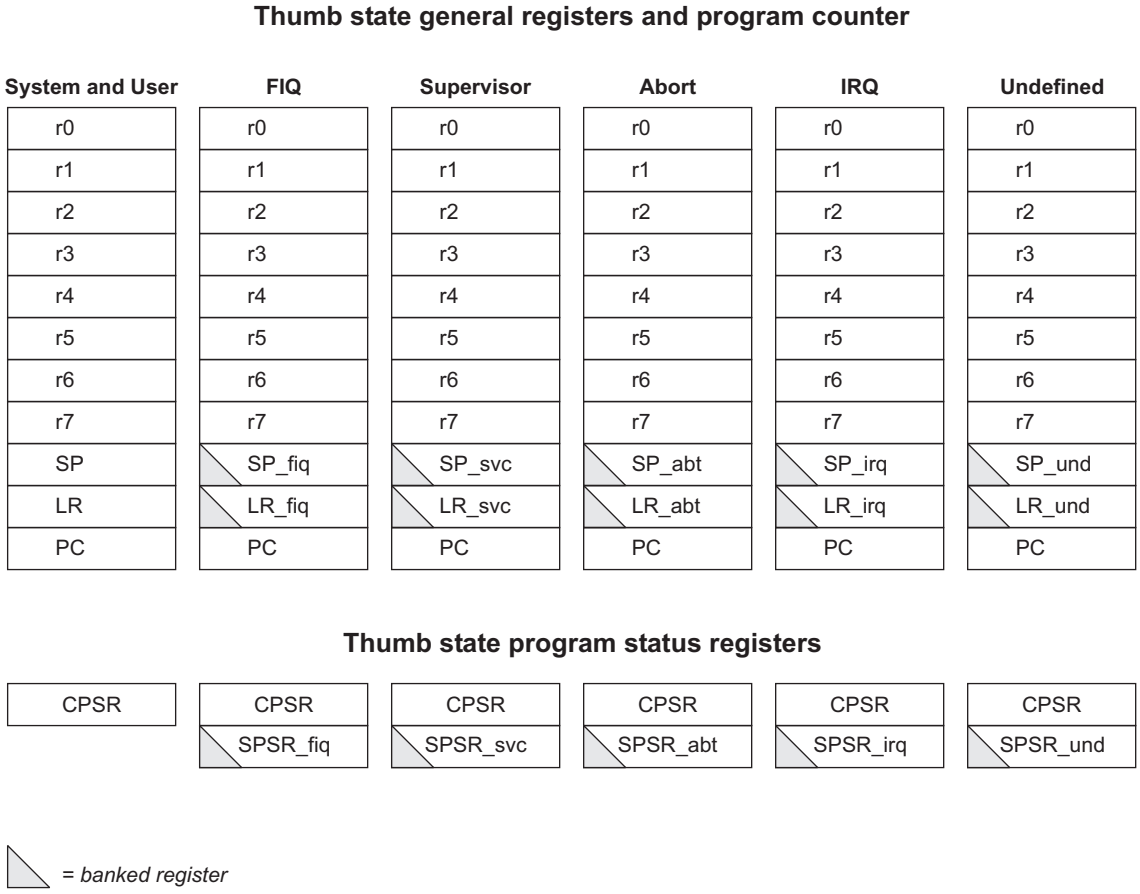



**Figure 7-5. Register organization in Thumb state**

### 7.7.3 The Relationship Between ARM State and Thumb State Registers

The Thumb state registers relate to the ARM state registers in the following way:

- Thumb state r0–r7, and ARM state r0–r7 are identical
- Thumb state CPSR and SPSRs, and ARM state CPSR and SPSRs are identical
- Thumb state SP maps onto ARM state r13
- Thumb state LR maps onto ARM state r14
- The Thumb state PC maps onto the ARM state PC (r15).

These relationships are shown in Figure 7-6.

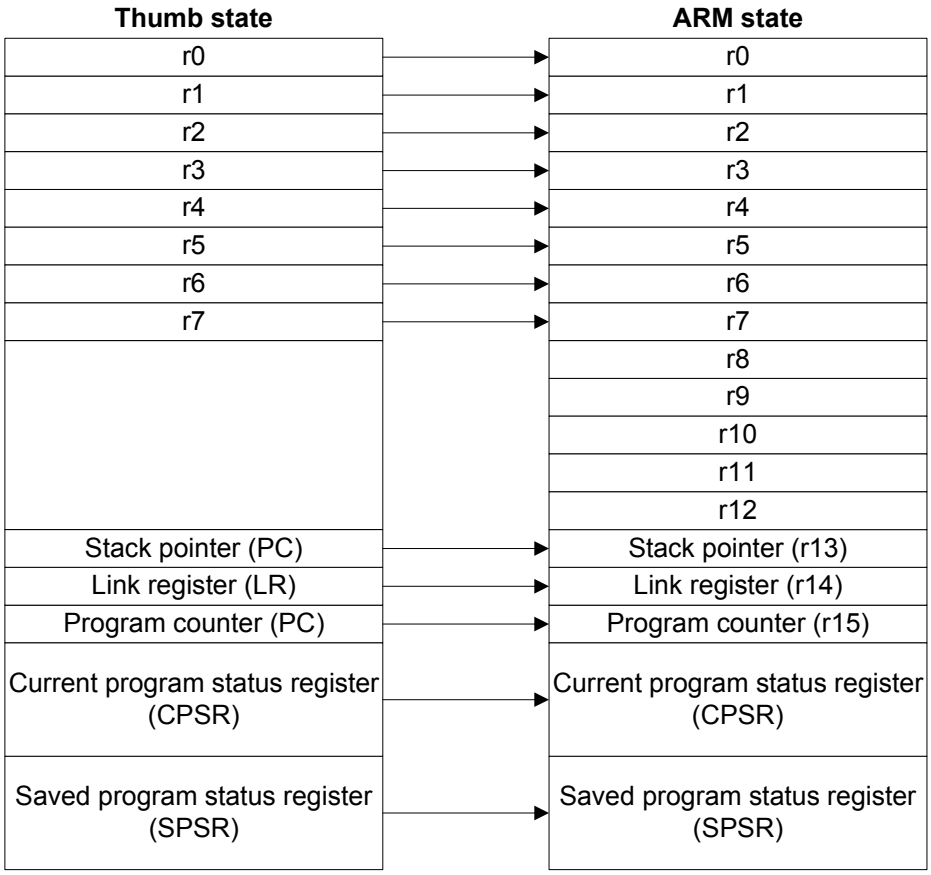



**Figure 7-6. Mapping of Thumb state registers onto ARM state registers**

Registers r0–r7 are known as the low registers. Registers r8–r15 are known as the high registers.

### 7.7.4 Accessing High Registers in Thumb State

In Thumb state, the high registers (r8–r15) are not part of the standard register set. The assembly language programmer has limited access to them, but can use them for fast temporary storage.

You can use special variants of the MOV instruction to transfer a value from a low register (in the range r0–r7) to a high register, and from a high register to a low register. The CMP instruction enables you to compare high register values with low register values. The ADD instruction enables you to add high register values to low register values. For more details, see the *ARM Architecture Reference Manual*.

## 7.8 The Program Status Registers

The ARM7TDMI-S core contains a CPSR and five SPSRs for exception handlers to use. The program status registers:

- hold the condition code flags
- control the enabling and disabling of interrupts
- set the processor operating mode

The arrangement of bits is shown in Figure 7-7.

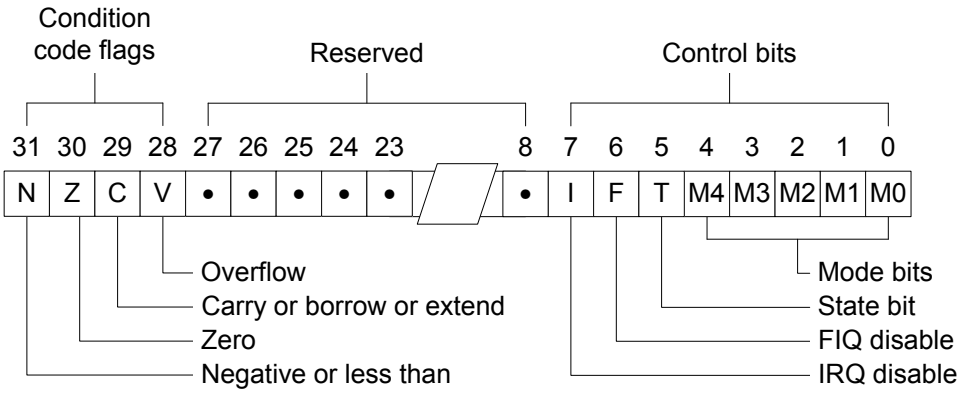



**Figure 7-7. Program status register format**

To maintain compatibility with future ARM processors, and as good practice, you are strongly advised to use a read-write-modify strategy when changing the CPSR.

### 7.8.1 The Condition Code Flags

The N, Z, C, and V bits are the condition code flags, You can set these bits by arithmetic and logical operations. The flags can also be set by MSR and LDM instructions. The ARM7TDMI-S processor tests these flags to determine whether to execute an instruction.

All instructions can execute conditionally in ARM state. In Thumb state, only the Branch instruction can be executed conditionally. For more information about conditional execution, see the *ARM Architecture Reference Manual*.

## 7.8.2 The Control Bits

The bottom eight bits of a PSR are known collectively as the control bits. They are the:

- Section 7.8.3, “Interrupt Disable Bits”
- Section 7.8.4, “T Bit”
- Section 7.8.5, “Mode Bits”

The control bits change when an exception occurs. When the processor is operating in a privileged mode, software can manipulate these bits.

## 7.8.3 Interrupt Disable Bits

The I and F bits are the interrupt disable bits:

- when the I bit is set, IRQ interrupts are disabled
- when the F bit is set, FIQ interrupts are disabled.

## 7.8.4 T Bit

The T bit reflects the operating state:

- when the T bit is set, the processor is executing in Thumb state
- when the T bit is clear, the processor executing in ARM state.

The operating state is reflected by the **CPTBIT** external signal.

Never use an MSR instruction to force a change to the state of the T bit in the CPSR. If you do this, the processor enters an unpredictable state.

## 7.8.5 Mode Bits

The M4, M3, M2, M1, and M0 bits (M[4:0]) are the mode bits. These bits determine the processor operating mode as shown in Table 7-2. Not all combinations of the mode bits define a valid processor mode, so take care to use only the bit combinations shown.

**Table 7-2. PSR mode bit values**

| M[4:0] | Mode | Visible Thumb state registers | Visible ARM state registers |
|---|---|---|---|
| 10000 | User | r0–r7, SP, LR, PC, CPSR | r0–r14, PC, CPSR |
| 10001 | FIQ | r0–r7, SP_fiq, LR_fiq PC, CPSR, SPSR_fiq | r0–r7, r8_fiq–r14_fiq, PC, CPSR, SPSR_fiq |
| 10010 | IRQ | r0–r7, SP_IRQ, LR_IRQ, PC, CPSR, SPSR_IRQ | r0–r12, r13_IRQ, r14_IRQ, PC, CPSR, SPSR_IRQ |
| 10011 | Supervisor | r0–r7, SP_svc, LR_svc, PC, CPSR, SPSR_svc | r0–r12, r13_svc, r14_svc, PC, CPSR, SPSR_svc |
| 10111 | Abort | r0–r7, SP_abt, LR_abt, PC, CPSR, SPSR_abt | r0–r12, r13_abt, r14_abt, PC, CPSR, SPSR_abt |
| 11011 | Undefined | r0–r7, SP_und, LR_und, PC, CPSR, SPSR_und | r0–r12, r13_und, r14_und, PC, CPSR, SPSR_und |
| 11111 | System | r0–r7, SP, LR, PC, CPSR | r0–r14, PC, CPSR |

If users program an illegal value into M[4:0], the processor enters an unrecoverable state.

### 7.8.6 Reserved Bits

The remaining bits in the PSRs are unused but are reserved. When changing a PSR flag or control bits make sure that these reserved bits are not altered. Also, make sure that your program does not rely on reserved bits containing specific values because future processors might have these bits set to one or zero.

## 7.9 Exceptions

Exceptions arise whenever the normal flow of a program has to be halted temporarily, for example to service an interrupt from a peripheral. Before attempting to handle an exception, the ARM7TDMI-S core preserves the current processor state so that the original program can resume when the handler routine has finished.

If two or more exceptions arise simultaneously, the exceptions are dealt with in the fixed order given in Section 7.9.10, “Exception Priorities”.

This section provides details of the exception handling on the ARM7TDMI-S processor:

- Section 7.9.1, “Exception Entry/Exit Summary”
- Section 7.9.2, “Entering an Exception”
- Section 7.9.3, “Leaving an Exception”

### 7.9.1 Exception Entry/Exit Summary

Table 7-3 shows the PC value preserved in the relevant r14 on exception entry and the recommended instruction for exiting the exception handler.

**Table 7-3. Exception Entry and Exit**

| Exception or entry | Return instruction | Previous state | | Notes |
|---|---|---|---|---|
| | | ARM r14_x | Thumb r14_x | |
| BL | MOV PC, R14 | PC + 4 | PC + 2 | Where the PC is the address of the BL, SWI, undefined instruction Fetch, or instruction that had the Prefetch Abort. |
| SWI | MOVS PC, R14_svc | PC + 4 | PC + 2 | |
| Undefined instruction | MOVS PC, R14_und | PC + 4 | PC + 2 | |
| Prefetch Abort | SUBS PC, R14_abt, #4 | PC + 4 | PC + 4 | |
| FIQ | SUBS PC, R14_fiq, #4 | PC + 4 | PC + 4 | Where the PC is the address of the instruction that was not executed because the FIQ or IRQ took priority. |
| IRQ | SUBS PC, R14_IRQ, #4 | PC + 4 | PC + 4 | |
| Data Abort | SUBS PC, R14_abt, #8 | PC + 8 | PC + 8 | Where the PC is the address of the Load or Store instruction that generated the Data Abort. |
| RESET | Not applicable | - | - | The value saved in r14_svc on reset is UNPREDICTABLE. |

### 7.9.2 Entering an Exception

When handling an exception the ARM7TDMI-S core:

1. Preserves the address of the next instruction in the appropriate LR. When the exception entry is from:

- ARM state, the ARM7TDMI-S copies the address of the next instruction into the LR (current PC + 4, or PC + 8 depending on the exception)
- Thumb state, the ARM7TDMI-S writes the value of the PC into the LR, offset by a value (current PC + 4, or PC + 8 depending on the exception)

The exception handler does not have to determine the state when entering an exception. For example, in the case of a SWI, MOVS PC, r14_svc always returns to the next instruction regardless of whether the SWI was executed in ARM or Thumb state.

2. Copies the CPSR into the appropriate SPSR.
3. Forces the CPSR mode bits to a value which depends on the exception.
4. Forces the PC to fetch the next instruction from the relevant exception vector.

The ARM7TDMI-S core also sets the interrupt disable flags on interrupt exceptions to prevent otherwise unmanageable nestings of exceptions.

Exceptions are always handled in ARM state. When the processor is in Thumb state and an exception occurs, the switch to ARM state takes place automatically when the exception vector address is loaded into the PC.

### 7.9.3 Leaving an Exception

When an exception is completed, the exception handler must:

1. Move the LR, minus an offset to the PC. The offset varies according to the type of exception, as shown in Table 7-4.
2. Copy the SPSR back to the CPSR.
3. Clear the interrupt disable flags that were set on entry.

The action of restoring the CPSR from the SPSR automatically restores the T, F, and I bits to whatever value they held immediately prior to the exception.

### 7.9.4 Fast Interrupt Request (FIQ)

The Fast Interrupt Request (FIQ) exception supports data transfers or channel processes. In ARM state, FIQ mode has eight private registers to remove the need for register saving (this minimizes the overhead of context switching). An FIQ is externally generated by taking the nFIQ signal input LOW.

Irrespective of whether exception entry is from ARM state, or from Thumb state, an FIQ handler returns from the interrupt by executing:

```
SUBS PC,R14_fiq,#4
```

You can disable FIQ exceptions within a privileged mode by setting the CPSR F flag. When the F flag is clear, the ARM7TDMI-S checks for a LOW level on the output of the FIQ synchronizer at the end of each instruction.

### 7.9.5 Interrupt Request (IRQ)

The Interrupt Request (IRQ) exception is a normal interrupt caused by a LOW level on the nIRQ input. IRQ has a lower priority than FIQ, and is masked on entry to an FIQ sequence. You can disable IRQ at any time, by setting the I bit in the CPSR from a privileged mode.

Irrespective of whether exception entry is from ARM state, or Thumb state, an IRQ handler returns from the interrupt by executing:

```
SUBS PC,R14_IRQ,#4
```

### 7.9.6 Abort

An abort indicates that the current memory access cannot be completed. It is signalled by the external ABORT input. The ARM7TDMI-S checks for the abort exception at the end of memory access cycles.

There are two types of abort:

- a Prefetch Abort occurs during an instruction prefetch
- a Data Abort occurs during a data access.

**NOTE**

- Memory access aborts are supported only for memory with the exception of the UART modules. All other peripheral functions do not support abort.
- Abort on misaligned accesses is not supported.

#### 7.9.6.1 Prefetch Abort

When a Prefetch Abort occurs, the ARM7TDMI-S core marks the prefetched instruction as invalid, but does not take the exception until the instruction reaches the execute stage of the pipeline. If the instruction is not executed because a branch occurs while it is in the pipeline, the abort does not take place.

After dealing with the reason for the abort, the handler executes the following instruction irrespective of the processor operating state:

```
SUBS PC,R14_abt,#4
```

This action restores both the PC and the CPSR and retries the aborted instruction.

#### 7.9.6.2 Data Abort

When a Data Abort occurs, the action taken depends on the instruction type:

- Single data transfer instructions (LDR, STR) write back modified base registers. The abort handler must be aware of this

- The swap instruction (SWP) aborts as though it had not been executed. (The abort must occur on the read access of the SWP instruction.)
- Block data transfer instructions (LDM, STM) complete. When write-back is set, the base is updated. If the instruction would have overwritten the base with data (when it has the base register in the transfer list), the ARM7TDMI-S prevents the overwriting

  The ARM7TDMI-S core prevents all register overwriting after an abort is indicated. This means that the ARM7TDMI-S core always preserves r15 (always the last register to be transferred) in an aborted LDM instruction

The abort mechanism enables the implementation of a demand-paged virtual memory system. In such a system, the processor is allowed to generate arbitrary addresses. When the data at an address is unavailable, the Memory Management Unit (MMU) signals an abort. The abort handler must then work out the cause of the abort, make the requested data available, and retry the aborted instruction. The application program does not have to know the amount of memory available to it, nor is its state in any way affected by the abort.

After fixing the reason for the abort, the handler must execute the following return instruction irrespective of the processor operating state at the point of entry:

```
SUBS PC,R14_abt,#8
```

This action restores both the PC, and the CPSR, and retries the aborted instruction.

### 7.9.7 Software Interrupt Instruction

The Software Interrupt (SWI) is used to enter Supervisor mode, usually to request a particular supervisor function. A SWI handler returns by executing the following irrespective of the processor operating state:

```
MOVS PC, R14_svc
```

This action restores the PC and CPSR, and returns to the instruction following the SWI. The SWI handler reads the opcode to extract the SWI function number.

### 7.9.8 Undefined Instruction

When the ARM7TDMI-S processor encounters an instruction that neither it nor any coprocessor in the system can handle, the ARM7TDMI-S core takes the undefined instruction trap. Software can use this mechanism to extend the ARM instruction set by emulating undefined coprocessor instructions.

The ARM7TDMI-S processor is fully compliant with the ARM architecture v4T, and traps all instruction bit patterns that are classified as undefined.

After emulating the failed instruction, the trap handler executes the following irrespective of the processor operating state:

```
MOVS PC,R14_und
```

This action restores the CPSR and returns to the next instruction after the undefined instruction.

For more information about undefined instructions, see the *ARM Architecture Reference Manual*.

### 7.9.9 Exception Vectors

Table 7-4 shows the exception vector addresses. In the table, columns I and F represent the previous value.

**Table 7-4. Exception vectors**

| ARM Address (ROM) | Functional Address[1] (RAM) | Exception | Mode on entry | I state on entry | F state on entry |
|---|---|---|---|---|---|
| 0x0000_0000 | 0x0040_0000 | Reset | Supervisor | Disabled | Disabled |
| 0x0000_0004 | 0x0040_0004 | Undefined instruction | Undefined | I | F |
| 0x0000_0008 | 0x0040_0008 | Software interrupt | Supervisor | Disabled | F |
| 0x0000_000C | 0x0040_000C | Abort (Prefetch) | Abort | I | F |
| 0x0000_0010 | 0x0040_0010 | Abort (Data) | Abort | I | F |
| 0x0000_0014 | 0x0040_0014 | Reserved | Reserved | - | - |
| 0x0000_0018 | 0x0040_0018 | IRQ | IRQ | Disabled | F |
| 0x0000_001C | 0x0040_001C | FIQ | FIQ | Disabled | Disabled |

[1] The normal ARM exception vectors are located in ROM and are not programmable. For all exceptions except Reset, the ROM vector function is a simple jump to the appropriate address in RAM such that the actual vector software handler can begin execution at the RAM vector address.

### 7.9.10 Exception Priorities

When multiple exceptions arise at the same time, a fixed priority system determines the order in which they are handled:

1. Reset (highest priority).
2. Data Abort.
3. FIQ.
4. IRQ.
5. Prefetch Abort.
6. Undefined instruction.
7. SWI (lowest priority).

Some exceptions cannot occur together:

- The Undefined Instruction and SWI exceptions are mutually exclusive. Each corresponds to a particular (non-overlapping) decoding of the current instruction.
- When FIQs are enabled and a Data Abort occurs at the same time as an FIQ, the ARM7TDMI-S core enters the Data Abort handler and proceeds immediately to the FIQ vector.

  A normal return from the FIQ causes the Data Abort handler to resume execution.

  Data Aborts must have higher priority than FIQs to ensure that the transfer error does not escape detection. You must add the time for this exception entry to the worst-case FIQ latency calculations in a system that uses aborts.

## 7.10 Interrupt Latencies

Interrupt latencies are described in:

- Section 7.10.1, “Maximum Interrupt Latencies”
- Section 7.10.2, “Minimum Interrupt Latencies”.

### 7.10.1 Maximum Interrupt Latencies

When FIQs are enabled, the worst-case latency for FIQ comprises a combination of:

- $T_{syncmax}$, the longest time the request can take to pass through the synchronizer. $T_{syncmax}$ is two processor cycles.
- $T_{ldm}$, the time for the longest instruction to complete. (The longest instruction is an LDM that loads all the registers including the PC.) $T_{ldm}$ is 20 cycles in a zero wait state system.
- $T_{exc}$, the time for the Data Abort entry. $T_{exc}$ is three cycles.
- $T_{fiq}$, the time for FIQ entry. $T_{fiq}$ is two cycles.

The total latency is therefore 27 processor cycles, slightly less than 0.7 microseconds in a system that uses a continuous 40MHz processor clock. At the end of this time, the ARM7TDMI-S executes the instruction at 0x1c.

The maximum IRQ latency calculation is similar, but must allow for the fact that FIQ, having higher priority, might delay entry into the IRQ handling routine for an arbitrary length of time.

### 7.10.2 Minimum Interrupt Latencies

The minimum latency for FIQ or IRQ is the shortest time the request can take through the synchronizer, $T_{syncmin}$ plus $T_{fiq}$ (four processor cycles).

## 7.11 Reset

When the nRESET signal goes LOW, the ARM7TDMI-S processor abandons the executing instruction.

When nRESET goes HIGH again the ARM7TDMI-S processor:

- Forces M[4:0] to b10011 (Supervisor mode).
- Sets the I and F bits in the CPSR.
- Clears the CPSR T bit.
- Forces the PC to fetch the next instruction from address 0x00.
- Reverts to ARM state and resumes execution.

After reset, all register values except the PC and CPSR are indeterminate.

# Chapter 8
# Interrupt Controller (ITC)

## 8.1 MC1322x MCU Interrupt Operation Overview

The MCU interrupt request service can be viewed as comprised of three basic functions as illustrated in Figure 8-1.

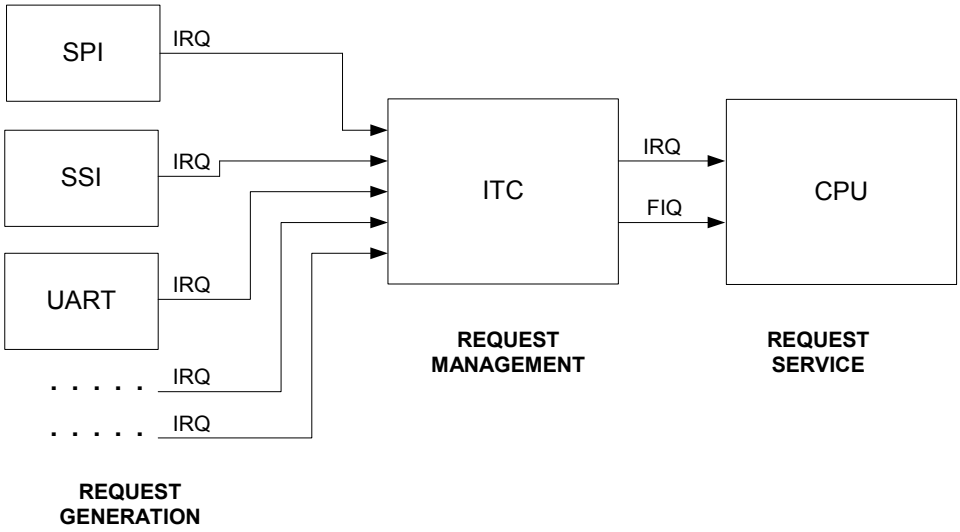



**Figure 8-1. Interrupt Request Service Functions**

- Request Generation - An interrupt request typically gets generated at a peripheral device. The peripheral control determines the source(s) of the interrupt request. There is a single request per peripheral.
- Request Management - The Interrupt Controller (ITC) accepts the interrupt request from each module source. These multiple requests are prioritized and mapped to one of two basic requests to the CPU, i.e., the fast interrupt request (FIQ) and the normal interrupt request (IRQ).
- Request Service - The CPU has the final responsibility of servicing the interrupt requests that have been mapped into the two simple signals of FIQ and IRQ.

The interrupt service is controlled as well as disabled or enabled across these three functions. Request generation is generated in the peripheral device.

**NOTE**

Freescale provides a software driver utility for the Interrupt Controller. This is briefly referenced in Section Appendix B, "MC1322x Software Driver Utilities" and is documented in the *MC1322x Software Driver Reference Manual*, (22XDRVRRM).

### 8.1.1 CPU Request Service

The CPU functionality is detailed in Chapter 7, "Central Processing Unit (CPU)" in greatest detail, but is summarized here as a review, and to show how the ITC relates to the request service.

The ARM7TDMI-S processor has seven operating modes and two of these are used to service interrupts:

- Fast interrupt (FIQ) mode supports a data transfer or channel process.
- Interrupt (IRQ) mode is used for general-purpose interrupt handling.

The interrupt modes are entered as exception processes. Exceptions arise whenever the normal flow of a program has to be halted temporarily, for example to service an interrupt from a peripheral. There are seven exception conditions and when multiple exceptions arise at the same time, a fixed priority system determines the order in which they are handled. The ARM7TDMI-S core imposes the following priority among the various exceptions:

- Reset (highest priority)
- Data abort
- Fast interrupt
- Normal interrupt
- Prefetch abort
- Undefined instruction and SWI (lowest priority)

The ARM7TDMI-S core has Program Status Registers (PSR) for exception handlers to use. The PSRs include a Current Program Status Register (CPSR) and five Stored Program Status Registers (SPSR).The program status registers:

- hold the condition code flags
- control the enabling and disabling of interrupts
- set the processor operating mode

The PSR arrangement of bits is shown in Figure 8-2.

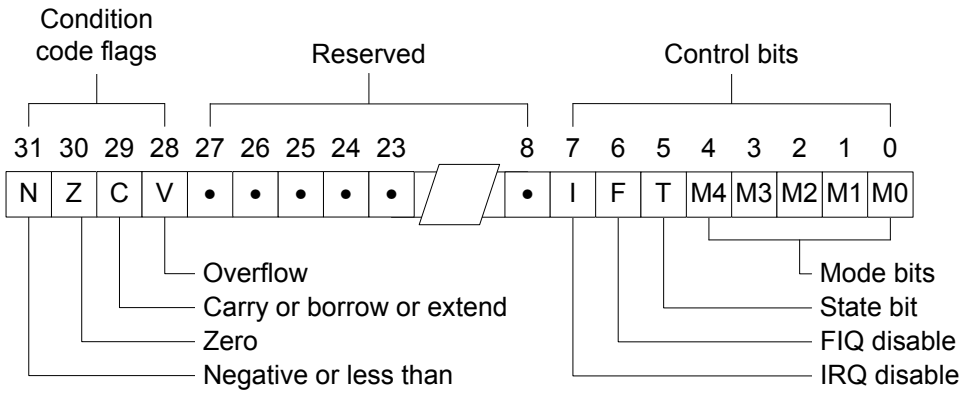



**Figure 8-2. Program Status Register Format**

The PSR bits important to interrupt service are the Interrupt Disable Bits:

- I Bit (IRQ Disable) - when the I bit is set, IRQ interrupts are disabled

- F Bit (FIQ Disable) - when the F bit is set, FIQ interrupts are disabled.

Before attempting to handle an exception, the ARM7TDMI-S core preserves the current processor state so that the original program can resume when the handler routine has finished. When handling an exception the ARM7TDMI-S core:

1. Preserves the address of the next instruction.
2. Copies the CPSR into the appropriate SPSR.
3. Forces the CPSR mode bits to a value which depends on the exception.
4. Forces the PC to fetch the next instruction from the relevant exception vector.

The ARM7TDMI-S core also sets the interrupt disable flags on interrupt exceptions to prevent otherwise unmanageable nestings of exceptions. Exceptions are always handled in ARM state.

When an exception service is completed, the exception handler must:

1. Restore the address of the next instruction.
2. Copy the SPSR back to the CPSR.
3. Clear the interrupt disable flags that were set on entry.

The action of restoring the CPSR from the SPSR automatically restores the T, F, and I bits to whatever value they held immediately prior to the exception.

#### 8.1.1.1 Fast Interrupt Request (FIQ)

The Fast Interrupt Request (FIQ) exception supports data transfers or channel processes and is caused by a request on the FIQ internal signal. In ARM state, FIQ mode has eight private registers to remove the need for register saving (this minimizes the overhead of context switching).

FIQ exceptions can be disabled within a privileged mode by setting the CPSR F flag.

#### 8.1.1.2 Interrupt Request (IRQ)

The Interrupt Request (IRQ) exception is a normal interrupt caused by a request on the IRQ internal signal. IRQ has a lower priority than FIQ, and is masked on entry to an FIQ sequence. IRQ can be disabled at any time, by setting the I bit in the CPSR from a privileged mode.

#### 8.1.1.3 Software Interrupt Instruction

It should be noted the CPU also supports a Software Interrupt (SWI) that is used to enter Supervisor mode, usually to request a particular supervisor function. This has no relation to interrupt requests coming from IRQ and FIQ and has no connection to the ITC.

### 8.1.2 Interrupt Request Generation

Interrupt requests are generated within a peripheral function. Each function has all its internal interrupt request sources channelled to its single IRQ signal. The ITC then priorities these requests and maps them to either the IRQ or FIQ inputs to the CPU.

**NOTE**

Interrupt requests for given modules are described in the chapter that describes that module. Users should see to the individual module descriptions in those chapters as needed.

## 8.2 Interrupt Controller Overview

The ITC is a peripheral function that resides on the CPU data bus which collects interrupt requests from the peripheral sources and provides an interface to the CPU core. The ITC consists of a set of control registers and associated logic to perform interrupt masking and priority mapping of requests to normal and fast interrupts. The priority levels can be controlled by software by simply masking interrupts. Figure 8-3 shows a logical block diagram of the ITC.

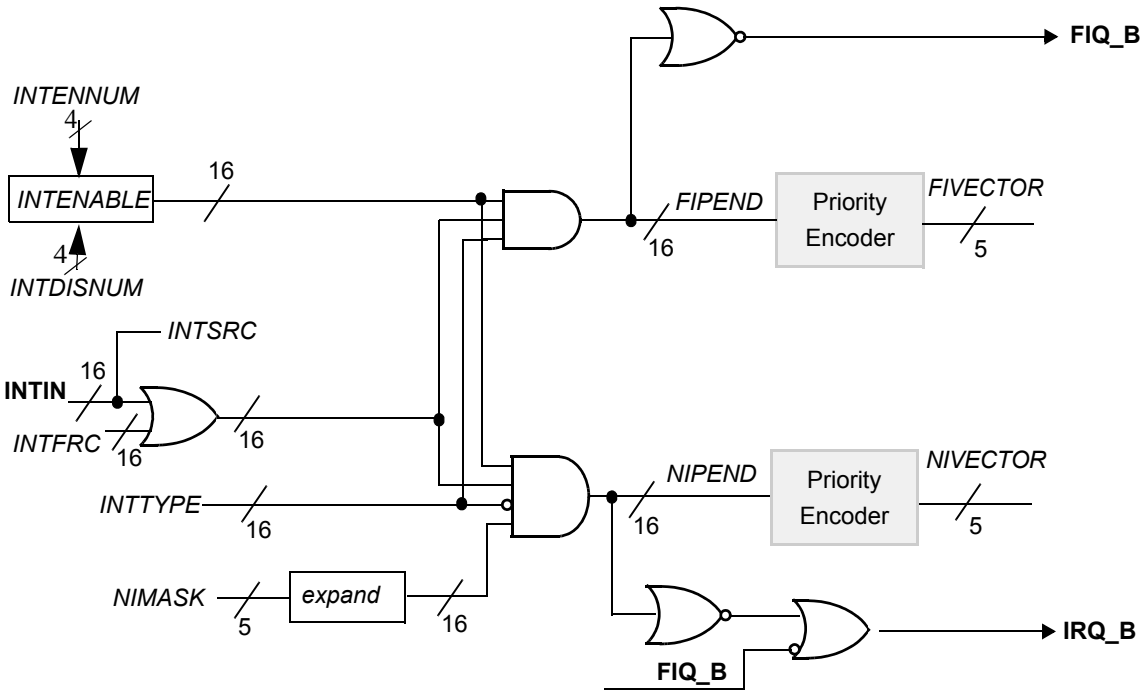



**Figure 8-3. ITC Logic Block Diagram**

## 8.3 ITC Features

The ITC provide the following features:

- Supports all internal interrupt sources
- Maps sources to normal and fast interrupt requests
- Indicates pending interrupt sources via a register for normal and fast interrupts
- Indicates highest priority interrupt number via register
- Independently enables/disables any interrupt source

- Provides a mechanism for software to force an interrupt

## 8.4 Interrupt Controller Operation

**NOTE**

This section covers a number of registers/fields that support 16 separate interrupt requests. However, due to the MC1322x peripherals implementation, only 11 individual interrupt requests are used (see Table 8-1).

Referring to Figure 8-3, the Interrupt Source Register (INTSRC) is a 16-bit status register with a single interrupt source associated with each of the 16 bits. An interrupt request signal (INTIN are module IRQ inputs) is routed from each interrupt source to the INTSRC Register. This allows up to 16 distinct interrupt sources in an implementation. As an aid to software development and debug, interrupt requests can also be forced to be asserted by way of the Interrupt Force Register (INTFRC). Each bit in the INTFRC Register is logically ORed with the corresponding hardware request line and the result is routed into the INTSRC Register inputs.

For each interrupt request, there is a corresponding bit in the Interrupt Enable Register (INTENABLE) which allows masking (enable/disable) of the individual bits of the INTSRC register. The INTENABLE bits can be set or cleared by a direct write, or alternatively, the Interrupt Enable Number Register (INTENNUM) can set one source at-a-time, or the Interrupt Disable Number Register (INTDISNUM) can clear (disable) one source at a time.

Once the interrupt request has been enabled (masked/unmasked), there is also an Interrupt Type Register (INTTYPE) which selects whether the interrupt source generates (is mapped to) a normal (IRQ) or fast (FIQ) interrupt to the ARM7TDMI-S core.

**NOTE**

Mapping and prioritization of interrupt requests is detailed in Section 8.4.1, “ITC Prioritization of Interrupt Sources”.

For reading status and to generate the hardware IRQ and FIQ signals to the CPU core, there are two interrupt pending registers:

- Normal Interrupt Pending Register (NIPEND) - this register indicates all pending normal interrupt requests.
  - Each bit is the equivalent of the logical AND of the INTSRC bit, the INTENABLE bit, and the NOT of the INTTYPE bit.
  - The NIPEND Register bits are bit-wise ORed together to form the IRQ signal to the ARM7TDMI-S core.
  - The IRQ core input signal is maskable by the Normal Interrupt Disable Bit (I bit) in the Processor Status Register (CPSR). The Normal Interrupt Vector (NIVECTOR) indicates the vector index of highest priority pending normal interrupt.
- Fast Interrupt Pending Register (FIPEND) - this register indicates all pending fast interrupt requests.

— Each bit is the equivalent of the logical AND of the INTSRC bit, the INTENABLE bit, and the the INTTYPE bit.
— The FIPEND Register bits are bit-wise NORed together to form the FIQ signal routed to the ARM7TDMI-S core.
— The FIQ core input signal is maskable by the Fast Interrupt Disable Bit (F bit) in the CPSR. The fast interrupt vector (FIVECTOR) indicates the vector index of highest priority pending fast interrupt.

A second set of registers provides indication of the highest level pending interrupt request:

- Normal Interrupt Vector (NIVECTOR) - this register indicates the vector number of highest priority pending normal interrupt.
- Fast Interrupt Vector (FIVECTOR) - this register indicates the vector number of highest priority pending fast interrupt.

All interrupt controller registers are accessible in both supervisor and user modes. Writes attempted to read-only registers are ignored. These registers should be written with 32-bit stores only.

The Interrupt Force Register (INTFRC) is provided for software generation of interrupts. By enabling interrupts for these bit positions, software can force an interrupt request for debugging hardware interrupt service routines by providing an alternate method of interrupt assertion.

The interrupt requests are prioritized in the following sequence:

1. Fast interrupt requests, in order of highest number
2. Normal interrupt requests, in order of highest number

If two normal interrupts are asserted at the same time, the normal interrupt with the highest source number is selected.

### 8.4.1 ITC Prioritization of Interrupt Sources

The ITC module sets the ultimate priority of any interrupt request source. Prioritization is determined at two levels:

- The CPU has two interrupt request inputs, i.e., IRQ and FIQ. The fast interrupt request FIQ always has priority over the normal interrupt request IRQ.
- Once the CPU responds to an IRQ or FIQ, the pending interrupt requests are prioritized by source number where higher source numbers have higher priority.

To restate this, the interrupt requests are prioritized in the following sequence:

1. Fast interrupt requests, in order of highest source number.
2. Normal interrupt requests, in order of highest source number.

Table 8-1 shows the priority of the interrupt sources.

**Table 8-1. Interrupt Sources**

| Interrupt Number | Source Name |
|---|---|
| 15(highest) | not used |
| 14 | not used |
| 13 | not used |
| 12 | not used |
| 11 | not used |
| 10 | SPI |
| 9 | ADC |
| 8 | SSI |
| 7 | MACA |
| 6 | SPIF |
| 5 | TMR |
| 4 | I2C |
| 3 | CRM |
| 2 | UART2 |
| 1 | UART1 |
| 0 (lowest) | ASM |

## 8.4.2 Assigning and Enabling Interrupt Sources

As stated in the chapter overview, interrupt requests can be enabled/disabled in the three basic functions, i.e., the CPU core, the ITC, and the peripheral function. At the system level to fully enable an interrupt request:

- The first step is to program the peripheral source(s) to generate interrupt requests
- The second step is to enable and assign interrupt requests in the ITC
- The final step is to enable the IRQ and or FIQ interrupt inputs in the core by clearing the normal interrupt disable (I) and/or the fast interrupt disable (F) bits in the program status register (CPSR).

Within the ITC, it first prioritizes the sources in Table 8-1 via their bit position in the INTENABLE Register. The assigned bit position maps to the prioritization level, as an example, INTENABLE[2] enables the UART2 interrupt request).

Assigning and enabling of interrupt requests at the ITC is done at several levels:

1. An interrupt request from a given source must first be enabled via its assigned bit in the INTENABLE Register
   - The bits may be enabled or disabled directly by writing the INTENABLE Register.
   - One source at-a-time can be disabled via Interrupt Disable Number Register (INTDISNUM)
   - One source at-a-time can be enabled via Interrupt Enable Number Register (INTENNUM)

2. The enabled source must then be mapped to the IRQ or FIQ space via its assigned bit in the INTTYPE Register
3. Exclusive to the IRQ, all interrupt sources below a given level can be disabled via the Normal Interrupt Mask Register (NIMASK).
4. Finally, either or both the IRQ and FIQ can be enabled or disabled via the Interrupt Control Register (INTCNTL)

## 8.5 Interrupt Controller Memory Map

Table 8-2 provides an overview of the ITC register memory map.

- The ITC base address is 0x8002_0000.
- Register writes to ITC offset addresses above the FIPEND Register address (0x8002_0040 - 0x8002_01FF) are ignored.
- Register reads from write-only registers and ITC addresses above the FIPEND Register offset address read as all 0's.

**Table 8-2. ITC Register Memory Map**

| Offset | [31:24] | [23:16] | [15:8] | [7:0] | Access Type | Access Width |
|---|---|---|---|---|---|---|
| + 0x00 | Interrupt Control Register (INTCNTL) | | | | R/W | 32-bit only write/ any read |
| + 0x04 | Normal Interrupt Mask Register (NIMASK) | | | | R/W | 32-bit only write/ any read |
| + 0x08 | Interrupt Enable Number Register (INTENNUM) | | | | W | 32-bit only write |
| + 0x0C | Interrupt Disable Number Register (INTDISNUM) | | | | W | 32-bit only write |
| + 0x10 | Interrupt Enable Register (INTENABLE) | | | | R/W | 32-bit only write/ any read |
| + 0x14 | Interrupt Type Register (INTTYPE) | | | | R/W | 32-bit only write/ any read |
| + 0x18 to +0x24 | Reserved | | | | - | - |
| + 0x28 | Normal Interrupt Vector (NIVECTOR) | | | | R | Any read |
| + 0x2C | Fast Interrupt Vector (FIVECTOR) | | | | R | Any read |
| + 0x30 | Interrupt Source Register (INTSRC) | | | | R | Any read |
| + 0x34 | Interrupt Force Register (INTFRC) | | | | R/W | 32-bit only write/ any read |
| + 0x38 | Normal Interrupt Pending Register (NIPEND) | | | | R | Any read |
| + 0x3C | Fast Interrupt Pending Register (FIPEND) | | | | R | Any read |

## 8.6 ITC Registers

The following sections provide descriptions of the ITC registers.

### 8.6.1 Interrupt Control Register (INTCNTL)

The Interrupt Control Register (INTCNTL) enables/disables the interrupt arbiters. This register should be updated with 32-bit stores only.

| INTCNTL | Interrupt Control Register | | | | | | | | | | | | Addr $8002_0000 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | | | | | | | | | | | | NIAD | FIAD | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | rw | rw | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | | | | | | | | | | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 8-3. INTCNTL Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| Bits 31-21 | Reserved | |
| Bit 20 | **NIAD** — Normal Interrupt Arbiter Disable. This bit, when asserted, prevents (or masks) the assertion of bus request to the core when the normal interrupt signal (nIRQ) is asserted. If an alternate master has ownership of the bus when a normal interrupt occurs, the bus is given back to the processor core after the DMA device has completed its accesses. Therefore, the IRQ_DIS bit does not affect alternate master accesses that are in progress.<br><br>In order to prevent an alternate master from accessing the bus during an interrupt service routine, the interrupt flag should not be cleared until the end of the service routine. | 0 = Allow normal (IRQ) interrupt request to CPU<br>1 = Mask or disable assertion of IRQ to CPU |

**Table 8-3. INTCNTL Bit Descriptions (continued)**

| Bit Number | Description | Operation |
|---|---|---|
| Bit 19 | **FIAD** — Fast Interrupt Arbiter Disable. This bit, when asserted, prevents the assertion of bus request to the core when the fast interrupt signal (nFIQ) is asserted. If an alternate master has ownership of the bus when a fast interrupt occurs, the bus is given back to the processor core after the DMA device has completed its accesses. Therefore, the IRQ_DIS bit does not affect alternate master accesses that are in progress.<br><br>In order to prevent an alternate master from accessing the bus during an interrupt service routine, the interrupt flag should not be cleared until the end of the service routine. | 0 = Allow fast (FIQ) interrupt request to CPU<br>1 = Mask or disable assertion of FIQ to CPU |
| Bits 18-0 | Reserved | |

## 8.6.2 Normal Interrupt Mask Register (NIMASK)

The Normal Interrupt Mask Register (NIMASK) controls the normal interrupt mask level. All normal interrupts (IRQ) with a priority lower than or equal to the NIMASK are disabled. The priority of normal interrupts are determined by their assigned interrupt number (see Section Table 8-1., “Interrupt Sources”). The reset state of this register does not disable any normal interrupts. This register should be updated with 32-bit stores only.

**NOTE**

Writing a value of 0x10 or greater to NIMASK[4:0] does not disable any normal interrupt.

| NIMASK | | | | Normal Interrupt Mask Register | | | | | | | | | Addr $8002_0004 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | | | | | | | | | | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | | | | | | | | | | | | NIMASK[4:0] | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |

**Table 8-4. NIMASK Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| Bits 31-3 | Reserved | |
| Bits 4-0 | **NIMASK[4:0]** — Normal Interrupt Mask. Controls normal interrupt mask level. All normal interrupts of priority level lower than or equal to the NIMASK are disabled. | 0 = Disable normal interrupt 0<br>1 = Disable normal interrupt 1 and lower<br>2 = Disable normal interrupt 2 and lower<br>...<br>11 - 31 = Do not disable any normal interrupts |

## 8.6.3 Interrupt Enable Number Register (INTENNUM)

The Interrupt Enable Number Register (INTENNUM) provides a hardware accelerated means of enabling of interrupt requests. Any write to the ENNUM[3:0] field enables one interrupt source. If the field equals 4b0000, then interrupt source 0 is enabled, etc. Multiple writes to this register enable a corresponding number of interrupts. This hardware mechanism alleviates the need for an atomic read/modify/write sequence to enable an interrupt source.

This register should be updated with 32-bit stores only. This register always reads back all 0s.

| INTENNUM | Interrupt Enable Number Register | | | | | | | | | | | | Addr $8002_0008 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | | | | | | | | | | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | | | | | | | | | | | | | ENNUM[3:0] | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | slfclr | slfclr | slfclr | slfclr |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 8-5. INTENNUM Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| Bits 31-4 | Reserved | |
| Bits 3-0 | **ENNUM[3:0] —** Interrupt Enable Number. Writing to this field enables the interrupt source associated with this value. | 0 = Enable interrupt source 0<br>1 = Enable interrupt source 1<br>...<br>10= Enable interrupt source 10<br>11 - 15 = Unused |

## 8.6.4 Interrupt Disable Number Register (INTDISNUM)

The Interrupt Disable Number Register (INTDISNUM) provides a hardware accelerated means of disabling of interrupt requests. Any write to this register disables one interrupt source. If the field equals

4b0000, then interrupt source 0 is disabled, etc. Multiple writes to this register disable a corresponding number of interrupts. This hardware mechanism alleviates the need for an atomic read/modify/write sequence to disable an interrupt source.

This register should be updated with 32-bit stores only. This register always reads back all 0s.

| INTDISNUM | Interrupt Disable Number Register | | | | | | | | | | | | Addr $8002_000C | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | | | | | | | | | | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | | | | | | | | | | | | | DISNUM[3:0] | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | slfclr | slfclr | slfclr | slfclr |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 8-6. INTDISNUM Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| Bits 31-4 | Reserved | |
| Bits 3-0 | **DISNUM[3:0]** — Interrupt Disable Number. Writing to this field disables the interrupt source associated with this value. | 0 = Disable interrupt source 0<br>1 = Disable interrupt source 1<br>...<br>10 = Disable interrupt source 10<br>11 - 15 = Unused |

## 8.6.5 Interrupt Enable Register (INTENABLE)

The Interrupt Enable Register (INTENABLE) is used to enable pending interrupt requests to the CPU core. Bits in this register correspond to an interrupt source available in the system (Sources 0 through 10 are used). The reset state of this register is all interrupts masked.

**NOTE**

- The interrupts enabled in this register can be modified by writing directly to the INTENABLE Register, or by writing the INTENNUM Register to set bits, or by writing the INTDISNUM Register to clear bits.
- See Table 8-1 for mapping of interrupt enable bits to interrupt sources.

This register should be updated with 32-bit stores only.

| INTENABLE | Interrupt Enable Register | | | | | | | | | | | | Addr $8002_0010 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | | | | | | | | | | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | INTENABLE[15:0] | | | | | | | | | | | | | | | |
| TYPE | rwm | rwm | rwm | rwm | rwm | rwm | rwm | rwm | rwm | rwm | rwm | rwm | rwm | rwm | rwm | rwm |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 8-7. INTENABLE Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| Bits 31-16 | Reserved | |
| Bits 15-0 | **INTENABLE[15:0]** — Interrupt Enable. These bits enable the corresponding interrupt source to request either a normal interrupt or a fast interrupt.<br>• A reset operation clears these bits.<br>• If an interrupt source is enabled, it is mapped to the IRQ or FIQ type request depending on the associated **INTTYPE** setting.<br>• Only Bits 10 - 0 map to a valid interrupt source | 0 = Interrupt disabled<br>1 = Interrupt enabled |

## 8.6.6 Interrupt Type Register (INTTYPE)

The Interrupt Type Register (INTTYPE) is used to select whether a pending interrupt source, when enabled by the INTENABLE register, creates a normal interrupt request (IRQ) or a fast interrupt request (FIQ) to the CPU core. Bits in this register correspond to an interrupt source available in the system (Sources 0 through 10 are used). The reset state of this register is that all interrupts generate a normal interrupt.

| INTTYPE | Interrupt Type Register | | | | | | | | | | | | Addr $8002_0014 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | | | | | | | | | | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | INTTYPE[15:0] | | | | | | | | | | | | | | | |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 8-8. INTTYPE Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| Bits 31-16 | Reserved | |
| Bits 15-0 | **INTTYPE[15:0]** — Interrupt Type. These bits control whether the corresponding interrupt source requests a normal interrupt or a fast interrupt.<br>• A reset operation clears these bits.<br>• The corresponding **INTENABLE** bit must be asserted to enable the interrupt request<br>• Only Bits 10 - 0 map to a valid interrupt source | 0 = Interrupt source generates a normal interrupt (nIRQ)<br>1 = Interrupt source generates a fast interrupt (nFIQ) |

## 8.6.7 Normal Interrupt Vector (NIVECTOR)

The Normal Interrupt Vector Register (NIVECTOR) provides the highest pending normal interrupt number. This number can be directly used as an index into a vector table to select the highest pending normal interrupt source. This register value is derived directly from the normal interrupt pending register (NIPEND).

| NIVECTOR | | | | Normal Interrupt Vector | | | | | | | | | Addr $8002_0028 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | | | | | | | | | | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | | | | | | | | | | | | | NIVECTOR[3:0] | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

**Table 8-9. NIVECTOR Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| Bits 31-4 | Reserved | |
| Bits 3-0 | **NIVECTOR[3:0]** — Normal Interrupt Vector. Indicates number for the highest pending normal interrupt. | 0 - 10 = Highest pending normal interrupt<br>11 - 15 = Unused (invalid) |

## 8.6.8 Fast Interrupt Vector (FIVECTOR)

The Fast Interrupt Vector Register (FIVECTOR) provides the highest pending fast interrupt vector number. This number can be directly used as an index into a vector table to select the highest pending fast interrupt source. This register value is derived directly from the fast interrupt pending register (FIPEND).

| FIVECTOR | | | | Fast Interrupt Vector | | | | | | | | | Addr $8002_002C | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | | | | | | | | | | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | | | | | | | | | | | | | FIVECTOR[3:0] | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

**Table 8-10. FIVECTOR Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| Bits 31-4 | Reserved | |
| FIVECTOR Bits 31-0 | FIVECTOR[3:0] — Fast Interrupt Vector. Indicates number for the highest pending fast interrupt. | 0 - 10 = Highest pending fast interrupt<br>11 - 15 = Unused (invalid) |

## 8.6.9 Interrupt Source Register (INTSRC)

The Interrupt Source Register (INTSRC) reflects the status of all interrupt request inputs into the interrupt controller. Unused bit positions always read zero (no request pending). The state of this register out of reset is determined by the peripheral circuits generating the requests; normally, the requests would be inactive.

| INTSRC | Interrupt Source Register | | | | | | | | | | | | Addr $8002_0030 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | | | | | | | | | | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | INTSRC[15:0] | | | | | | | | | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 8-11. INTSRC Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| Bits 31-16 | Reserved | |
| Bits 15-0 | **INTSRC[15:0]** — Interrupt Source. Indicates the state of the corresponding hardware interrupt source.<br>• Only Bits 10 - 0 map to a valid interrupt source<br>• Bits 15 - 11 are not used | 0 = Interrupt source unasserted<br>1 = Interrupt source asserted |

## 8.6.10 Interrupt Force Register (INTFRC)

The Interrupt Force Register (INTFRC) allows for software generation of interrupts for each of the possible interrupt sources for functional or debug purposes.

This register should be updated with 32-bit stores only.

| INTFRC | Interrupt Force Register | | | | | | | | | | | | Addr $8002_0034 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | | | | | | | | | | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | INTFRC[15:0] | | | | | | | | | | | | | | | |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 8-12. INTFRC Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| Bits 31-16 | Reserved | |
| Bits 15-0 | INTFR[15:0] — Interrupt Source Force Request. Used to force a request for the corresponding interrupt source.<br>• Only Bits 10 - 0 map to a valid interrupt source<br>• Bits 15 - 11 are not used | 0 = Standard interrupt operation<br>1 = Interrupt force asserted |

## 8.6.11 Normal Interrupt Pending Register (NIPEND)

The Normal Interrupt Pending Register (NIPEND) indicates all pending normal (IRQ) interrupt requests. This register status is determined by the Interrupt Enable Register (INTENABLE), Interrupt Type Register (INTTYPE), and the Interrupt Source Register (INTSRC).

| NIPEND | Normal Interrupt Pending Register | | | | | | | | | | | | Addr $8002_0038 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | | | | | | | | | | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | NIPEND[15:0] | | | | | | | | | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 8-13. NIPEND Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| Bits 31-16 | Reserved | |
| Bits 15-0 | **NIPEND[15:0]** — Normal Interrupt Pending Field. Indicates all pending normal interrupt requests to the CPU core. For an active pending normal request:<br>• A corresponding normal interrupt enable t must be set<br>• A corresponding IRQ must be active<br>• Only Bits 10 - 0 map to a valid interrupt source<br>• Bits 15 - 11 are not used | 0 = No normal interrupt request<br>1 = Normal interrupt request pending |

## 8.6.12 Fast Interrupt Pending Register (FIPEND)

The Fast Interrupt Pending Register (FIPEND) indicates all pending fast (FIQ) interrupt requests. This register status is determined by the Interrupt Enable Register (INTENABLE), Interrupt Type Register (INTTYPE), and the Interrupt Source Register (INTSRC).

| FIPEND | Fast Interrupt Pending Register | | | | | | | | | | | | Addr $8002_003C | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | | | | | | | | | | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | FIPEND[15:0] | | | | | | | | | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 8-14. FIPEND Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| Bits 31-16 | Reserved | |
| FIPEND<br>Bits 15-0 | FIPEND[15:0] — Fast Interrupt Pending Field. Indicates all pending fast interrupt requests to the CPU core. For an active pending fast request:<br>• A corresponding fast interrupt enable must be set<br>• A corresponding IRQ must be active<br>• Only Bits 10 - 0 map to a valid interrupt source<br>• Bits 15 - 11 are not used | 0 = No fast interrupt request<br>1 = Fast interrupt request pending |

# Chapter 9
# MAC Accelerator (MACA)

## 9.1 Overview

This chapter describes the low-level MAC and PHY link controller, i.e., MAC Accelerator (MACA), which together with software running on the ARM CPU, implements the baseband protocols and other low-level link routines for media access and link control.

The MACA provides the following:

- The interface between the network layer and the RF modem
- A reduction in MCU load
- Contains embedded features that control parts of the IEEE 802.15.4 PHY and MAC layers
- Provides the protocol and control mechanisms required for channel access
- Builds the transmit packets
- Parses the received packets
- Handles acknowledgements and TxPoll sequences independent of the ARM processor

**NOTE**

The MACA is not used as a standalone function. It is an integral part of the complete IEEE 802.15.4 Standard MAC/PHY services provided on the MC1322x. This chapter provides a reference to understand basic operation of the MACA, but is not intended as an applications guide to implement the 802.15.4 MAC function. Freescale provides the complete MAC solution on board and intends the user to access the functionality through the provided software tools.

**NOTE**

The MC1322x device ID register is located within the MACA register map at Address 0x8000_4018.

## 9.2 Features

The following is a partial list of 802.15.4 Standard network features:

- Over-the-air data rates of 250 Kbps
- Star or peer-to-peer topology
- Allocated 16-bit short or 64-bit extended addresses
- Allocation of Guaranteed Time Slots (GTSs)

- Carrier Sense Multiple Access with Collision Avoidance (CSMA-CA) channel access (slotted and non-slotted modes)
- Fully acknowledged protocol for transfer reliability
- Energy detection (ED)
- Link quality indication (LQI) for received packets
- 16 channels in the 2.450 GHz ISM band

The following is a list of MACA core features:

- Sequence manager for auto sequences:
  - Automatic acknowledgment frame reception on transmitted packets
  - Automatic acknowledgment frame transmission on received packets
  - Auto-Rx for continuous reception as coordinator
  - Auto sequence for transmitted MAC data.request
  - Assist for efficient response to MAC data.requests
  - Embedded channel assessment in sequence
  - Support for sequences with slotted CSMA-CA mode access
  - Timer-triggered or immediately executed actions
    - Actions can be triggered on either native or relative clock
    - Relative clock is based on native (free-running) clock with an added offset
  - Support for extended Rx for reception in random backoff and battery life extension
  - Support for promiscuous mode
- Dedicated DMA for hardware Tx/Rx data transfer between main RAM and MACA
- Packet manager
  - Handles preamble data
  - Handles frame check sequence (CRC)
  - Embedded header filter for received packets
- Beacon Support Mode
- Controlled by CPU accessible registers

## 9.3 Primary Functionality

To reduce CPU loading, the MACA constructs transmit packets, parses the received packets, and handles acknowledgements and TxPoll sequences independent of the ARM processor.

For packet transmit, the MACA constructs the entire packet which includes the preamble, Start of Frame Delimiter (SFD) and the Frame Check Sequence (FCS). During receive, the modem recognizes the preamble and SFD, provides a pulse to mark the first bit of frame length, receives the entire packet starting with the first bit of frame length and checks the FCS. Figure 9-1 shows a diagram of an 802.15.4 packet.

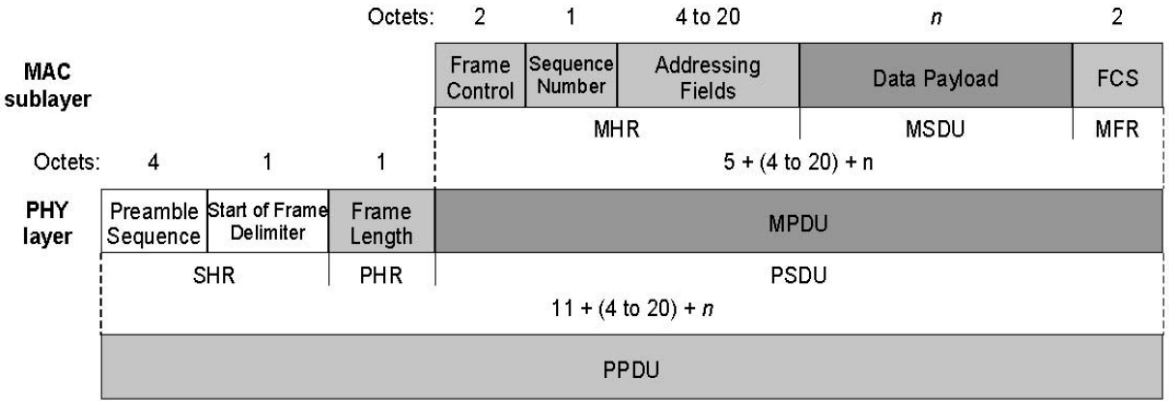



**Figure 9-1. 802.15.4 Packet Diagram**

The MACA supports slotted and non-slotted CSMA-CA modes. Slotted and non-slotted CSMA-CA use a basic time unit called a Backoff Period (BP). A single BP unit is equal to 20 Symbols (a symbol is 4 bits of an octet, or 2 symbols equal one octet). A Symbol time is 16 µsec.

Slotted CSMA-CA For use in beacon-enabled networks, slotted CSMA-CA channel access divides time into slots. The slots are aligned with the start of beacon transmission. Each time a device needs to transmit data frames during the contention access period (CAP), it locates the boundary of the next slot and then waits for a random number of slots. If the channel is busy (following the BP) the device waits for another random number of slots before trying to access the channel again. If the channel is idle, the device begins transmission on the next available slot boundary.

Non-slotted CSMA-CA For use in non-beaconed networks, non-slotted CSMA-CA is random. Every time a device needs to transmit data frames or MAC commands, it waits for a random period of time. If the channel is idle (following the BP) it transmits the data. If the channel is busy (following the BP) the device waits for another random time period before accessing the channel again. Acknowledgment frames are sent without using any CSMA-CA method.

For more information on slotted and non-slotted CSMA-CA modes, see the IEEE 802.15.4 Standard.

## 9.4 MACA Block Diagram

Figure 9-2 shows a functional overview of the MACA and its interfaces.

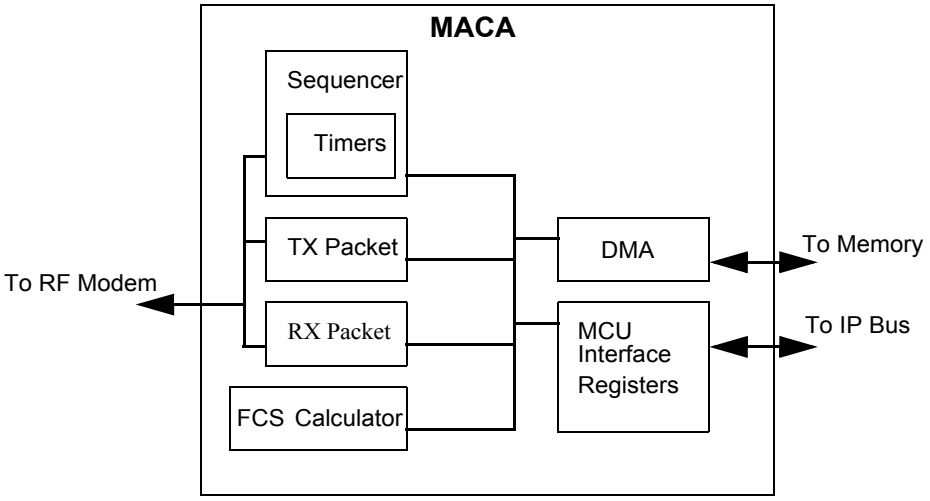



**Figure 9-2. MACA Simplified Block Diagram**

The following list provides a brief description of each MACA module and its functionality:

- Sequencer — The internal state machines are responsible for controlling user requested auto-sequences. See Section 9.5.1, “Sequencer” for more information about sequences. See subsections for information about the following sequencer details:
  - — Timer-triggered start of actions (Section 9.5.1.1, “Timer-Triggered Start Time”)
  - — Transmission sequence, Section 9.5.1.2, “Transmission Sequence”, as well as the following items:
    - – Automatic acknowledgment frame reception
    - – Poll sequence handling
  - — Receive sequence, Section 9.5.1.3, “Receiving Sequence”, as well as the following items:
    - – Receive time-out
    - – Extended Rx
    - – Transmission while receiver active
    - – Handling received MAC data request (POLL)
    - – Generation of acknowledgement frame
  - — Clear Channel Assessment (CCA)
  - — Energy Detection (ED)
  - — Clock generation
  - — Interrupt generation
  - — Packet generation
  - — Packet receiver
  - — FCS handler

- The Dedicated Direct Memory Access (DDMA) transfers data between memory and the MACA buffers (Section 9.5.2, “Dedicated Direct Memory Access (DDMA)”)
- The Random generator is a 32-bit random number generator which is also usable for random number generation in security applications (Section 9.5.3, “Random Generator”)
- The Radio/modem control is an interface to the modem (Section 9.5.4, “Radio Modem Control”)
- Beacon mode support (Section 9.5.5, “Beacon Mode Support”)

## 9.5 Module descriptions

The following sections provide more details about the MACA core features and their functionality. All modules are controlled through memory mapped registers. See Section 9.6, “MACA Register Memory Map” for detailed register descriptions.

### 9.5.1 Sequencer

The sequencer handles auto-sequences which are a set of actions executed with respect to timing requirements defined by the ARM specification. Individual independent actions are:

- Clear Channel Assessment (CCA)
- Transmit
- Receive
- TxAck
- RxAck
- TxPoll

By combining these actions into longer autonomous auto-sequences, the MCU receives fewer interrupts, and the sequence internal timing requirements are managed by the hardware. Sequences are controlled by the MACA_CONTROL register.

**NOTE**

Writing to the control register should only be done when a sequence is fully completed because a new action terminates any currently running sequence. Aborting a sequence generates an ActionComplete_IRQ with an “aborted” status.

When a sequence completes (finished, aborted, or failed) it generates an ActionComplete_IRQ. Additional information about why the sequence was completed is located in the MACA_STATUS register COMPLETE_CODE field which contains the most recent status code.

The sequencer provides the following auto-sequences and actions:

- TxSequence — [CCA]-[CCA]-Tx-[RxAck]. See Section 9.5.1.2, “Transmission Sequence”
- TxPollSequence — [CCA]-[CCA]-Tx(MAC-Data.request)-RxAck-[RxData][1]-[TxAck]. See Section 9.5.1.2.2, “Poll Sequence”
- RxSequence — Rx-[TxAck]-[Rx]. See Section 9.5.1.3, “Receiving Sequence”

1. Receiver is only started if the received ACK indicates data pending.

- RxSequence(MAC-Data.request) — Rx-TxAck-TxData-[Rx]. This is an alternative sequence if the MAC data request is received, see Section 9.5.1.3.5, “Handling Received MAC Data Request”
- Wait — This is for no operation, but it generates an interrupt on time-out. See Section 9.5.1.5, “Wait (Timer Controlled NOP)”.

### 9.5.1.1 Timer-Triggered Start Time

All sequences can be started immediately or they can be timer-triggered. If an action is timer-triggered, the start clock must be written before the control register is written. Timer-triggered actions are a one-shot action only. That is, once the action is started (or has been aborted), the timer comparator circuit is disabled. When the action starts, an ActionStarted_IRQ is generated.

#### 9.5.1.1.1 Late Start

In the event that a starting time occurs in the past, this late start must be detected by calculating the time to action start. Because the timers are 32 bits in length, they exceed the 802.15.4 Standard requirements of at least 24-bit timers running at symbol clock (62.5 kHz). This allows detection for actions occurring in the past. If time to action is more than 31 bits (negative), then it is assumed to have occurred in the past, and the action is terminated immediately. In this case, an ActionComplete_IRQ is generated, and a “Late Start” reason code is placed in the status register. See Section 9.7.4, “Status Register (MACA_STATUS)” for further details about complete codes, and Section 9.5.1.8, “MACA Interrupt Requests” for more information about interrupts.

Detection is performed immediately upon writing to the start clock register (MACA_STARTCLK). This detection mechanism is also applied to MACA_CPLTIM and MACA_SFTTIM registers.

A special instance of “Late Start” is when a timer-triggered action is started by writing to MACA_CONTROL register but no active timer is running. In this case, the action is aborted immediately with the generation of a ActionComplete_IRQ, with the “Late Start” status.

### 9.5.1.2 Transmission Sequence

When a transmission sequence starts, it can be initiated with no CCA or slotted/non-slotted CSMA-CA CCA. These are controlled by the MACA_CONTROL register. If CCA is required, the sequence terminates if the channel is detected as busy. A termination is sent to the software by generating an “ActionComplete_IRQ” interrupt, and it sets the result code to “Channel Busy” in the MACA_STATUS register.

The timing between CCA sequences and transmission start is fixed for both slotted CSMA-CA and non-slotted CSMA-CA CCA mode.

For transmissions in slotted CSMA-CA mode, the CCA must begin on an offset boundary. An offset boundary is defined as the beginning of a beacon, and then every 20 symbols. Transmit actions are started using a timer trigger, and the software is responsible for correctly aligning the sequence.

If the transmission sequence succeeds, it completes by generating an “ActionComplete_IRQ” interrupt, and the “Success” status. If a requested acknowledgment frame is not returned, the sequence completes

with the “No Ack” status. Retransmission and backoff time calculation are the responsibility of the software.

Data is transferred from memory to the packet generator using DDMA, see Section 9.5.2, “Dedicated Direct Memory Access (DDMA)”.

The internal timing for aligning CCA back-to-back is controlled by the CCADELAY bit, and the delay between CCA and transmit (in both beacon and non-beacon mode) is controlled by the TXDELAY bit. Both bit settings are located in the MACA_CCADELAY register. The delay between the transmitted packet and the opening of the receiver for acknowledgement is controlled by the MACA_RXACKDELAY register. This register also contains the receiver window length.

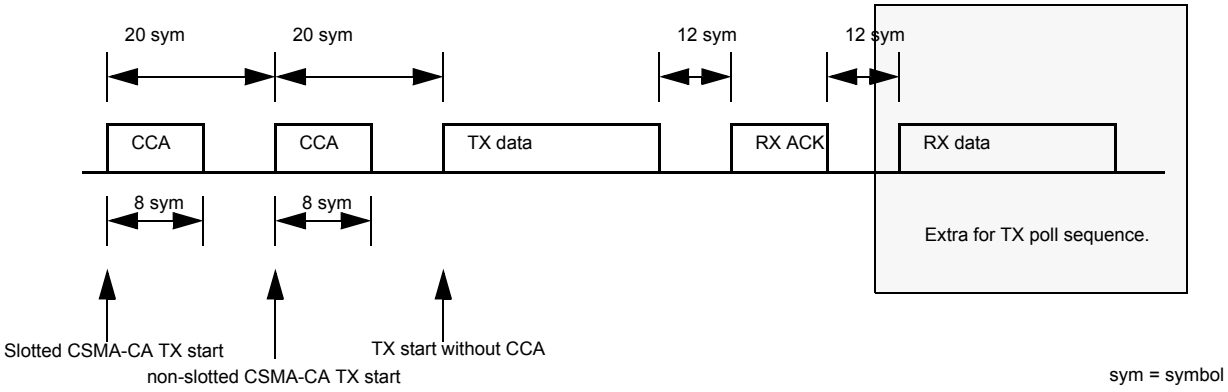



**Figure 9-3. Transmit Transmission Timeline**

#### 9.5.1.2.1 Automatic Acknowledgement Reception

Upon completion of data transmission, the sequencer may be instructed to receive an acknowledgement frame. An acknowledgment frame can be expected to arrive 11 symbols from the last bit transmitted.

**NOTE**

The turnaround time is approximately 12 symbols so an accuracy of +/- one symbol is implied.

The acknowledgment frame includes a sequence number which is matched against the sequence number in the transmitted packet. If the sequence number does not match, then the acknowledgment frame is discarded. The packet receiver is responsible for extracting the sequence number. The transmitted sequence number is mirrored in the TXSEQNR register.

The data received in the acknowledgement frame should not be transferred to memory using DDMA but only used internally in the sequencer.

**NOTE**

The sequence number and acknowledgement request is embedded in the TxData, but instead of having hardware parse this information, it is supplied through the MACA_TXLEN and MACA_TXSEQNR registers.

#### 9.5.1.2.2 Poll Sequence

An extension of the transmission sequence supports autonomous handling of the MAC Data.request, which is used for the device to retrieve data from the coordinator as for example, when polling the coordinator. This sequence is selected by setting the “POLL” bit in the MACA_CONTROL register.

**NOTE**

This sequence must be set to transmit.

In the acknowledgement frame that is received as the response to the Data.request, the “Frame pending” bit inside the frame control field indicates that the coordinator has data available and that the device should open its receiver. If the “frame pending” bit is set, the sequencer prepares for data reception within 11 symbols from receiving the acknowledgment frame. The sequencer opens the receiver for data reception. In this case, a receive time-out must be specified and it is the responsibility of the software to calculate the time-out. If data is received, a DataIndication_IRQ interrupt is issued, the complete clock is cancelled, and the poll sequence is complete by generating an ActionComplete_IRQ and setting the status to “success”.

If no data is received before the time-out, the sequence completes by generating an ActionComplete_IRQ and setting the status to “time-out”.

#### 9.5.1.2.3 Random Backoff Before Transmission

Prior to transmissions using CCA, a random backoff is performed. It is the responsibility of the software to calculate and implement this period by setting up a timer-triggered action.

### 9.5.1.3 Receiving Sequence

Receive operations may be initiated using timer-triggered start or they can be started immediately after writing to the control register. Once the receiver is started, the sequencer waits for sync-detection from the RF modem and then starts to receive data into memory using the DDMA. The first byte transferred starts with the frame control. The last byte of data transferred is the last byte of the payload. The frame length can be read from the GET_RX_LVL register.

The receiver only transfers data to memory once for every time a pointer is written to the DDMA pointer register. Writing a new pointer implies buffer availability. When a packet is received, the buffer is used and a new buffer must be provided (through a new write to the pointer register). If a sync is detected, but no buffer is available, the reminder of the packet is handled as a failed packet. See Section 9.7.18, “DMA Rx Data Pointer Register (MACA_DMARX)” for more information about the MACA_DMARX pointer register.

Once data is fully received and the FCS is valid, the sequencer evaluates the frame control field. If the acknowledgement request bit is set, the auto-sequence continues by transmitting an acknowledgement frame.

Whenever data is fully received (and acknowledged if required) a DataIndication_IRQ interrupt is issued. If the sequencer finishes at this point, an ActionComplete_IRQ is generated with a “success” status. A data indication interrupt is guaranteed to occur before (or at the same time as) a complete interrupt.

Upon completion of the receive sequence, the receiver is restarted immediately if the “AUTO” bit is set in the control word. Otherwise, the sequence completes with a “Success” status (the packet was successfully

received), and any time outs are aborted. This means that if the filter or checksum fails, the receive sequence is always restarted regardless of the state of “AUTO”.

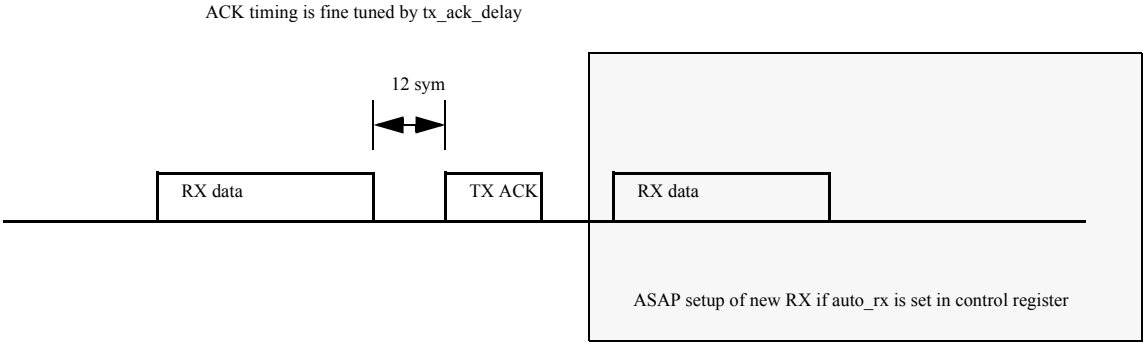



**Figure 9-4. Receive Operation Timeline with Optional Receiver Restart**

#### 9.5.1.3.1 Receive Time-out

When the receive time-out occurs, the receive sequence is terminated immediately, even though a sync was found and a reception was in progress. In this case, the sequence is completed, and an interrupt ActionComplete_IRQ is generated. The complete status is set to “time-out”. Time-out is controlled by the MACA_CPLTIM register.

#### 9.5.1.3.2 Soft Time-out

An alternative to the hard time-out provided by the MACA_CPLTIM register, is a less harsh receive time-out referred to as a soft time-out. The soft time-out does not terminate a receive sequence if a packet is being received (for example, when an SFD is found and end of packet not reached). Instead, the time-out is postponed and generated at the end of the packet after filter rejects, checksum fails, or packet successfully received. The radio is also returned to Idle mode and an ActionComplete_IRQ is generated, with the “SftPndTimeout” status. If the time-out triggers and is not being postponed, the action completes with ActionComplete_IRQ and the “SftTimeout” status.

The soft-time-out is programmed using the MACA_SFTTIM register.

#### 9.5.1.3.3 Battery Life Extension

Battery life extension is implemented using a soft-time-out, and with the “AUTO” bit disabled in MACA_CONTROL register. This allows the receiver to be open for a given time bounded by MACA_SFTTIM and MACA_CPLTIM register settings. This setting can provide the following scenarios:

- No data received — ActionComplete_IRQ generated at the soft-time-out, with “SftTimeout” status. Any complete timer is aborted.
- Successful data before soft time-out — (DataIndication_IRQ), ActionComplete_IRQ with “success” status, because the receiver is not restarted.
  Any complete timer is aborted.
- Successful data while soft time-out — (DataIndication_IRQ if data Ok), ActionComplete_IRQ with “SftPndTimeout” status.
  Any complete timer is aborted.

- If the “hard” time-out (given by the MACA_CPLTIM register) is reached in any scenario, the receive is aborted with the ActionComplete_IRQ with “time-out” status.
  Any complete timer is aborted.

#### 9.5.1.3.4 Transmission While Receiver Active

As a coordinator, the receiver is usually enabled all the time to allow devices to transmit at any time. But if the coordinator wants to initiate a direct data transfer, a random backoff must be calculated, and the receiver temporarily stopped. In order to gently stop the receiver at a calculated time, use the soft time-out process. During the random backoff period, the receiver is intentionally allowed to run until the backoff period ends. However, do not force the receiver off if a reception is in progress.

For this functionality to work, users can program and use the MACA_SFTTIM register while the receiver is active. This sets a “soft” time-out which behaves almost like battery life extension. However, instead of completing the action on the very first successfully received packet, the receiver is restarted. (If “AUTO” is set in the MACA_CONTROL register).

This means that this sequence does not complete until either a (pended) soft time-out or a hard time-out occurs. The following scenarios describe this behavior.

1. No data received — An ActionComplete_IRQ is generated on the “soft” time-out (“SftTimeout” status).
   Any complete timer is aborted.
2. Successful data before soft time-out — (DataIndication_IRQ interrupts occur every time a successful packet is received). An ActionComplete_IRQ is generated on the “soft” time-out (“SftTimeout” status).
   Any complete timer is aborted.
3. Successful data while soft time-out — An ActionComplete_IRQ is generated on end of packet (“SftPndTimeout” status) regardless of packet success (filter or checksum passed).
   Any complete timer is aborted.
4. If the “hard” time-out (provides by the MACA_CPLTIM register) is reached in any scenario, the receiver is aborted, and an ActionComplete_IRQ is generated with the “time-out” status.
   Any complete timer is aborted.

#### 9.5.1.3.5 Handling Received MAC Data Request

A receive sequence special case is when a MAC data request is received. In this special case, the remote side expects a data response. There are two ways of responding with data:

- Responding immediately
- With random backoff and CCA before transmission

As shown in Figure 9-5, there are two methods of transmitting a data response:

| | |
|---|---|
| Upper | Data may be transmitted immediately (12 symbols) after acknowledgement, and no CCA is needed. |
| Lower | Data may be transmitted later, but a CCA is required. In slotted CSMA-CA mode, the transmissions must be aligned to backoff boundaries. |

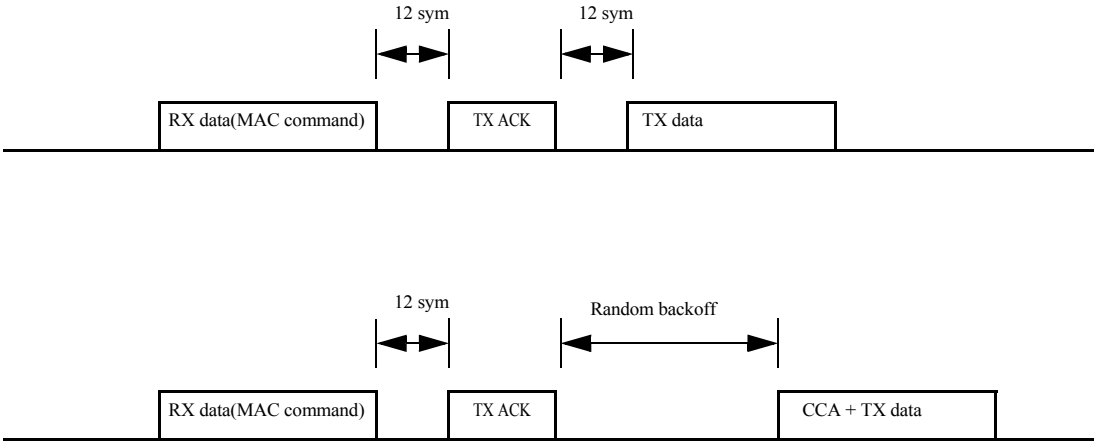
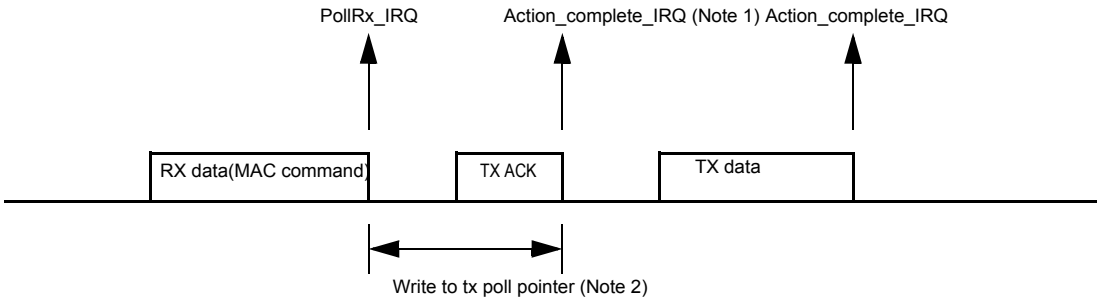



**Figure 9-5. Data Response Methods**

A faster response results in lower power consumption because the receiver (usually a battery powered device) would otherwise have to open its receiver for longer time periods).

In unslotted mode, a fast response must start transmission 12 symbols after the acknowledgment frame. This is the slotted CSMA-CA mode required transmission time to be aligned with the backoff boundary. See Section 9.5.5.1, "Slotted CSMA-CA Mode Timing". This leaves very little time left for the software to determine what possible data to transfer and to setup the transmission. To assist accomplishing a fast response, a "PollRx_IRQ" interrupt is generated on the end of the MAC request (CRC valid, and acknowledgment frame about to be transmitted), see Figure 9-5. When the software sees this interrupt, the data response can be determined (assuming that there is data in indirect queue to the polling device). By writing the data pointer to the MACA_DMAPOLL register, the DDMA is armed for transmission data, and writing to this register also informs the sequence to continue with fast data response.




Note 1: Action_complte_IRQ will occur if maca_frmpend==0, or maca_frmpend updated after usage, or tx_poll pointer not updated before tx done.

Note 2: If tx poll pointer is written, Tx sequence will be started. Else, complete with "success" status.

**Figure 9-6. Generating a Fast Response**

If the MACA_DMAPOLL register is not written before the completion of the acknowledgment frame which indicates data pending, the sequence is assumed to be a slow response, and does not start-up the transmitter. The auto-sequence will then complete or restart the receiver (if AUTO enabled). In any case, a DataIndication_IRQ is generated. If the sequence completes, an ActionComplete_IRQ is generated.

**NOTE**

Ensure that the ActionComplete_IRQ does not occur before the DataIndication_IRQ.

If the MACA_DMAPOLL register is written after completion of the acknowledgment frame, the sequencer must immediately abort its sequence and issue a complete interrupt with the “Late start” status. For example, writing to the MACA_DMAPOLL register should only be done in the window between PollRx_IRQ generation and the completion of the following acknowledgement.

In slotted CSMA-CA mode, the acknowledgment frame must be transmitted on a backoff boundary (see Section 9.5.5.1, “Slotted CSMA-CA Mode Timing”). This extends the time available for setting up a fast response, but the procedure is the same as far as software is concerned.

It is possible to update the MACA_FRMPND register up until the time when the status is used to generate the acknowledgement response information. If the MACA_FRMPND register is zero (regardless if the MACA_DMAPOLL register is set up), the sequence terminates after the acknowledgement is transmitted.

If the MACA_FRMPND register is updated after the information is used in the acknowledgement, but before the completion of the transmitted acknowledgement, the MACA_FRMPND register is not updated, and the sequence completes after acknowledgement transmission with the ActionComplete_IRQ, and the “Late Start” status. Software can then read the MACA_FRMPND register to know the pending-frame contents of the transmitted acknowledgement.

If the packet is protected by a security integrity check, the software is responsible for validation.

#### 9.5.1.3.6 Generating the Acknowledgement Frame

Acknowledgement frames are generated internally and contain 3 bytes of data (excluding FCS). The data is a frame control field and a sequence number. The sequence number is copied from the just received data packet. When building the frame control field, the “frame pending” bit must correspond to the state of the MACA_FRMPND register.

In unslotted mode, acknowledgement frames are transmitted 12 symbols after reception of data frames that require acknowledgement, but in slotted CSMA-CA mode, the frame must be further delayed until a backoff boundary. See Section 9.5.5.1, “Slotted CSMA-CA Mode Timing” for further information about slotted CSMA-CA alignment.

The handling of minimum turnaround may be fine-tuned by using of the MACA_TXACKDELAY register.

### 9.5.1.4 Clear Channel Assessment (CCA)

CCA is handled by the RF modem. The action is setup by the MACA sequencer, and when the assessment completes, output from the RF modem is used to determine how the MACA auto sequence is continued.

A CCA has an outcome of either busy or clear. If the channel is busy, the “BUSY” bit in MACA_STATUS register is set. The follow-up section is controlled by the transmission sequence.

A CCA is always be used in conjunction with a transmission sequence.

#### 9.5.1.5 Wait (Timer Controlled NOP)

By programming the time-out register and afterwards writing a “NOP” action command to the control register, the MACA module will be idle until the time-out triggers. At this point, a CompleteAction_IRQ is generated with the “time-out” status.

#### 9.5.1.6 Header Filtering Block

Received data must be filtered according to the 802.15.4 Standard. The filter may be disabled by enabling promiscuous mode (The PRM bit in the MACA_CONTROL register). If the filter is disabled, auto-acknowledgement is not enabled.

A packet is received if it meets the following criteria:

- The frame type subfield of the frame control field can not contain an illegal frame type
- If the frame type indicates that the frame is a beacon frame, the source PAN identifier must match *macPANId* unless *macPANId* is equal to 0 x ffff. In this case, the beacon frame must be accepted, regardless of the source PAN identifier
- If a destination PAN identifier is included in the frame, it must match the *macPANId* or it must be the broadcast PAN identifier (0 x ffff)
- If a short destination address is included in the frame, it must match either the *macShortAddress* or the broadcast address (0 x ffff). Otherwise, if an extended destination address is included in the frame, it must match the *aExtendedAddress*
- If only source addressing fields are included in the data or MAC command frame, the frame must be accepted only if the device is a PAN coordinator and the source PAN identifier matches the *macPANId*

If the filter determines that the currently received data is not destined for this device, the reception is stopped, and the receiver is restarted, otherwise the packet is received until its end.

The macPANId is extracted from MACA_MACPANID register and the macShortAddress is found in the MACA_MAC16ADDR register. The extended address is found by combining the MACA_MAC64HI and MACA_MAC64LO registers.

To determine the current mode of operation, the control register bit (ROLE) is set when the device is a PAN coordinator.

The usage of the short or extended address is based on the received frame control field. The packet address information is extracted in the packet receiver. See Section 9.5.1.10, “Packet Receiver”.

An additional check is applied to the packet length as a sanity check:

- Length < 2 — Reject
- Length < 5 — Reject if not promiscuous mode

If the receiver filter fails for any reason, or the CRC fails, then the receiver is restarted.

#### 9.5.1.6.1 Buffer Starvation Prevention

An extension to the header filter prevents buffer starvation which occurs on routers where incoming data is received (and acknowledged) using the last available buffer. This means that any further incoming packets are discarded. In the worst case, the system can not empty the outgoing queue if that data needs to be POLLed out.

To avoid this scenario, the filter must reject all MAC command packets that are not Data.requests. (See Section 9.5.1.3.5, “Handling Received MAC Data Request”) for information about whether the POLL bit is set in the MACA_FLTREJ register.

#### 9.5.1.6.2 Explicit Packet Type Rejection

As an option, an extension to the header filter rejects specific packet types. Packet types are found in the the first 3 bits of the frame control field. Frame types identified by CMD, ACK, DATA, and BCN in the MACA_FLTREJ register must be rejected.

#### 9.5.1.6.3 Auto-Reception of Acknowledgement

If the sequencer is set up for automatic reception of the acknowledgment frame, then only these types of packets are allowed. Any other packet type is rejected. This is controlled internally in the MACA.

#### 9.5.1.6.4 Ensure Frame Control Field Reserved Bits are Zero

To ensure that reserved bits in the frame control field are zero, the FC_MASK field in MACA_FLTREJ register must be AND’ed with the received frame control field. If the outcome is non-zero, it indicates that some reserved bit(s) are set, and the packet is rejected. This approach allows explicit rejection/acceptance of 802.15.4b packets.

#### 9.5.1.6.5 Detection of Data Request

A special extension of the filter is the detection of received data request packets. This requires the receive filter to check the payload data of MAC command frames and check for a match against the command frame identifier equal to 0x04. Integrity (security) bits in the packet are skipped and validated in software.

### 9.5.1.7 Clock Generation

The raw clock source to the MACA module is the Peripheral Clock established by the prescaler in the CRM Control Register. This source must be further divided in the MACA_CLKDIV Register (see Section 9.7.34, “Clock Divider Register ( MACA_CLKDIV)”) to establish a MACA module clock of 250 KHz.

- The divide ratio N = Divide[15:0] + 1
- Set the bit rate to 250 KHz to match the 802.15.4 250 kb/s rate.
- Typical register value is 95 (dec) or N = 96 for Peripheral Clock of 24 MHz

All timers are 32-bit wide, and run at 250kHz. Triggers are armed by writing to the appropriate register (MACA_CPLCLK). Triggers are one-shot only and need to be re-armed once expired. Also, if timers are aborted, a re-arm is also needed.

In order to detect timer triggers occurring “in the past”, the MACA checks the difference between the current clock and the time until a trigger should occur. If the difference is larger than 31 bits (a negative result that corresponds to more than 2 hours into the future) it is assumed to be an error (or as already occurred) and an ActionComplete_IRQ is generated with the “Late start” status. After this, the sequencer is in the idle state.

If the complete clock is written to while armed and the sequencer is not idle, a new clock is set. For the remainder of the new time-out, the receiver operates in extended mode until completion of the sequence. A timer can be aborted by writing to the MACA_TMRDIS register.

#### 9.5.1.8 MACA Interrupt Requests

The MACA has a single interrupt request signal that is connected to the interrupt controller. MACA interrupt requests are event generated, and the multiple interrupt sources are all OR’ed into the one interrupt request source. The interrupt requests are controlled by four registers:

- Interrupt Status (MACA_IRQ) - reports the status of the IRQ sources
- Interrupt Clear (MACA_CLRIRQ) - writing a 1 to the proper bit will clear a status.
- Interrupt Set (MACA_SETIRQ) - writing a 1 will set a status (used for test and debug of IRQs)
- Interrupt Mask (MACA_MASKIRQ) - used to enable/disable individual IRqs

##### 9.5.1.8.1 Interrupt Sources

The following lists shows the individual interrupt sources inside the MACA:

- ActionComplete_IRQ (ACPL)
- PollRx_IRQ (POLL)
- DataIndication_IRQ (DI)
- Timeout_IRQ (CM)
- SoftTimeout_IRQ (SFT)
- SyncFound_IRQ (SYNC)
- ActionStarted_IRQ (STRT)
- CrcFail_IRQ (CRC)
- FilterFail_IRQ (FLT)
- RxLevel_IRQ (LVL)

Figure 9-7 shows some example scenarios that use interrupts throughout a packet transmission and reception.

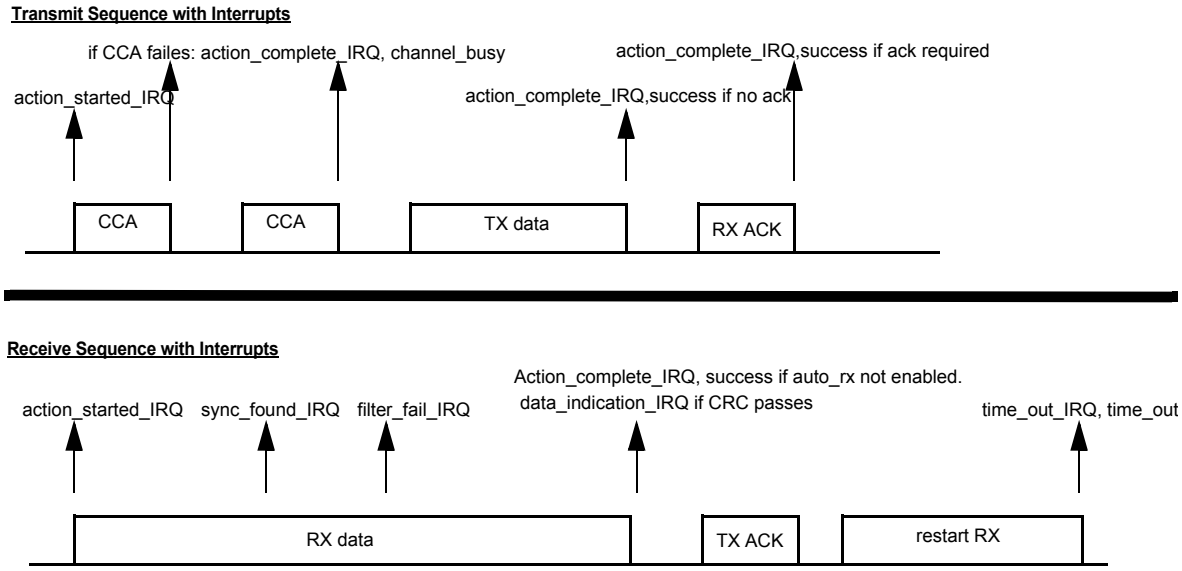



**Figure 9-7. Transmit and Receive Sequences**

#### 9.5.1.8.2 ActionComplete_IRQ

The ActionComplete_IRQ is generated when a sequence action is completed. Sequence actions are started by writing an action to the control register. The sequencer then performs the required actions and moves to idle mode after completion. When generating this interrupt, the MACA_STATUS register must also be updated to reflect the complete reason code.

If the start timer trigger is armed and a trigger occurs before an action is written to the control register, an ActionComplete_IRQ is generated instead of an ActionStarted_IRQ. Every time this interrupt is generated, the sequencer moves to idle mode.

To ensure that the RF modem is idle before a new action is issued, this interrupt may be delayed by writing the time in symbols to MACA_EOFDELAY register. Any time outs are aborted and the complete clock MACA_CPLTIM register is updated.

#### 9.5.1.8.3 PollRx_IRQ

The PollRx_IRQ is generated during an auto-sequence which has received a MAC data indication, and the FCS passes. See Section 9.5.1.3.5, “Handling Received MAC Data Request” for further description. The PollRx_IRQ is generated instead of a the DataIndication_IRQ.

#### 9.5.1.8.4 DataIndication_IRQ

When data, MAC, or beacons are received during receive mode, and FCS and header filtering passes, the DataIndication_IRQ interrupt is generated when the packet is received.

If a receive action finished at this point, it is required that the DataIndication_IRQ is generated before (or at the same time as) an ActionComplete_IRQ.

#### 9.5.1.8.5 Timeout_IRQ

The timeout_IRQ interrupt is set when a complete clock triggers.

#### 9.5.1.8.6 SoftTimeout_IRQ

The soft timeout_IRQ interrupt is set when a soft complete clock triggers.

#### 9.5.1.8.7 SyncFound_IRQ

This interrupt is generated during receive mode, when a packet is detected and frame delimiter and length are received. The interrupt is triggered on the end boundary of the length field. At this point, the clock is also latched into the MACA_TIMESTAMP register.

#### 9.5.1.8.8 ActionStarted_IRQ

This interrupt is generated when the start clock triggers and a pending action was written to the control register. If the clock triggers and no action is written to the control register (since setting up the start timer trigger), an ActionComplete_IRQ is generated instead.

#### 9.5.1.8.9 CrcFail_IRQ

This interrupt is generated when the checksum fails.

#### 9.5.1.8.10 FilterFail_IRQ

This interrupt is generated if the packet is rejected due to header filtering.

#### 9.5.1.8.11 RxLevel_IRQ

This interrupts is generated when the buffer FIFO level matches or exceeds the value programmed by the MACA_SETRXLVL register.

### 9.5.1.9 Packet Generation

The packet generation module generates the packet data which is passed to the RF modem. This includes internal generation of the synchronization header and length field, and control of the DDMA for reading transmit data from memory. It also inserts the frame control sequence (FCS) in the end of the packet. For details about calculating the FCS, see Section 9.5.1.11, “Frame Control Sequence (FCS) Handler”.

### 9.5.1.10 Packet Receiver

During receive mode, the packet receiver ensures that data is transferred into memory using the DDMA, and validates the frame control sequence. Also, the packet receiver interfaces to the header filtering block for validating packet destination, and extracts the header information from the packet, as well as the sequence number for acknowledgement reply/test.

#### 9.5.1.11 Frame Control Sequence (FCS) Handler

The FCS Handler is a 16-bit CRC. Input is provided by either the packet generator or the packet receiver. The intermediate result is kept inside this module, and is read by the packet generator or receiver to apply or check the FCS.

If the checksum fails, the receiver is restarted.

#### 9.5.1.12 Action finish clock

Whenever a complete interrupt (see Section 9.5.1.8.2, “ActionComplete_IRQ”) is generated, the clock will be latched at this point too. This shall be right at the end of a transmitted or received packet. the value is placed in the MACA_CPLTIM register.

### 9.5.2 Dedicated Direct Memory Access (DDMA)

This module transfers data from RAM to the MACA packet generator for transmission or from the MACA packet receiver to RAM during receiving. Although stealing minor bandwidth from the CPU, this off loads any need to provide support data transfer via the CPU.

- The data transfer occurs as required on a byte-by-byte basis for transmit or receive.
- The bytes are loaded in RAM such that the first byte received or transmitted is in the least significant address (little-endian) and the packet builds with increasing address.
- DMA Rx Data Pointer (MACA_DMARX) provides the pointer for the first byte of the received data packet in RAM.
- DMA Tx Data Pointer Register (MACA_DMATX) provides the pointer for the first byte of the data packet in RAM to be transmitted.
- Tx Length Register (MACA_TXLEN) provides the length of the Tx packet for transmission.
- Only the memory source/destination pointers are configurable.
- DDMA transfers are controlled by the MACA packet receiver module or the MACA packet generator module, and will stall any access to the bus by other peripherals or the ARM core (DMA has highest priority).

### 9.5.3 Random Generator

The 32 bit random number generator is used for backoff time calculation but can also be used for security purposes. The read returns a 32-bit random number while a write seeds the LFSR with an initial value. The LFSR is free running from the bus clock (13Mhz to 26Mhz). There are three ways the initial value can be set.

1. A hard system reset. (sets start value to 32’h1)
2. A soft reset from the register map. (sets start value to 32’h1)
3. A write to the register to “seed” the start value with the value written to the register.

The random generator has the following features.

- The LFSR used is a 32 bit primitive polynomial. It will repeat in $2^{32}$ clocks

- If somehow a group of “nodes” could be powered up and released from reset at precisely the same time, set with the same seed value and then, assuming that the frequency of the clock crystal of each node are offset by only 1ppm, the random number generator will be un-correlated in 0.078 seconds of operation
- The LFSR is running from the same clock as the ARM7. Even back-to-back reads return different values

## 9.5.4 Radio Modem Control

Interface to the RF modem is a high-level generation of signal starting actions as follows:

- Receive
  - Including fast restart of the receiver without a warm up or warm down period
- Transmit
- CCA
- ED
- Force Idle

Internal warm up timing in the sequences are not maintained by the accelerator module. Also, the interface must allow the software to change the frequency of operation. It is possible to place the RF modem into test-mode for evaluation. This requires the following modes:

- Continuous receive
- Continuous transmit with modulation
- Continuous transmit without modulation

The system must be able to enter these states from software. The RF modem returns a signal to indicate the current state (Rx, Tx, CCA, ED, Idle) to the MACA. Communication between RF modem and the MACA may be parallel or serial, depending on which implementation is most efficient.

### 9.5.4.1 Radio Test Modes

It is possible to enter continuous Rx, and Continuous Tx (with and without modulation). The control of these test modes is outside the scope of the MAC accelerator description.

## 9.5.5 Beacon Mode Support

Beacon handling is performed entirely in software and thus is completely responsible for timing in the system.

### 9.5.5.1 Slotted CSMA-CA Mode Timing

In slotted CSMA-CA mode, all transmissions are transmitted on a beacon offset boundary. The beacon defines the first boundary, and consecutive boundaries are placed at 20 symbol intervals.

Because a transmission is delayed by a warm-up period, it must be started at some point prior to the actual transmission of data. This is accomplished with the MACA_SLOTOFFSET register, which contains an

initial counter value. This value is loaded when a beacon is received or transmitted. See Section 9.7.16, "Slot Offset Adjustment Register (MACA_SLOTOFFSET)" for details about how to do this. This will align the counter to include any internal delays. The offset is used when the RSTO bit is set in the control register.

### 9.5.6 FIFO Buffer Interrupts

For mitigation purposes, it is possible to handle filtering in software. To accomplish this, a "stream" mode is supported.

It is also possible to set a receive level interrupt (after N bytes received) which generates a RxLevel_IRQ. The trigger is level triggered and is compared against the value provided by the MACA_SETRXLVL register. This interrupt allows the CPU to fetch the packet data from RAM before the packet has been completely received. In this way, software filtering can be accomplished in software as the packet is still being received.

The level may be set multiple times for an optimum software filtering of the header. In addition, the software can read the current number of received bytes using the MACA_GETRXLVL register.

## 9.6 MACA Register Memory Map

The MACA module is programmed through memory-mapped registers. Table 9-1 lists all registers in the MACA module and their relative address in memory.

**NOTE**

- The MACA base address is **0x8000_4000**
- Only 32-bit access is supported.

**Table 9-1. MACA Registers Memory Map**

| Address | Register name | Access Type | Access Width |
|---|---|---|---|
| Base + 0x00 | Reserved | | |
| Base + 0x04 | Reset (MACA_RESET) | R/W | 32-bit only |
| Base + 0x08 | Random number (MACA_RANDOM) | R/W | 32-bit only |
| Base + 0x0C | Control (MACA_CONTROL) | W | 32-bit only |
| Base + 0x10 | Status (MACA_STATUS) | R | 32-bit only |
| Base + 0x14 | Frame Pending (MACA_FRMPND) | R/W | 32-bit only |
| Base + 0x18 | MC1322X_ID (MACA_MC1322x_ID) | R | 32-bit only |
| Base + 0x1C | Reserved | | |
| Base + 0x40 | Enable timers (MACA_TMREN) | R/W | 32-bit only |
| Base + 0x44 | Disable timers (MACA_TMRDIS) | R/W | 32-bit only |
| Base + 0x48 | Clock (MACA_CLK) | R/W | 32-bit only |
| Base + 0x4C | Start Clock (MACA_STARTCLK) | R/W | 32-bit only |

**Table 9-1. MACA Registers Memory Map (continued)**

| Address | Register name | Access Type | Access Width |
|---|---|---|---|
| Base + 0x50 | Complete Clock (MACA_CPLCLK) | R/W | 32-bit only |
| Base + 0x54 | Soft Time-out Clock (MACA_SFTCLK) | R/W | 32-bit only |
| Base + 0x58 | Clock offset (MACA_CLKOFFSET) | R/W | 32-bit only |
| Base + 0x5C | Relative Clock (MACA_RELCLK) | R/W | 32-bit only |
| Base + 0x60 | Action Complete Timestamp (MACA_CPLTIM) | R | 32-bit only |
| Base + 0x64 | Tx Slot Offset Adjustment (MACA_SLOTOFFSET) | R/W | 32-bit only |
| Base + 0x68 | Receive Time Stamp (MACA_TIMESTAMP) | R | 32-bit only |
| Base + 0x80 | DMA Rx Data Pointer (MACA_DMARX) | R/W | 32-bit only |
| Base + 0x84 | DMA Tx Data Pointer (MACA_DMATX) | R/W | 32-bit only |
| Base + 0x88 | DMA Tx Poll Response Pointer (MACA_DMAPOLL) | R/W | 32-bit only |
| Base + 0x8C | Tx Length (MACA_TXLEN) | R/W | 32-bit only |
| Base + 0x90 | Tx Sequence Number (MACA_TXSEQNR) | R/W | 32-bit only |
| Base + 0x94 | Set Rx Level Interrupt (MACA_SETRXLVL) | R/W | 32-bit only |
| Base + 0x98 | Read Number of Received Bytes (MACA_GETRXLVL) | R | 32-bit only |
| Base + 0xC0 | Interrupt Status (MACA_IRQ) | R | 32-bit only |
| Base + 0xC4 | Interrupt Clear (MACA_CLRIRQ) | W | 32-bit only |
| Base + 0xC8 | Interrupt Set (MACA_SETIRQ) | W | 32-bit only |
| Base + 0xCC | Interrupt Mask (MACA_MASKIRQ) | R/W | 32-bit only |
| Base + 0x100 | MAC PAN ID (MACA_MACPANID) | R/W | 32-bit only |
| Base + 0x104 | MAC Short Address (MACA_MAC16ADDR) | R/W | 32-bit only |
| Base + 0x108 | High MAC Extended Address (MACA_MAC64HI) | R/W | 32-bit only |
| Base + 0x10C | Low MAC Extended Address (MACA_MAC64LO) | R/W | 32-bit only |
| Base + 0x110 | Filter Rejection Mask (MACA_FLTREJ) | R/W | 32-bit only |
| Base + 0x114 | Clock Divider (MACA_CLKDIV) | W | 32-bit only |
| Base + 0x118 | Warmup (MACA_WARMUP) | R/W | 32-bit only |
| Base + 0x11C | Preamble (MACA_PREAMBLE) | R/W | 32-bit only |
| Base + 0x120 | Reserved | | |
| Base + 0x124 | Frame Sync Word 0 (MACA_FRAMESYNC0) | R/W | 32-bit only |
| Base + 0x128 | Frame Sync Word 1 (MACA_FRAMESYNC1) | R/W | 32-bit only |
| Base + 0x140 | Tx Acknowledgement Delay (MACA_TXACKDELAY) | R/W | 32-bit only |
| Base + 0x144 | Rx Acknowledgement Delay (MACA_RXACKDELAY) | R/W | 32-bit only |
| Base + 0x148 | End of Frame Delay (MACA_EOFDELAY) | R/W | 32-bit only |
| Base + 0x14C | CCA Delay (MACA_CCADELAY) | R/W | 32-bit only |

**Table 9-1. MACA Registers Memory Map (continued)**

| Address | Register name | Access Type | Access Width |
|---|---|---|---|
| Base + 0x150 | Rx End (MACA_RXEND) | R/W | 32-bit only |
| Base + 0x154 | TX CCA Delay (MACA_TXCCADELAY) | R/W | 32-bit only |
| Base + 0x158 | MACA_KEY3 | R | 32-bit only |
| Base + 0x15C | MACA_KEY2 | R | 32-bit only |
| Base + 0x160 | MACA_KEY1 | R | 32-bit only |
| Base + 0x164 | MACA_KEY0 | R | 32-bit only |
| Base + 0x180 | MACA Options (MACA_OPTIONS) | W | 32-bit only |

## 9.7 MACA Register Description

The following sections provide detailed register descriptions.

### 9.7.1 Reset Register (MACA_RESET)

Writing to this register initiates a software reset of the MACA. After this reset, all internal registers are set to their reset values. This register also controls the signal that the MACA sends back to the clock controller module. If the MACA is not to be used for an extended period of time, the MACA clock can be disabled by writing a 0 to the CLK_ON bit. This saves additional power.

| | MACA_RESET | | | | | | | | Base + 0x04 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | Reserved | | | | | | | | | | | | | | CLK_ON | RST |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 9-2. MACA_RESET Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-2 | **Reserved** | |
| 1 | **CLK_ON** - Setting this bit causes the clock control module to activate the clock to the MACA. The default setting is “0”. By default the MACA is on. To save additional power when the MACA is not in use, the software can clear this bit to turn the MACA clocks off. | 1 = MACA clock on<br>0 = MACA clock off |
| 0 | **RST** - MACA Reset. Setting this bit initiates a module reset, completed by reset interrupt. This is NOT a self clearing bit. To release the module from reset, this bit must be cleared. This allows the software to hold the module in reset for any length of time. | 1 = Reset<br>0 = Do not reset |

## 9.7.2 Random Number Register (MACA_RANDOM)

The MACA_RANDOM register generates random numbers. Writing to this register initializes the engine with a seed. Reading from this register returns a new 32-bit random value every time the register is read.

| | MACA_RANDOM | | | | | | | | Base + 0x08 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | (R)DATA / (W)SEED | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | (R)DATA / (W)SEED | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 9-3. MACA_RANDOM Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| Bit 31-0 | **DATA[31:0]** - Read random data. Every read access returns a new random value. | Read |
| Bit 31-0 | **SEED[31:0]** - Generator seed. Initializes the random generator with new seed. | Write |

## 9.7.3 Control Register (MACA_CONTROL)

The MACA_CONTROL Register controls the MACA. This register starts auto sequences (receive, transmit), and associated behavior for these sequences. If timer triggers are used for the sequence, these must be setup before writing to the control register. Writing to the control register clears the status register and sets the complete code to “not completed”.

**NOTE**

All bits must be properly set for the entire sequence when writing to the MACA_CONTROL register. Any further writes are taken as start-up of a new auto sequence.

| | MACA_CONTROL | | | | | | | | Base + 0x0C | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | Reserved | | | | | | | | | | | ISM | PRE_COUNT | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | RSTO | Rsv | ROLE | NOFC | PRM | REL | ASAP | BCN | AUTO | LFSR | TM | MODE | | SEQUENCE | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 9-4. MACA_CONTROL Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-21 | **Reserved** | |
| 20 | **ISM** - Reserved | |
| 19-16 | **PRE_COUNT[3:0]** - Preamble repeat counter. This field can be set to a value of 0 to 15 and sets the number of repeats. To repeat the preamble pattern 8 times, set this field to 7. | |
| 15 | **RSTO** - Reset Slot Offset . Setting this bit resets the internal 20-counter for slotted CSMA-CA access. For TX, the counter is reset after transmission of length field. For RX, the counter is reset after reception of length field. | 1 = Reset counter<br>0 = Do not reset counter |
| 14 | **Reserved** | |
| 13 | **ROLE** - Current Role. This bit sets the operating role needed for proper header filtering. | 1 = PAN Coordinator<br>0 = Not PAN Coordinator |
| 12 | **NOFC** - No Frame Check. Setting this bit disables check for correct checksum on received packets. | 1 = Disable FCS<br>0 = Enable FCS |
| 11 | **PRM** - Promiscuous Mode. Setting this bit enables promiscuous mode. No data filtering is performed on received packets. | 1 = Promiscuous mode<br>0 = Normal mode |
| 10 | **REL** - Clock Selector. This bit selects either use of absolute or relative clock for timing actions in the sequence. | 1 = Relative clock<br>0 = Absolute clock |

**Table 9-4. MACA_CONTROL Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 9 | **ASAP** - Start Action Sequence ASAP. Setting this bit starts the sequence immediately. Otherwise, the start-up is postponed until the start clock condition is satisfied. | 1 = Start ASAP<br>0 = Start timer triggered |
| 8 | **BCN** - Filter Beacon Only. In receive mode, setting this bit filters out packets which are not beacon packets. | 1 = Receive only beacon packets<br>0 = Receive all packets |
| 7 | **AUTO** - Restart Rx Automatically. In receive mode, this bit instructs the sequencer to restart receiver after receiving data. Otherwise the sequence completes when the first valid data has been received. | 1 = Restart Rx<br>0 = Complete after first packet |
| 6 | **LFSR** - Reserved | |
| 5 | **TM** - Test mode. Setting this bit either forces the MACA to continuously send the preamble pattern, or forces the MACA to continuously receive data.<br>• Write 32'h00000223 to the Control Register for continuous TX.<br>• Write 32'h00000224 to the Control Register for continuous RX. | 1=test mode<br>0=normal mode |
| 4-3 | **MODE[4:3]** - Transmission Access Mode. This control field sets up transmission with CCA, and mode (slotted CSMA-CA has 2 CCA, non-slotted CSMA-CA has 1 CCA prior to transmission). | 0 = No CCA<br>1 = Non slotted CSMA-CA<br>2 = Slotted CSMA-CA<br>3 = Reserved |
| 2-0 | **SEQUENCE[2:0]** - Sequence. This field sets the new mode of operation of the MACA. Abort forces the sequencer into idle. Wait is a sequence which is idle, but generates a complete on time-out. Poll is an extended Tx sequence for transmitting MAC data requests. CCA and Energy Detect (ED) are single actions. | 0 = Nop<br>1 = Abort<br>2 = Wait<br>3 = Tx<br>4 = Rx<br>5 = TxPoll<br>6 = CCA<br>7 = ED |

## 9.7.4 Status Register (MACA_STATUS)

The MACA_STATUS Register contains the completion codes, which provide additional information from the MACA when a completion action interrupt is generated. This register is cleared when writing to the MACA_CONTROL register which sets the COMPLETE_CODE to “Not completed”.

| | MACA_STATUS | | | | | | | | Base + 0x10 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | TO | CRC | BUSY | OVR | Rsvd | Reserved | | | | | | | COMPLETE_CODE | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 9-5. MACA_STATUS Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-16 | **Reserved** | |
| 15 | **TO** - Time-out. This bit is set when a time-out occurred in the last completed sequence. | 1 = Time-out<br>0 = No time-out |
| 14 | **CRC -** Checksum Failed. If this bit is set, the last sequence detected a bad CRC. | 1 = CRC failure detected<br>0 = No CRC failure detected |
| 13 | **BUSY -** Channel Busy Detection. The CCA detected a busy channel. | 1 = Busy channel detected<br>0 = No busy channel detected |
| 12 | **OVR -** Rx Buffer Overrun. No Rx DMA point available when data received. Packet discarded. | 1 = Rx overrun detected<br>0 = No overrun detected |
| 11 | **Reserved** | |

**Table 9-5. MACA_STATUS Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 10-4 | **Reserved** | |
| 3-0 | **COMPLETE_CODE[3:0] -** Sequence Complete Codes. This field reflects the status of the last completed sequence. | 0 = Success<br>1 = Time-out<br>2 = Channel Busy<br>3 = CRC failed<br>4 = Aborted<br>5 = No Ack<br>6 = No data<br>7 = Late start<br>8 = ExtTimeout<br>9 = ExtPndTimeout<br>10 = not used<br>11 = not used<br>12 = PLL unlock<br>13 = External abort<br>14 = Not completed<br>15 = DMA bus error |

## 9.7.5 Frame Pending Register (MACA_FRMPND)

The MACA_FRMPND Register sets the proper frame pending information in acknowledgement response to a MAC data request packet. This reflects the status of the indirect transmission queue (empty or not).

| | MACA_FRMPND | | | | | | | | Base + 0x14 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | Reserved | | | | | | | | | | | | | | | PND |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 9-6. MACA_FRMPND Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-1 | **Reserved** | |
| 0 | **PND -** Acknowledgement Frame Pending Status. This bit is copied into acknowledgement responses to MAC data request packets and instructs the remote side to open its receiver for data. | 1 = Data available<br>0 = No data pending |

## 9.7.6 MC1322X_ID Register (MACA_MC1322x_ID)

The MACA_MC1322X_ID Register contains the version code for the MC1322x device. Valid version codes include:

- Engineering silicon - 0x0001_0000
- Later MC13224 silicon
  - — 0x0000_0001
  - — 0x0000_0009
- MC13226 silicon
  - — 0x0000_0011

| | MACA_MC1322X_ID | | | | | | | | Base + 0x18 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | X |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | Reserved | | | | | | | MC1322X_ID | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | X | X | X | X | X | X | X | X | X |

**Table 9-7. MACA_MC1322x_ID Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-9 | **Reserved** | |
| 8-0 | **MC1322X_ID[8:0]** - This field is the 9-bit MC1322x ID | See above |

## 9.7.7 Enable Timers Register (MACA_TMREN)

The MACA_TMREN Register enables the timers. Only timers selected by the respective bits are affected. Reading this register returns the status of the individual timers.

| | MACA_TMREN | | | | | | | | Base + 0x40 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | Reserved | | | | | | | | | | | | | SFT | CPL | STRT |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 9-8. MACA_TMREN Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-3 | **Reserved** | |
| 2 | **SFT -** Enable Soft Complete Clock. This bit enables the soft complete clock. | 1 = Enable clock<br>0 = Clock not affected |
| 1 | **CPL -** Enable Complete Clock. This bit enables the complete clock circuit. | 1 = Enable complete clock<br>0 = Clock not affected |
| 0 | **STRT -** Enable Start Clock. This bit enables the start clock circuit. | 1 = Enable start clock<br>0 = Clock not affected |

## 9.7.8 Disable Timers Register (MACA_TMRDIS)

The MACA_TMRDIS register disables the timers. Only the timers selected by the respective bits are affected. Reading this register returns the status of the individual timers.

| | MACA_TMRDIS | | | | | | | | Base + 0x44 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | Reserved | | | | | | | | | | SFT_OFF | CPL_OFF | STRT_OFF | SFT | CPL | STRT |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 9-9. MACA_TMRDIS Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-4 | **Reserved** | |
| 5 | **SFT_OFF** - Abort Soft Complete Clock. This bit aborts a running soft complete clock, and terminates it immediately. If it is armed, a time-out is generated. | 1 = Abort soft clock<br>0 = Clock not affected |
| 4 | **CPL_OFF** - Abort Complete Clock. This bit aborts a running complete clock, and terminates it immediately. If it is armed, a time-out is generated. | 1 = Abort complete clock<br>0 = Clock not affected |
| 3 | **STRT_OFF** - Abort Start Clock. This bit aborts a running start clock, and terminates it immediately. If it is armed, a time-out is generated. | 1 = Abort start clock<br>0 = Clock not affected |
| 2 | **SFT** - Disable Soft Complete Clock. This bit disables the c'soft' complete clock circuit. | 1 = Disable soft clock<br>0 = Clock not affected |
| 1 | **CPL** - Disable Complete Clock. This bit disables the complete clock circuit. | 1 = Disable complete clock<br>0 = Clock not affected |
| 0 | **STRT** - Disable Start Clock. This bit disables the start clock circuit. | 1 = Disable start clock<br>0 = Clock not affected |

## 9.7.9 Clock Register (MACA_CLK)

The MACA_CLK Register is a 32-bit clock running at 250kHz. Writing to this register sets a new absolute clock value. Ensure that the timers are not active because they will not be modified accordingly.

| | MACA_CLK | | | | | | | | Base + 0x48 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | ZIGBEE_CLOCK | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | ZIGBEE_CLOCK | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 9-10. MACA_CLK Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-0 | **ZIGBEE_CLOCK[31:0]** - ZigBee (802.15.4 Standard) Absolute Clock. Reflects the current absolute clock. | 250 kHz Clock rate |

## 9.7.10 Start Clock Register (MACA_STARTCLK)

The MACA_STARTCLK Register is a one-shot trigger used for starting actions. The clock is matched against either the absolute or relative clock, depending on the “SEL” bit in the MACA_TMREN register. If enabled, and once a match occurs, an ActionStarted_IRQ interrupt is generated and the timer is automatically disabled. In addition, the sequencer state machine uses this as an internal signal to proceed with its current auto-sequence.

| | MACA_STARTCLK | | | | | | | | Base + 0x4C | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | START_CLOCK | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | STAR_CLOCK (cont) | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 9-11. MACA_STARTCLK Register Bit Descriptions**

| Name | Description | Settings |
|---|---|---|
| 31-0 | **START_CLOCK[31:0]** - 802.15.4 Standard Trigger Start Clock. The desired trigger time for an action to start. | 250 kHz clock rate |

## 9.7.11 Complete Clock Register (MACA_CPLCLK)

The MACA_CPLCLK Register is a one-shot trigger used for terminating actions (primarily receive actions). The clock is matched against either the absolute or relative clock, depending of the “SEL” bit in the MACA_TMREN register. If enabled, and once a match occurs, an internal signal is generated for use in the sequencer. Normally, it terminates a receive sequence.

| | MACA_CPLCLK | | | | | | | | Base + 0x50 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | COMPLETE_CLOCK | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | COMPLETE_CLOCK | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 9-12. MACA_CPLCLK Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-0 | **COMPLETE_CLOCK[31:0]** - 802.15.4 Standard Trigger Complete Clock. The desired trigger time for an action to finish. | 250 kHz clock rate |

## 9.7.12 Soft Time-out Clock Register (MACA_SFTCLK)

The MACA_SFTCLK Register is a one-shot trigger that completes receive actions with the “extended” time-out. The clock is matched against either the absolute or relative clock, depending of the “SEL” bit in the MACA_TMREN register. If enabled, and once a match occurs, an internal signal is generated for use in the sequencer. See Section 9.5.1.3.2, “Soft Time-out”.

| | MACA_SFTCLK | | | | | | | | Base + 0x54 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | SOFT_CLOCK | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | SOFT_CLOCK | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 9-13. MACA_SFTCLK Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-0 | **SOFT_CLOCK[31:0]** - Soft Complete Clock. The desired trigger time for an action to finish if no packet is detected. Trigger is postponed until packet completion (or rejection) if receiver has detected SFD. | 250 kHz clock rate |

## 9.7.13 Clock Offset Register (MACA_CLKOFFSET)

The MACA_CLKOFFSET Register is the offset between the absolute and relative clock. Updating the relative clock also updates this register. Likewise, updating the offset register updates the relative clock. The relative clock is defined as absolute clock + offset.

| | MACA_CLKOFFSET | | | | | | | | Base + 0x58 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | CLOCK_OFFSET | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | CLOCK_OFFSET | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 9-14. MACA_CLKOFFSET Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-0 | **CLOCK_OFFSET[31:0]** - Clock offset. The offset between the absolute clock and the relative clock. | 250 kHz clock rate |

## 9.7.14 Relative Clock Register (MACA_RELCLK)

The MACA_RELCLK Register is a 32-bit clock running at 250kHz and is based on the absolute clock + offset. Writing to this register sets a new relative clock and updates the offset. Ensure that the timers are not active because they will not be modified accordingly.

| | MACA_RELCLK | | | | | | | | Base + 0x5C | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | RELATIVE_CLOCK | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | RELATIVE_CLOCK | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 9-15. MACA_RELCLK Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-0 | **RELATIVE_CLOCK[31:0]** - Relative Clock. Reflects the current relative clock, and sets a new relative clock. | 250 kHz clock rate |

## 9.7.15 Action Complete Timestamp Register (MACA_CPLTIM)

The MACA_CPLTIM Register is a timestamp latched upon the completion of an auto-sequence. Depending on the “SEL” bit in the MACA_TMREN register, either the absolute or relative clock is latched.

| | MACA_CPLTIM | | | | | | | | Base + 0x60 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | COMPLETE_TIME | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | COMPLETE_TIME | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 9-16. MACA_CPLTIM Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-0 | **COMPLETE_TIME[31:0]** - Complete Time. Timestamp of the time a sequence completed. | 250 kHz clock rate |

## 9.7.16 Slot Offset Adjustment Register (MACA_SLOTOFFSET)

The MACA_SLOTOFFSET Register properly aligns access to the channel in slotted CSMA-CA mode. An internal backoff counter counts from 0 to 79. Access is then only allowed every 20 symbols (clock runs at 4x symbol speed). To account for radio setup time and radio delay chains, the MACA_SLOTOFFSET sets up an initial offset to the received/transmitted beacon.

If the “RSTO” bit is set in the control register, and a beacon is received, the internal backoff counter is loaded with the contents of MACA_SLOTOFFSET (the offset is loaded after reception of the length field). If the “RSTO” is set, and the sequence is a transmission, the offset is loaded after transmission of the length field.

| | MACA_SLOTOFFSET | | | | | | | | Base + 0x64 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | Reserved | | | | RX_SLOT_OFFSET | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | Reserved | | | | TX_SLOT_OFFSET | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 9-17. MACA_SLOTOFFSET Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-28 | **Reserved** | |
| 27-16 | **RX_SLOT_OFFSET[11:0]** - Receive Slot Offset. The value for “resetting” the internal backoff counter for slotted CSMA-CA access. Each bit corresponds to an offset of 0.25 symbols. This values is used when synchronizing to a received beacon. | |
| 15-12 | **Reserved** | |
| 11-0 | **TX_SLOT_OFFSET[11:0]** - Transmit Slot Offset. The value used for ”resetting” the internal backoff counter for slotted CSMA-CA access. Each bit corresponds to an offset of 0.25 symbols. This value is used when transmitting a beacon. | |

## 9.7.17 Receive Time Stamp Register (MACA_TIMESTAMP)

The MACA_TIMESTAMP Register contains the latched clock value for the last received packet. The clock value is latched immediately after receiving the length field. Depending on the “SEL” bit in the MACA_TMREN register, either the relative or absolute clock is latched.

| | MACA_TIMESTAMP | | | | | | | | Base + 0x68 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | TIMESTAMP | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | TIMESTAMP | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 9-18. MACA_TIMESTAMP Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-0 | **TIMESTAMP[31:0]** - Timestamp. Latched clock time stamping the last received packet. | 250 kHz clock |

## 9.7.18 DMA Rx Data Pointer Register (MACA_DMARX)

The MACA_DMARX Register sets up a receive byte address pointer used for DMA transfer. The DMA transfers data from the MACA receive queue into the memory location referred to by the MACA_DMARX register one byte at-a-time. The register is not updated during reception of a packet because this allows the sequencer to reload the pointer if, for example, the CRC fails on a received packet.

Writing to this register indicates buffer availability to the sequencer. When a DataIndication_IRQ is generated, the buffer is used. If no write to the MACA_DMARX is performed before a new packet sync is found, the sequencer handles the received packet as having a “bad” header. In addition, the “OVR” bit in the status register is set.

| | MACA_DMARX | | | | | | | | Base + 0x80 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | DEST_PTR | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | DEST_PTR | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 9-19. MACA_DMARX Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-0 | **DEST_PTR[31:0]** - DMA Destination Pointer. Pointer to memory where to transfer data. | |

## 9.7.19 DMA Tx Data Pointer Register (MACA_DMATX)

The MACA_DMATX register sets up a transmission pointer used for DMA transfer. The DMA transfers data from memory to the MACA transmit queue on a byte-by-byte basis. The register is not updated during transmission of a packet.

| | MACA_DMATX | | | | | | | | Base + 0x84 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | SRC_PTR | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | SRC_PTR | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 9-20. MACA_DMATX Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-0 | **SRC_PTR[31:0]** - DMA Source Pointer. Pointer in memory from which to get Tx data. | |

## 9.7.20 DMA Tx Poll Response Pointer Register (MACA_DMAPOLL)

The MACA_DMAPOLL Register sets up a transmission pointer used for DMA transfer during a fast poll response sequence. Writing to this register arms an immediate reply to a remote data request (See Section 9.5.1.3.5, “Handling Received MAC Data Request”). The DMA transfers data from memory to the MACA transmit queue on a byte-by-byte basis. The register is not updated during transmission of a packet.

| | MACA_DMAPOLL | | | | | | | | Base + 0x88 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | SRC_PTR | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | SRC_PTR | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 9-21. MACA_DMAPOLL Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-0 | **SRC_PTR[31:0]** - DMA Source Pointer. Pointer in memory from which to get Tx data. | |

## 9.7.21 Tx Length Register (MACA_TXLEN)

The MACA_TXLEN Register specifies how many bytes of data is to be transmitted during a Tx auto-sequence (including checksum, but excluding PHY header). This register should be written prior to MACA_DMATX or MACA_DMAPOLL.

| | MACA_TXLEN | | | | | | | | Base + 0x8C | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | RES | RX_LEN | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | RES | TX_LEN | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 9-22. MACA_TXLEN Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31 | **Reserved** | |
| 30-16 | **RX_LEN[14:0]** - **Reserved** | |
| 15 | Reserved | |
| 14-0 | **TX_LEN[14:0]** - Tx payload length. Length of transmitted data including checksum. In 802.15.4 Standard mode only the lower 7 bits are used.<br>{choose_lfsr,tx_len} | |

## 9.7.22 Tx Sequence Number Register (MACA_TXSEQNR)

The MACA_TXSEQNR Register is a copy of the sequence number embedded in the data to transmit. It validates acknowledgement. If acknowledgement is not required, this register need not be updated for transmissions. This register must be written prior to MACA_DMATX or MACA_DMAPOLL.

| | MACA_TXSEQNR | | | | | | | | Base + 0x90 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | Reserved | | | | | | | | TXSEQN | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 9-23. MACA_TXSEQNR Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-8 | **Reserved** | |
| 6-0 | **TXSEQN[6:0]** - Tx Sequence Number. Sequence number of transmitted packet for validating acknowledgement. | |

## 9.7.23 Set Rx Level Interrupt Register (MACA_SETRXLVL)

The MACA_SETRXLVL Register specifies the trigger level for generating an IRQ based on the amount of data received on incoming packets. Once the level is reached (data is transferred to the destination RAM using DMA), an RxLevel_IRQ is generated.

It is possible to update the level register during packet reception which allows for multiple interrupts for packet processing.

| | MACA_SETRXLVL | | | | | | | | Base + 0x94 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | FIFO_LVL | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 9-24. MACA_SETRXLVL Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-16 | **Reserved** | |
| 15-0 | **FIFO_LVL[15:0]** - FIFO Level. Indicates the number of bytes to receive before generating an RxLevel_IRQ. | |

## 9.7.24 Read Number Of Received Bytes Register (MACA_GETRXLVL)

Reading the MACA_GETRXLVL Register (read only) returns the number of bytes already received and transferred to memory using the DMA.

| | MACA_GETRXLVL | | | | | | | | Base + 0x98 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | RecvBytes | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 9-25. MACA_GETRXLVL Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-16 | **Reserved** | |
| 15-0 | **RecvBytes[15:0] -** Received Bytes. Indicates the number of available received bytes. | |

## 9.7.25 Interrupt Status Register (MACA_IRQ)

The MACA_IRQ Register is a read-only register that contains interrupt request source status. Status is cleared by writing to the Interrupt Clear Register (MACA_CLRIRQ).

| | MACA_IRQ | | | | | | | | Base + 0xC0 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Field | STRT | SYNC | CM | CRC | FLT | SFT | LVL | Reserved | | | | RST | WU | DI | POLL | ACPL |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 9-26. MACA_IRQ Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-16 | **Reserved** | |
| 15 | **STRT** - Action Started Interrupt. An auto-sequence is started, either immediately or by timer trigger. | 1 = Status valid<br>0 = Status invalid |
| 14 | **SYNC** - Sync Detected Interrupt. The modem has detected the beginning of a new packet | 1 = Status valid<br>0 = Status invalid |
| 13 | **CM** - Complete Clock Interrupt. The complete clock has generated a trigger. | 1 = Status valid<br>0 = Status invalid |
| 12 | **CRC** - Checksum Failed Interrupt. The checksum failed for the received packet. | 1 = Status valid<br>0 = Status invalid |
| 11 | **FLT** - Filter Failed Interrupt. The receive header filter failed. | 1 = Status valid<br>0 = Status invalid |
| 10 | **SFT** - Soft Complete Clock Interrupt. The soft complete clock has generated a trigger. | 1 = Status valid<br>0 = Status invalid |
| 9 | **LVL** - FIFO Level interrupt. The receive FIFO level is reached or exceeded. | 1 = Status valid<br>0 = Status invalid |
| 8-5 | **Reserved** | |
| 4 | **RST** - Reserved | |
| 3 | **WU** - Reserved | |
| 2 | **DI** - Data Indication Interrupt. During receive, a packet was successfully received. | 1 = Status valid<br>0 = Status invalid |

**Table 9-26. MACA_IRQ Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 1 | **POLL** - Poll Indication Interrupt. Issued when data request received (and before ACK transmitted). MCU may then set MACA_FRMPND and prepare fast response. | |
| 0 | **ACPL** - Action Complete Interrupt. Marks the completion of a complete auto-sequence. | 1 = Status valid<br>0 = Status invalid |

## 9.7.26 Interrupt Clear Register (MACA_CLRIRQ)

The MACA_CLRIRQ Register (write only) clears interrupt sources. Writing bit-fields to 1 clears interrupts selected by set bits. Internally, the MACA module inverts the bit-field and "AND's" it on the interrupt sources. For example, writing 0x0001 clears interrupt source "0" (ACPL), but other sources are untouched.

| | MACA_CLRIRQ | | | | | | | | Base + 0xC4 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | STRT | SYNC | CM | CRC | FLT | SFT | LVL | Reserved | | | | RST | WU | DI | POLL | ACPL |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 9-27. MACA_CLRIRQ Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-16 | **Reserved** | |
| 15 | **STRT** - Action Started Interrupt. An auto-sequence is started, either immediately or by timer trigger. | Write 1 to clear status |
| 14 | **SYNC** - Sync Detected Interrupt. The modem has detected the beginning of a new packet | Write 1 to clear status |
| 13 | **CM** - Complete Clock Interrupt. The complete clock has generated a trigger. | Write 1 to clear status |
| 12 | **CRC** - Checksum Failed Interrupt. The checksum failed for the received packet. | Write 1 to clear status |
| 11 | **FLT** - Filter Failed Interrupt. The receive header filter failed. | Write 1 to clear status |
| 10 | **SFT** - Soft Complete Clock Interrupt. The soft complete clock has generated a trigger. | Write 1 to clear status |
| 9 | **LVL** - FIFO Level interrupt. The receive FIFO level is reached or exceeded. | Write 1 to clear status |

**Table 9-27. MACA_CLRIRQ Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 8-5 | **Reserved** | |
| 4 | **RST** - Reserved | |
| 3 | **WU** - Reserved | |
| 2 | **DI** - Data Indication Interrupt. During receive, a packet was successfully received. | Write 1 to clear status |
| 1 | **POLL** - Poll Indication Interrupt. Issued when data request received (and before ACK transmitted). MCU may then set MACA_FRMPND and prepare fast response. | |
| 0 | **ACPL** - Action Complete Interrupt. Marks the completion of a complete auto-sequence. | Write 1 to clear status |

## 9.7.27 Interrupt Set Register (MACA_SETIRQ)

Useful for debugging software, the MACA_SETIRQ Register (write only) forces interrupt sources. Writing a bit-field sets interrupts selected by set bits. Internally, the MACA module “OR’s” the bit-field on the interrupt sources. For example, writing 0x0001, sets interrupt source “0” (ACPL), but other sources are untouched. Clear sources via the MACA_CLRIRQ Register .

| | MACA_SETIRQ | | | | | | | | Base + 0xC8 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | STRT | SYNC | CM | CRC | FLT | SFT | LVL | Reserved | | | | RST | WU | DI | POLL | ACPL |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 9-28. MACA_SETIRQ Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-16 | **Reserved** | |
| 15 | **STRT** - Action Started Interrupt. An auto-sequence is started, either immediately or by timer trigger. | Write 1 to set IRQ source |
| 14 | **SYNC** - Sync Detected Interrupt. The modem has detected the beginning of a new packet | Write 1 to set IRQ source |
| 13 | **CM** - Complete Clock Interrupt. The complete clock has generated a trigger. | Write 1 to set IRQ source |

**Table 9-28. MACA_SETIRQ Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 12 | **CRC** - Checksum Failed Interrupt. The checksum failed for the received packet. | Write 1 to set IRQ source |
| 11 | **FLT** - Filter Failed Interrupt. The receive header filter failed. | Write 1 to set IRQ source |
| 10 | **SFT** - Soft Complete Clock Interrupt. The soft complete clock has generated a trigger. | Write 1 to set IRQ source |
| 9 | **LVL** - FIFO Level interrupt. The receive FIFO level is reached or exceeded. | Write 1 to set IRQ source |
| 8-5 | **Reserved** | |
| 4 | **RST** - Reserved | |
| 3 | **WU** - Reserved | |
| 2 | **DI** - Data Indication Interrupt. During receive, a packet was successfully received. | Write 1 to set IRQ source |
| 1 | **POLL** - Poll Indication Interrupt. Issued when data request received (and before ACK transmitted). MCU may then set MACA_FRMPND and prepare fast response. | |
| 0 | **ACPL** - Action Complete Interrupt. Marks the completion of a complete auto-sequence. | Write 1 to set IRQ source |

## 9.7.28 Interrupt Mask Register (MACA_MASKIRQ)

The MACA_MASKIRQ register enables interrupts. Only enabled interrupt sources generate a MACA interrupt the MCU. Write a 1 to enable the IRQ.

| | MACA_MASKIRQ | | | | | | | | Base + 0xCC | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | STRT | SYNC | CM | CRC | FLT | SFT | LVL | Reserved | | | | | | DI | POLL | ACPL |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 9-29. MACA_MACKRIQ Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-16 | **Reserved** | |
| 15 | **STRT** - Action Started Interrupt. An auto-sequence is started, either immediately or by timer trigger. | Write 1 to enable IRQ |
| 14 | **SYNC** - Sync Detected Interrupt. The modem has detected the beginning of a new packet | Write 1 to enable IRQ |
| 13 | **CM** - Complete Clock Interrupt. The complete clock has generated a trigger. | Write 1 to enable IRQ |
| 12 | **CRC** - Checksum Failed Interrupt. The checksum failed for the received packet. | Write 1 to enable IRQ |
| 11 | **FLT** - Filter Failed Interrupt. The receive header filter failed. | Write 1 to enable IRQ |
| 10 | **SFT** - Soft Complete Clock Interrupt. The soft complete clock has generated a trigger. | Write 1 to enable IRQ |
| 9 | **LVL** - FIFO Level interrupt. The receive FIFO level is reached or exceeded. | Write 1 to enable IRQ |
| 8-5 | **Reserved** | |
| 4 | **RST** - Reserved | |
| 3 | **WU** - Reserved | |
| 2 | **DI** - Data Indication Interrupt. During receive, a packet was successfully received. | Write 1 to enable IRQ |
| 1 | **POLL** - Poll Indication Interrupt. Issued when data request received (and before ACK transmitted). MCU may then set MACA_FRMPND and prepare fast response. | |
| 0 | **ACPL** - Action Complete Interrupt. Marks the completion of a complete auto-sequence. | Write 1 to enable IRQ |

## 9.7.29 MAC PAN ID Register (MACA_MACPANID)

The MACA_MACPANID Register provides the header filtering block with the needed MAC PAN ID value.

| | MACA_MACPANID | | | | | | | | Base + 0x100 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | PANID | | | | | | | | | | | | | | | |
| Reset | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

**Table 9-30. MACA_MACPANID Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-16 | **Reserved** | |
| 15-0 | **PANID[15:0]** - MAC PAN ID. Reflects the current MAC PAN ID for the 802.15.4 Standard network. | |

## 9.7.30 MAC Short Address Register (MACA_MAC16ADDR)

The MACA_MAC16ADDR Register is used in the header filtering and contains the assigned 16-bit MAC short address of the device.

| | MACA_MAC16ADDR | | | | | | | | Base + 0x104 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | ADDR | | | | | | | | | | | | | | | |
| Reset | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

**Table 9-31. MACA_MAC16ADDR Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-16 | **Reserved** | |
| 15-0 | **ADDR[15:0]** - This field reflects the current MAC short address for the device. | |

## 9.7.31 High MAC Extended Address Register (MACA_MAC64HI)

The MACA_MAC64HI Register is the upper part of the assigned 64-bit IEEE address. Together with MACA_MAC64LO, it is used for header filtering on extended (64-bit) addresses.

| | MACA_MAC64HI | | | | | | | | Base + 0x108 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | EXT_ADDR_HI | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | EXT_ADDR_HI | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Figure 9-8. MACA_MAC64HI Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-0 | EXT_ADDR_HI[31:0] - High part of Extended Address. Reflects the assigned IEEE address. | |

## 9.7.32 Low MAC Extended Address Register (MACA_MAC64LO)

The MACA_MAC64LO Register is the lower part of the assigned 64-bit IEEE address.

| | MACA_MAC64LO | | | | | | | | Base + 0x10C | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | EXT_ADDR_LOI | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | EXT_ADDR_LO | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 9-32. MACA_MAC64LO Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-0 | EXT_ADDR_LO[31:0] - Low part of Extended Address. Reflects the assigned IEEE address. | |

## 9.7.33 Filter Rejection Mask Register (MACA_FLTREJ)

The MACA_FLTREJ Register controls rejection of incoming packets during header filtering.

| | MACA_FLTREJ | | | | | | | | Base + 0x110 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | FC_MASK | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | Reserved | | | | | | | POLL | Reserved | | | | CMD | ACK | DATA | BCN |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 9-33. MACA_FLTREJ Register Bit Descriptions**

| Name | Description | Settings |
|---|---|---|
| 31-16 | **FC_MASK[15:0]** - Frame Control Mask. The mask identifies which bits in frame control must be zero for packet exception. | 1 = Bit must be zero<br>0 = Bit is don't care |
| 15-9 | **Reserved** | |
| 8 | **POLL** - Accept POLL packets. If this bit is set, only MAC Commands, which are actual "Data.request" (POLL) are accepted. | 1 = Accept only POLL<br>0 = No restriction |
| 7-4 | **Reserved** | |
| 3 | **CMD** - Reject MAC CMD packets. If this bit is set, MAC command types are rejected. | 1 = Reject MAC CMD<br>0 = No rejection |
| 2 | **ACK** - Reject ACK packets. If this bit is set, ACK packet types are rejected. | 1 = Reject ACK packet<br>0 = No rejection |
| 1 | **DATA** - Reject Data packets. If this bit is set, Data packet types are rejected. | 1 = Reject data packet<br>0 = No rejection |
| 0 | **BCN** - Reject Beacon packets. If this bit is set, beacon packet types are rejected. | 1 = Reject beacon packet<br>0 = No rejection |

## 9.7.34 Clock Divider Register ( MACA_CLKDIV)

The write-only MACA_CLKDIV Register sets the divider to generate the MACA transmit clock.

**NOTE**

- The clock source to the MACA is the Peripheral Clock set by the CRM. The divide ratio N = Divide[15:0] + 1, and should set the bit rate to 250 KHz to match the 802.15.4 250 kb/s rate.
- Typical value is 95 (dec) for Peripheral Clock of 24 MHz

| | MACA_CLKDIV | | | | | | | | Base + 0x114 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | Divider | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 9-34. MACA_CLKDIV Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-16 | **Reserved** | |
| 15-0 | **Divider** - This field sets the prescale value for the MACA transmit clock.<br>• Divide ratio N = Divider[15:0] + 1.<br>• Transmit bit clock must be 250 kHz for 802.15.4<br>• For example, if the peripheral clock is 24Mhz, set divider to 95 (0x5F) to set the transmit bit clock to 250khz. | |

## 9.7.35 Warmup Register (MACA_WARMUP)

The MACA_WARMUP Register sets the values for Tx and Rx warmup.

| | MACA_WARMUP | | | | | | | | Base + 0x118 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | Reserved | | | | TX_WARMUP | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | Reserved | | | | RX_WARMUP | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 9-35. MACA_WARMUP Bits**

| Bit Number | Description | Operation |
|---|---|---|
| 31-28 | **Reserved** | |
| 27-16 | **TX_WARMUP[11:0]** - This field sets the warmup time for the transmitter. For example, if this value is set to 10, then the sequencer tells the modem to start its warmup sequence 10 bits before the first bit of preamble is sent. | |
| 15-12 | **Reserved** | |
| 11-0 | **RX_WARMUP[11:0]** - This field sets the warmup time for the receiver. For example, if the value is set to 10, then the sequencer tells the modem to start its warmup sequence 10 bits before the first bit of preamble is expected to be received. | |

## 9.7.36 Preamble Register (MACA_PREAMBLE)

The MACA_PREAMBLE Register sets the preamble pattern.

| | MACA_PREAMBLE | | | | | | | | Base + 0x11C | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | PREAMBLE | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | PREAMBLE | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 9-36. MACA_PREAMBLE Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-0 | **PREAMBLE[31:0]** - This register sets the preamble pattern. In the Control Register there is a 4 bit field that sets the number of times this pattern is repeated. | |

## 9.7.37 Frame Sync Word 0 Register (MACA_FRAMESYNC0)

The MACA_FRAME_SYNC0 register sets the first half of the framesync pattern.

| | MACA_FRAMESYNC0 | | | | | | | | Base + 0x124 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | FRAMESYNC0 | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | FRAMESYNC0 | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 9-37. MACA_FRAMESYNC0 Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-0 | **FRAMESYNC0[31:0]** - This register sets the lower half of the framesync pattern.The lower half is transmitted first. | |

## 9.7.38 Frame Sync Word 1 Register (MACA_FRAMESYNC1)

The MACA_FRAME_SYNC1 Register sets the second half of the framesync pattern.

| | MACA_FRAMESYNC1 | | | | | | | | Base + 0x128 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | FRAMESYNC1 | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | FRAMESYNC1 | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 9-38. MACA_FRAMESYNC1 Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-0 | FRAMESYNC1[31:0] - This register sets the upper half of the framesync pattern.The upper half is transmitted after the lower half. In 802.15.4 Standard mode, only the lower 8 bits are used. | |

## 9.7.39 Tx Acknowledgement Delay Register (MACA_TXACKDELAY)

The MACA_TXACKDELAY Register fine tunes acknowledgement transmissions to occur exactly 12 symbols after the last received bit. This register can adjust for radio warm-up and delay in Rx/Tx chain.

In slotted CSMA-CA mode, the delay is a “minimum” delay only. Transmission occurs on the first backoff slot after the delay. The delay is represented with a resolution of 0.25 symbols and a delay of 0 indicates that acknowledgement frames will be transmitted immediately after packet reception (For example, Tx warm-up starts immediately).

| | MACA_TXACKDELAY | | | | | | | | Base + 0x140 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | Reserved | | | | TXPOLLDELAY | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | Reserved | | | | TXACKDELAY | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 9-39. MACA_TXACKDELAY Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-28 | **Reserved** | |
| 27-16 | **TXPOLLDELAY[11:0]** - Tx PollDelay. Delay in 0.25 symbols from end of Poll command ack to beginning of transmitted polled data. This must greater than the transmitter warmup time. | |
| 15-12 | **Reserved** | |
| 11-0 | **TXACKDELAY[11:0]** - Tx Acknowledgement Delay. Delay in 0.25 symbols from packet reception to beginning of acknowledgement Tx warm-up. | |

## 9.7.40 Rx Acknowledgement Delay Register (MACA_RXACKDELAY)

The MACA_RXACKDELAY Register adjusts for radio warm-up and delay in the Rx/Tx chain. The delay is represented with a resolution of 0.25 symbols and a delay of 0 indicates that acknowledgement must be transmitted immediately after packet reception (For example, Rx warm-up starts immediately).

| | MACA_RXACKDELAY | | | | | | | | Base + 0x144 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | Reserved | | | | RXAUTODELAY | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | Reserved | | | | RXACKDELAY | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 9-40. MACA_RXACKDELAY Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-28 | **Reserved** | |
| 27-16 | **RXAUTODELAY[11:0]** - Length of time ( in bits) to disable receiver before restarting receiver. | |
| 15-12 | **Reserved** | |
| 11-0 | **RXACKDELAY[11:0]** - Rx Acknowledgement Delay. Determines the beginning of the acknowledgement frame search window. Measured in 0.25 symbols after Tx completed. | |

## 9.7.41 End of Frame Delay Register (MACA_EOFDELAY)

The MACA_EOFDELAY Register is a programmable delay for generation of the ActionComplete_IRQ interrupt. This ensures proper power down of radio chain before issuing a new action. This allows concurrent software processing while the radio is powering down. The delay is measured in increments of 0.25 symbols each.

| | MACA_EOFDELAY | | | | | | | | Base + 0x148 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | Reserved | | | | EOF_DELAY | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 9-41. MACA_EOFDELAY Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-12 | **Reserved** | |
| 11-0 | **EOF_DELAY[11:0]** - End of Frame Delay. Delay in 0.25 symbols from auto-sequence completes to generation of ActionComplete_IRQ interrupt. | |

## 9.7.42 CCA Delay Register (MACA_CCADELAY)

The MACA_CCADELAY Register fine tunes exactly when the CCA timing occurs. This register can adjust radio warm-up and delay in the Rx/Tx chain. The delay is represented with a resolution of 0.25 symbols. A delay of 0 indicates that the acknowledgement frame must be transmitted immediately after packet reception (For example, CCA warm-up starts immediately).

| | MACA_CCADELAY | | | | | | | | Base + 0x14C | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | | | | | CCALENGTH | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | | | | | CCADELAY | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 9-42. MACA_CCADELAY Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-28 | **Reserved** | |
| 27-16 | **CCALENGTH[11:0]** - Length of time to perform CCA. | |
| 15-12 | **Reserved** | |
| 11-0 | **CCADELAY[11:0]** - CCA Delay (in bits) from first CCA to second CCA in slotted CSMA-CA mode. | |

## 9.7.43 Rx End Register (MACA_RXEND)

This MACA_RXEND Register fine tunes reception of acknowledgement by specifying the end of the search window for slotted CSMA-CA and non-slotted CSMA-CA mode. The delay is represented with a resolution of 0.25 symbols. A delay of 0 indicates that the acknowledgement must be transmitted immediately after packet reception (For example, Tx warm-up starts immediately).

| | MACA_RXEND | | | | | | | | Base + 0x150 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | Reserved | | | | RXSLOTTED_END | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | Reserved | | | | RXACK_END | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 9-43. MACA_RXEND Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-28 | **Reserved** | |
| 23-16 | **RXSLOTTED_END[11:0]** - Rx Acknowledgement Window End in slotted CSMA-CA mode. Delay in 0.25 symbols from Tx completed to end of Rx ACK window in slotted CSMA-CA mode. | |
| 15-12 | **Reserved** | |
| 11-0 | **RXACK_END[11:0]** - Rx Acknowledgement Window End in Normal Mode. Delay in 0.25 symbols from Tx completed to end of Rx ACK window in normal mode | |

## 9.7.44 TX CCA Delay Register (MACA_TXCCADELAY)

The MACA_TXCCADELAY Register fine tunes exactly when the CCA timing occurs. This register can adjust for radio warm-up and delay in the Rx/Tx chain. The delay is represented with a resolution of 0.25 symbols. A delay of 0 indicates that the acknowledgement frame must be transmitted immediately after packet reception. (For example, CCA warm-up starts immediately).

| | MACA_TXCCADELAY | | | | | | | | Base + 0x154 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | Reserved | | | | TXCCADELAY | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 9-44. MACA_TXCCADELAY Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-12 | **Reserved** | |
| 11-0 | TXCCADELAY[11:0] - Delay (in bits) from end of CCA to TX start. | |

## 9.7.45 Random Key Registers (MACA_KEY3:MACA_KEY0)

The MACA provides 4 registers that each return a separate random number for use as an encryption key. Reading any of these registers returns a 32 bit random number.

- MACA_KEY3 - Address Base + 0x158
- MACA_KEY2 - Address Base + 0x15C
- MACA_KEY1 - Address Base + 0x160
- MACA_KEY0 - Address Base + 0x164

| | MACA_KEYX | | | | | | | | Base + 0x158-164 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | KEY | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | KEY | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 9-45. MACA_KEYX Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-0 | **KEY[31:0]** - 32-bit random value | |

## 9.7.46 MACA Options Register (MACA_OPTIONS)

The MACA_OPTIONS Register can modify the behavior of the MACA.

| | MACA_OPTIONS | | | | | | | | Base + 0x | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BIT | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| Field | Reserved | | | | | | | | | | | | | | | |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| BIT | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | Reserved | | | | | | | | | | | | SEED _KEY | PLL_ IGNORE | PLL_ TX | POLL |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 9-46. MACA_OPTIONS Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-4 | **Reserved** | |
| 3 | **SEED_KEY** - If this bit is a 1 then the key generation can be seeded with a write to the maca_random register. If this bit is a 0 then the register can not be seeded and will be in a random power up state. The flip-flops used in the key generation do not have a async reset. | 1 = Active<br>0 = Not active |
| 2 | **PLL_IGNORE** - If this bit is a 1 then the MACA ignores the PLL unlock signal. If it is a 0 then the MACA responds. | 1 = Active<br>0 = Not active |
| 1 | **PLL_TX** - If this bit is a 1 then the MACA only responds to the PLL unlock when the MACA is performing a transmit. If this bit is a 0 then the MACA responds as soon as PLL unlock is asserted. | 1 = Active<br>0 = Not active |
| 0 | **POLL** - If this bit is a 1 then the DMA Tx poll response pointer (MACA_DMAPOLL) can be written to at any time. This allows writing to the pointer during the window between PollRx_IRQ generation and the completion of the following acknowledgement. | 1 = Active<br>0 = Not active |

# Chapter 10
# Advanced Security Module (ASM)

## 10.1 Overview

The ASM block is a hardware engine that encrypts/decrypts using the Advanced Encryption Standard (AES). It can perform "Counter" (CTR) and Cipher Block Chaining (CBC) encryption. The combination of these two modes of encryption is known as Counter with CBC-MAC (CCM) mode encryption. CCM is a generic authenticate and encrypt block cipher mode. CCM is only defined for use with 128-bit block ciphers, such as AES. The definition of CCM mode encryption is documented in the NIST publication SP800-38C. Section 10.3.4, “CCM Mode (combined CTR and CBC-MAC)” of this chapter provides a brief summary of how CCM mode works.

### 10.1.1 Features

The ASM has the following features:

- 32-bit access
- CTR encryption in 13 clock cycles.
- CBC encryption in 13 clock cycles.
- Encrypts 128 bits as a unit.
- The 128-bit data values (16 bytes) are each accessed as four 32-bit registers which are aligned to the 32-bit word boundaries.
- The ASM is clocked at the peripheral clock rate selected in the CRM.

## 10.2 ASM Module Block Diagram

The structure of the ASM module is shown in the Figure 10-1 block diagram. This structure combines both the CTR mode and CBC-MAC mode. This is also known as CCM mode encryption.

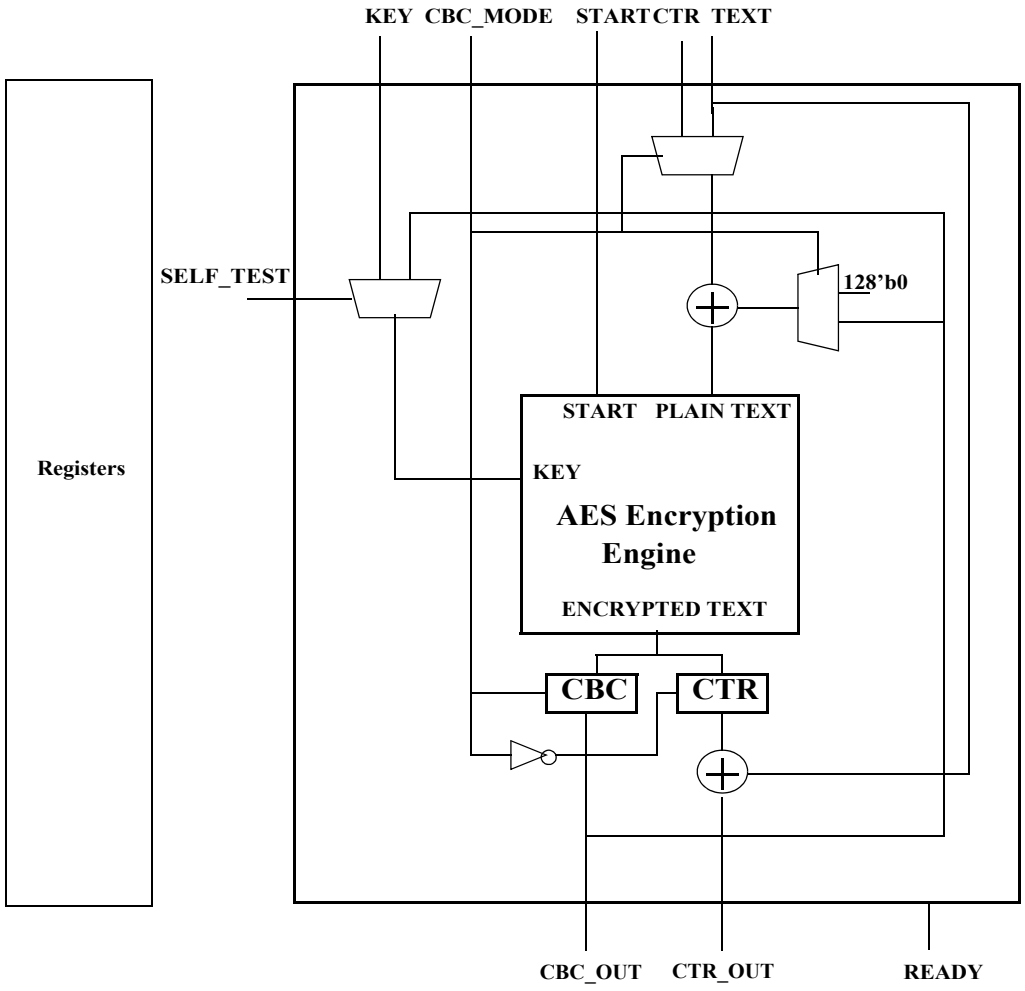



**Figure 10-1. ASM Module Block Diagram**

## 10.3 Modes of Operation

The ASM is designed to be loaded with data and then started with a self-clearing "start" bit. "Counter" mode encryption requires the following test registers:

- Four 32-bit registers of an encryption key
- Four 32-bit registers of a counter
- Four 32-bit registers of text.

Cipher Block Chaining (CBC) mode needs only the encryption key and text.

Typically, only the text fields and counter fields need to be continuously written since the key field won't change.

### 10.3.1 Self-Test Mode

The ASM module has a built-in self-test which requires 3330 clocks to complete.

- This test must be initiated by the software to make the module usable. Until this test is run and is passed, the ASM module is disabled.
- Typical usage would require that this test be run once when the module is powered up.
- The software can re-run this test at any time it becomes necessary to verify the operation of the encryption engine.

To use self-test:

1. Self-test mode is enabled by writing a "1" to the SELF_TEST Bit in Control 1 Register.
2. Self-test is initiated by writing a "1" to the START Bit in Control 0 Register.
3. After self_test is run (3330 clocks) -
   — If self test passes, the TEST_PASS Bit in the Status Register is set. The TEST_PASS bit remains set until a system reset, there is no means to clear it.
   — If self-test fails, the TEST_PASS Bit is cleared to a low, and the ASM module will be disabled and all outputs will be held to constant 0.
4. After using self-test, write a "0" to the SELF_TEST Bit to clear this mode.

### 10.3.2 CTR Mode

Counter mode encryption (CTR) is done by using the AES engine as shown in Figure 10-2. To use CTR mode:

- The 128-bit key is written to the Encryption Key Registers, the 128-bit counter value is written to the Counter Registers, and the 128-bit data to encrypt is written to the Data Registers.
- The encryption is run (13 clocks required)
- The encrypted data can be read from the Counter Results Register.

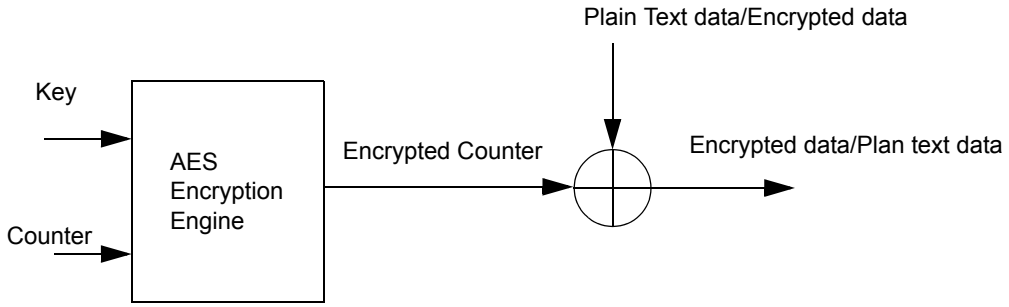



**Figure 10-2. CTR Block**

Decryption also uses the same AES encryption engine. The data to be decrypted is written to the Data Registers. The decrypted result can be read from the Counter Results Register.

### 10.3.3 CBC Mode

The cypher block chaining message authentication code (CBC-MAC) generation is done by using the AES engine as shown in Figure 10-3.

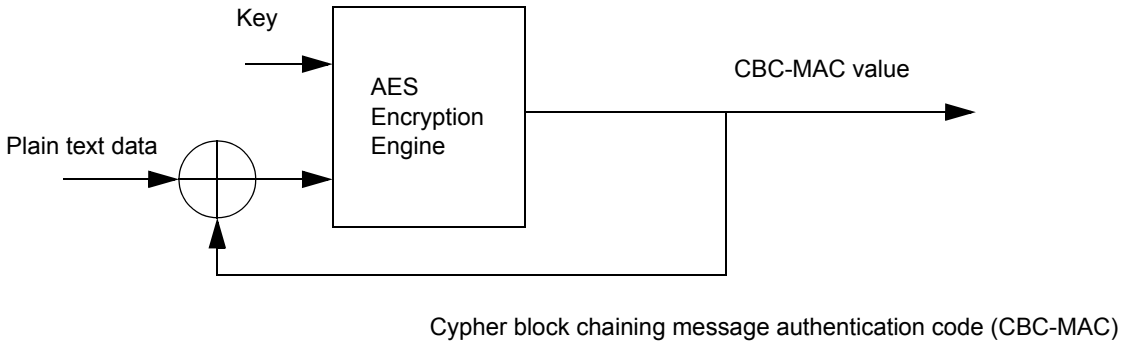



**Figure 10-3. CBC Block**

To use CBC-MAC mode:

- The 128-bit key is written to the Encryption Key Registers and the 128-bit data to encrypt is written to the Data Registers.
- The encryption is run (13 clocks required)
- The encrypted data can be read from the CBC Results Register.

### 10.3.4 CCM Mode (combined CTR and CBC-MAC)

CCM mode is the combination of using CTR mode for protecting the privacy of data, and using CBC mode to generate a MAC to protect the data from unauthorized modifications.

### 10.3.5 Boot Mode

Boot mode is the ASM default mode when it is released from reset.

- In boot mode the key value is set by an internal secret key.
- The boot ROM can then use the ASM module to decrypt or encrypt external ROM. When the boot ROM has completed this task it can then force the ASM module to normal mode by writing to the normal bit in the control register.
- It is not possible for the software to change the mode back to boot mode.
- While the module is in boot mode, the CCM modes (CTR and CBC) can be used for protecting the privacy of the external ROM.

### 10.3.6 Bypass mode

The ASM module can be put in bypass mode by setting the bypass bit in the control register. In bypass mode the data is passed through the module un-encrypted.

## 10.4 AES Encryption Engine Algorithm

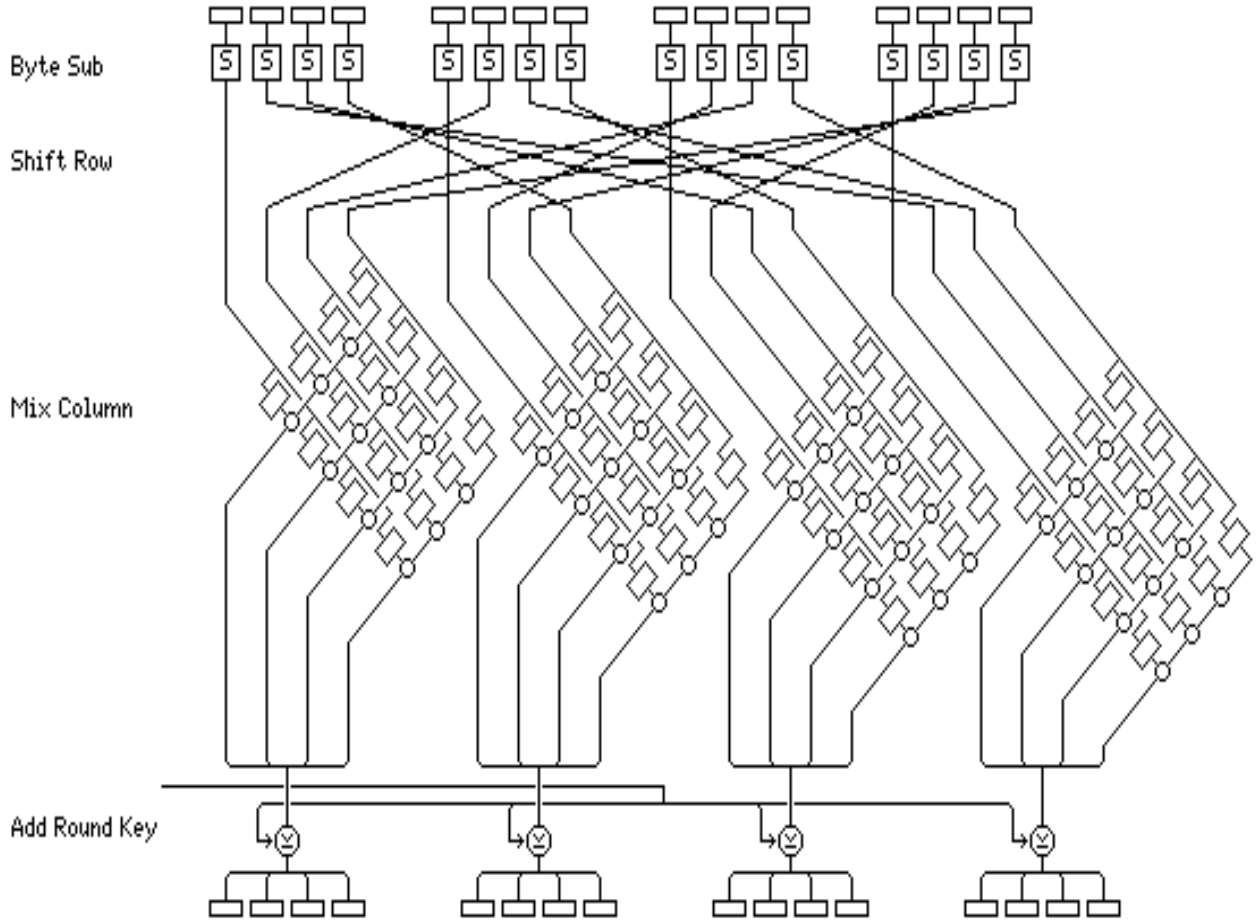



**Figure 10-4. AES Encryption**

The Figure 10-4 diagram documents the encryption process for a single round. Each 128-bit data block is iterated though this structure 10 times. Each time though, the round key is updated with a new value.

#### 10.4.0.1 Byte Substitution

This first part performs a non-linear byte substitution. Details of how this table is constructed can be found in Section 5.1.1 of FIPS-197 available at the NIST web site.

#### 10.4.0.2 Shift Row Transformation

This section performs cyclic shifts at the byte level. Details of this operation can be found in Section 5.1.2 of the FIPS-197 specification available at the NIST web site.

#### 10.4.0.3 Mix Column

This section performs column mixing at the byte level. Details of this operation can be found in Section 5.1.3 of the FIPS-197 specification available at the NIST web site.

#### 10.4.0.4 Add Round Key

This section adds the round key. This is performed by an EXOR with the current round key with the data coming from the previous step ( mix column). Details of this operation can be found in Section 5.1.4 of the FIPS-197 specification available at the NIST web site.

#### 10.4.0.5 Key Expansion

The key value programmed by the software is used to generate a set of 10 keys. Each one is 128 bits. The details of this operation can be found in Section 5.2 of the FIPS-197 specification.

## 10.5 ASM Interrupt Request

The ASM module has a single interrupt request signal that is connected to the interrupt controller. The ASM IRQ can be generated from only a single source.

Table 10-1 lists the ASM interrupt source and characteristics.

**Table 10-1. ASM Interrupt Source**

| Status Bit (ASM_Status Reg) | Mask Bit (ASM_Control1 Reg) | Source Description | Interrupt Clear Mechanism |
|---|---|---|---|
| DONE | MASK_IRQ | Program operation is complete | Writing a 1 to the CLEAR_IRQ bit, ASM_Control0 Register clears the ASM interrupt request (DONE status bit) |

## 10.6 ASM Register Memory Map

The ASM module is programmed via a set of memory-mapped registers and their respective offset addresses are listed in Table 10-2.

- The ADC base address is 0x8000_8000
- All registers are 32 bits wide with 32-bit access only

**Table 10-2. ASM Module Registers Memory Map**

| Address | Register Name | Access Type | Access Width |
|---|---|---|---|
| Base + 0x00 | 128-BIT ENCRYPTION KEY (1 of 4) (ASM_KEY0) | R/W | 32-bit only |
| Base + 0x04 | 128-BIT ENCRYPTION KEY (2 of 4) (ASM_KEY1) | R/W | 32-bit only |
| Base + 0x08 | 128-BIT ENCRYPTION KEY (3 of 4) (ASM_KEY2) | R/W | 32-bit only |
| Base + 0x0C | 128-BIT ENCRYPTION KEY (4 of 4) (ASM_KEY3) | R/W | 32-bit only |
| Base + 0x10 | 128-BIT DATA Register (1 of 4) (ASM_DATA0) | R/W | 32-bit only |
| Base + 0x14 | 128-BIT DATA Register (2 of 4) (ASM_DATA1) | R/W | 32-bit only |
| Base + 0x18 | 128-BIT DATA Register (3 of 4) (ASM_DATA2) | R/W | 32-bit only |
| Base + 0x1C | 128-BIT DATA Register (4 of 4) (ASM_DATA3) | R/W | 32-bit only |
| Base + 0x20 | 128-BIT COUNTER Register (1 of 4) (ASM_CTR0) | R/W | 32-bit only |
| Base + 0x24 | 128-BIT COUNTER Register (2 of 4) (ASM_CTR1) | R/W | 32-bit only |
| Base + 0x28 | 128-BIT COUNTER Register (3 of 4) (ASM_CTR2) | R/W | 32-bit only |
| Base + 0x2C | 128-BIT COUNTER Register (4 of 4) (ASM_CTR3) | R/W | 32-bit only |

**Table 10-2. ASM Module Registers Memory Map**

| | | | |
|---|---|---|---|
| Base + 0x30 | 128-BIT COUNTER RESULT Register (1 of 4) (ASM_CTR0_RESULT) | R | 32-bit only |
| Base + 0x34 | 128-BIT COUNTER RESULT Register (2 of 4) (ASM_CTR1_RESULT) | R | 32-bit only |
| Base + 0x38 | 128-BIT COUNTER RESULT Register (3 of 4) (ASM_CTR2_RESULT) | R | 32-bit only |
| Base + 0x3C | 128-BIT COUNTER RESULT Register (4 of 4) (ASM_CTR3_RESULT) | R | 32-bit only |
| Base + 0x40 | 128-BIT CBC MAC RESULT Register (1 of 4) (ASM_CBC0_RESULT) | R | 32-bit only |
| Base + 0x44 | 128-BIT CBC MAC RESULT Register (2 of 4) (ASM_CBC1_RESULT) | R | 32-bit only |
| Base + 0x48 | 128-BIT CBC MAC RESULT Register (3 of 4) (ASM_CBC2_RESULT) | R | 32-bit only |
| Base + 0x4C | 128-BIT CBC MAC RESULT Register (4 of 4) (ASM_CBC3_RESULT) | R | 32-bit only |
| Base + 0x50 | CONTROL 0 Register (Self Clearing) (ASM_CONTROL0) | W | 32-bit only |
| Base + 0x54 | CONTROL 1 Register (ASM_CONTROL1) | R/W | 32-bit only |
| Base + 0x58 | STATUS Register (ASM_STATUS) | R | 32-bit only |
| Base + 0x5C | Reserved | | |
| Base + 0x60 | 128-BIT CBC MAC Register (1 of 4) (ASM_MAC0) | R/W | 32-bit only |
| Base + 0x64 | 128-BIT CBC MAC Register (1 of 4) (ASM_MAC1) | R/W | 32-bit only |
| Base + 0x68 | 128-BIT CBC MAC Register (1 of 4) (ASM_MAC2) | R/W | 32-bit only |
| Base + 0x6C | 128-BIT CBC MAC Register (1 of 4) (ASM_MAC3) | R/W | 32-bit only |

## 10.7 ASM Registers

The following sections provide descriptions of the ASM registers.

**NOTE**

- The ASM uses a number of 128-bit data variables. These 128-bit data are provided as four 32-bit register values for each variable where the lowest address register contains the least significant 32 bits and the highest address register contains the most significant 32 bits. The intervening two registers are the remaining two 32-bit segments in ascending order.
- The byte ordering for the register does not map directly to the RAM byte ordering. When moving data between memory and the ASM registers, the byte order must be reversed.

### 10.7.1 ASM 128-Bit Encryption Key Registers (ASM_KEY3:ASM_KEY0)

The four Encryption Key Registers contain the 128-bit key used for encryption and include:

- ASM_KEY0 (least significant)- Address Base + 0x00
- ASM_KEY1 - Address Base + 0x04
- ASM_KEY2 - Address Base + 0x08
- ASM_KEY3 (most significant) - Address Base + 0x0C

| ASM_KEYX | | | | | | | | | | | | | Addr Base+0x0X | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | KET_DATA[31:16] | | | | | | | | | | | | | | | |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | KEY_DATA[15:0] | | | | | | | | | | | | | | | |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 10.7.2 ASM 128-Bit Data Registers (ASM_DATA3:ASM_DATA0)

The four Data Registers contain the 128-bit data value that will be either encrypted or decrypted and include:

- ASM_DATA0 (least significant)- Address Base + 0x10
- ASM_DATA1 - Address Base + 0x14
- ASM_DATA2 - Address Base + 0x18
- ASM_DATA3 (most significant) - Address Base + 0x1C

| ASM_DATAX | | | | | | | | | | | | | Addr Base+0x1X | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | DATA[31:16] | | | | | | | | | | | | | | | |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | DATA[15:0] | | | | | | | | | | | | | | | |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 10.7.3 ASM 128-Bit Counter Registers (ASM_CTR3:ASM_CTR0)

The four Counter Registers contain the 128-bit counter value used to encrypt the data in counter mode and include:

- ASM_CTR0 (least significant)- Address Base + 0x20
- ASM_CTR1 - Address Base + 0x24
- ASM_CTR2 - Address Base + 0x28

- ASM_CTR3 (most significant) - Address Base + 0x2C

| ASM_CTRX | | | | | | | | | | | | | Addr Base+0x2X | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | CTR_DATA[31:16] | | | | | | | | | | | | | | | |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | CTR_DATA[15:0] | | | | | | | | | | | | | | | |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 10.7.4 ASM 128-Bit Counter Result Registers (ASM_CTR3_RESULT: ASM_CTR0_RESULT)

The four Counter Result Registers contain the 128-bit counter result value when the encryption is done, and include:

- ASM_CTR0_RESULT (least significant)- Address Base + 0x30
- ASM_CTR1_RESULT - Address Base + 0x34
- ASM_CTR2_RESULT - Address Base + 0x38
- ASM_CTR3_RESULT (most significant) - Address Base + 0x3C

| ASM_CTRX_RESULT | | | | | | | | | | | | | Addr Base+0x3X | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | CTR3_RESULT_DATA[31:16] | | | | | | | | | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | CTR3_RESULT_DATA[15:0] | | | | | | | | | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 10.7.5 ASM 128-Bit CBC MAC Result Registers (ASM_CBC3_RESULT: ASM_CBC0_RESULT)

When CBC-MAC generation is done, the result can be read from this register.

The four CBC-MAC Result Registers contain the 128-bit result value when the CBC-MAC generation is done, and include:

- ASM_CBC0_RESULT (least significant)- Address Base + 0x40
- ASM_CBC1_RESULT - Address Base + 0x44
- ASM_CBC2_RESULT - Address Base + 0x48
- ASM_CBC3_RESULT (most significant) - Address Base + 0x4C

| **ASM_CBCX_RESULT** | | | | **CBC Result** | | | | | | | | | **Addr Base+0x4X** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | CBC_RESULT_DATA[31:16] | | | | | | | | | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | CBC_RESULT_DATA[15:0] | | | | | | | | | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 10.7.6 ASM Control 0 Register (ASM_CONTROL0)

The Control 0 Register provides control functions of MAC preload, clear IRQ, clear memory, and start encryption to the ASM module.

- The bits are self-clearing
- Writing a “1” initiates the control function

| ASM_CONTROL0 | | | | | | | | | | | | | Addr Base+0x50 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | CLEAR_IRQ | Reserved | | | | LOAD_MAC | CLEAR | START | Reserved | | | | | | | |
| TYPE | w | r | r | r | r | w | w | w | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | Reserved | | | | | | | | | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 10-3. ASM_CONTROL0 Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31 | **CLEAR_IRQ -** Clear Interrupt Request bit. Writing a 1 to this bit clears the ASM interrupt request (DONE status bit) | Reads as 0 |
| 30-27 | **Reserved** | Read only |
| 26 | **LOAD_MAC -** Load MAC bit. Writing a 1 to this bit pre-loads the MAC with the 128-bit value in the CBC MAC Registers | Reads as 0 |
| 25 | **CLEAR -** Clear Memory bit. Writing a 1 to this bit clears memory in the ASM. | Reads as 0 |
| 24 | **START -** Start bit. Writing a 1 to this bit initiates operation of programmed mode:<br>• CBC encryption mode<br>• CTR encryption mode<br>• Dual mode encryption<br>• Self-test | Reads as 0 |
| 23-0 | **Reserved** | Read only |

## 10.7.7 ASM Control 1 Register (ASM_CONTROL1)

The Control 1 Register provides additional control functions for the ASM module. These bits do not initiate an execution and are not self-clearing.

**NOTE**

Two bits in ASM_CONTROL1 enable Counter encryption mode and/or CBC-MAC encryption mode.

- CTR mode requires 13 clocks
- CBC-MAC mode requires 13 clocks
- If both are set (CTR and CBC-MAC), encryption requires 26 clocks.

| **ASM_CONTROL1** | | | | | | | | | | | | | **Addr Base+0x54** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | MASK_IRQ | | | | | SELF_TEST | CTR | CBC | | | | | | | | |
| TYPE | rw | r | r | r | r | rw | rw | rw | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | | | | | | | | | | | | | | BYPASS | NORMAL_MODE | ON |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 10-4. ASM_CONTROL1 Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31 | **MASK_IRQ -** Mask Interrupt Request bit. Writing a 1 to this bit disables the ASM IRQ | 1 = IRQ disabled<br>0 = IRQ enabled (default) |
| 30-27 | **Reserved** | Read only |
| 26 | **SELF_TEST -** Self-test bit. Writing a 1 to this bit puts the ASM in self-test mode | 1 = ASM Module in self-test mode<br>0 = ASM Module not in self-test (default) |
| 25 | **CTR -** Counter mode bit. Writing a 1 to this bit enables counter encryption mode. | 1 = Counter encryption mode enabled<br>0 = Counter encryption mode disabled (default) |
| 24 | **CBC -** CBC-MAC mode bit. Writing a 1 to this bit enables CBC-MAC encryption mode. | 1 = CBC-MAC encryption mode enabled<br>0 = CBC-MAC encryption mode disabled (default) |

| Bit Number | Description | Operation |
|---|---|---|
| 23-3 | **Reserved** | Read only |
| 2 | **BYPASS -** Bypass mode bit. Writing a 1 to this bit enables ASM bypass mode. | 1 = Bypass mode enabled. Data is passed through unmodified.<br>0 = Data will be encrypted. (default) |
| 1 | **NORMAL_MODE -** Normal mode bit. Write a 1 to this bit puts ASM in normal mode.<br>• By default the ASM module is in "boot" mode. In "boot" mode an internal secret key is used for all operations. This is to allow the software to use the ASM module to encrypt/decrypt the external ROM contents.<br>• Writing a "1" to the normal mode bit will change the mode from "boot" to "normal" mode. The key value can then be set by writing to the proper register.<br>• It is not possible for software to enter "boot" mode. The only way to return to "boot" is with a system reset. The boot ROM will then decide when to change mode from "boot" to "normal". | 1 = Enables “normal” operation after startup from a reset condition.<br>**Note:** This bit cannot be re-written to 0 after having been written to 1.<br>0 = Default; out of reset ASM “boot” mode is active. Writing a 1 disables this mode. |
| 0 | **ON -** ASM module enable bit. Writing a 1 to this bit enables the ASM module. | 1 = ASM module enabled<br>0 = ASM module is disabled (default) |

## 10.7.8 ASM Status Register (ASM_STATUS)

The Status Register provides test-pass and done status for the ASM.

| **ASM_STATUS** | | | | | | | | | | | | | **Addr Base+0x58** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | Reserved | | | | | | TEST PASS | DONE | Reserved | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | r | r | r | r | r | r | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | Reserved | | | | | | | | | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 10-5. ASM_STATUS Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-26 | **Reserved** | Read only |
| 25 | **TEST_PASS -** Test pass bit. If this bit is set to 1 after performing self-test, the result was successful.<br>• If self test fails then the module is not usable and the outputs will be set to all "0"<br>• The software must initiate a self test before the module can be used.<br>• Once set, the bit remains set until a system reset | 1 = Self-test has passed<br>0 = Self-test has failed |
| 26 | **DONE -** Done bit. This bit is set to 1 after an operation, and results registers can be read | 1 = Function complete. The result in the CTR or CBC register is ready to be read.<br>0 = The result in the CTR or CBC register is not ready to be read. |
| 23-0 | **Reserved** | Read only |

## 10.7.9 ASM 128-BIT CBC MAC Registers (ASM_MAC3:ASM_MAC0)

The four CBC MAC Registers contain the 128-bit start value used for CBC-MAC encryption and include:

- ASM_MAC0 (least significant) - Address Base + 0x60
- ASM_MAC1 - Address Base + 0x64
- ASM_MAC2 - Address Base + 0x68
- ASM_MAC3 (most significant) - Address Base + 0x6C

**NOTE**

The MAC start value is only used if loaded via the LOAD_MAC bit in Control 0 Register.

| **ASM_MACX** | | | | | | | | | | | | | **Addr Base+0x6X** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | MAC_START_DATA[31:16] | | | | | | | | | | | | | | | |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | MAC_START_DATA[15:0] | | | | | | | | | | | | | | | |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 10.8 ASM Use Models

The following sections provide very basic guidance on using the different modes of the ASM.

**NOTE**

Use of the ASM module to help implement the IEEE 802.15.4 Standard security options is beyond the scope of this document. However, Freescale includes implementation of the 802.15.4 security in its 802.15.4 MAC software and also provides ASM utilities in its MC1322x Simple MAC (SMAC) software. For more information, see the Freescale BeeKit Wireless Connectivity Toolkit and its documentation.

### 10.8.1 Counter Mode Encryption

The following is a typical procedure for using the counter (CTR) mode encryption which requires 13 clocks to complete once the START Bit is written:

1. Program the key value in Encryption Key Registers (Offset 0x00, 0x04, 0x08, and 0x0C)
2. Set the bit CTR Bit in Control 1 Register (Offset 0x54). The CTR Bit is static.
3. Program the text value in Data Registers (Offset 0x10, 0x14, 0x18, and 0x1C)
4. Program the counter value in Counter Registers (Offset 0x20, 0x24, 0x28, and 0x2C)
5. Set the START Bit, Control 0 Register (Offset 0x50). (This bit is a self-clearing.)
6. Wait for completion
   - — Poll for the DONE Bit, Status Register (Offset 0x58) to be set to a "1", or
   - — Wait for the ASM interrupt request.
7. Read the encrypted text from Counter Results Registers (Offset 0x30, 0x34, 0x38, and 0x3C)
8. Repeat Steps 3-7 for each 128-bit block.

**NOTE**

Decryption is a similar procedure. The same key and count values are used. The encrypted data is entered in Data Registers and the decrypted data is read from the Counter Results Registers using the about procedure.

### 10.8.2 Message Authentication Code Generation (MAC)

The following is a typical procedure for doing MAC generation which requires 13 clocks to complete each time the START Bit is written:

1. Program the key value in Encryption Key Registers (Offset 0x00, 0x04, 0x08, and 0x0C)
2. Set the bit CBC Bit in Control 1 Register (Offset 0x54). The CBC Bit is static.
3. Program the text value in Data Registers (Offset 0x10, 0x14, 0x18, and 0x1C)
4. Set the START Bit, Control 0 Register (Offset 0x50). (This bit is a self-clearing.)
5. Wait for completion
   - — Poll for the DONE Bit, Status Register (Offset 0x58) to be set to a "1", or

— Wait for the ASM interrupt request.

6. Repeat steps 3-6 for each 128 bits block.
7. When all blocks have been processed, read the final MAC (authentication code) from MAC Result Registers (Offset 0x40, 0x44, 0x48, and 0x4C).
8. The MAC accumulator should be cleared to prepare for the next payload.
   — Writing a 1 to the CLEAR Bit, Control 0 Register, will set the MAC accumulator back to all 0. CLEAR Bit is self clearing.
   — If for some reason the software needs to continue a MAC calculation from a previously saved value, the MAC accumulator can be pre-loaded with a initial value.
      – The initial value for the accumulator is written to CBC MAC Registers (Offset 0x60, 0x64, 0x68, and 0x6C) and then a 1 is written to the LOAD_MAC Bit, Control 0 Register. The LOAD_MAC Bit is self clearing.

### 10.8.3 Counter Mode and Authentication Mode Encryption Combined

The following is a typical procedure for doing CTR and authentication mode encryption combined which requires 13 clocks to complete each time the START Bit is written:

1. Program the key value in Encryption Key Registers (Offset 0x00, 0x04, 0x08, and 0x0C)
2. Set the bit CBC Bit in Control 1 Register (Offset 0x54). The CBC Bit is static.
3. Program the text value in Data Registers (Offset 0x10, 0x14, 0x18, and 0x1C)
4. Program the counter value in Counter Registers (Offset 0x20, 0x24, 0x28, and 0x2C)
5. Set the START Bit, Control 0 Register (Offset 0x50). (This bit is a self-clearing.)
6. Wait for completion
   — Poll for the DONE Bit, Status Register (Offset 0x58) to be set to a "1", or
   — Wait for the ASM interrupt request..
7. Read the encrypted text from Counter Results Registers (Offset 0x30, 0x34, 0x38, and 0x3C).
8. Repeat steps 3-7 for each 128-bit block.
9. When all blocks have been processed, read the final MAC (authentication code) from MAC Result Registers (Offset 0x40, 0x44, 0x48, and 0x4C).

# Chapter 11
# General Purpose I/O Module

## 11.1 Introduction

The MC1322x has a possible 64 GPIOs. Many of these pins are shared with on-chip peripherals such as the timer, external interrupts, or communication channels. When these other modules or functions are not using these pins, they can be configured as general-purpose I/O control. Each I/O port can be configured as an input or output, has programmable pull-up or pull-down resistors, and has complete read and write capability.

## 11.2 Features

The MC1322x GPIO features include:

- 64 General-Purpose IO pins
- Default to GPIO controlled as inputs with pull-down or pull-up resistors enabled (except for JTAG/NEXUS and some analog pins)
- Programmable input hysteresis
- Software controlled pull-ups or pull-downs on all pins
- Software controlled state keepers on all pins
- Logic levels specified at +/-1 mA drive current drive and 15pF loading for ac performance
- Logic levels specified at +/-5 mA drive current drive for higher low frequency loads such as LEDs
- Shared functionality with MCU peripherals or special functions
- The GPIO module/control generates no interrupts
- Eight “stay-alive” KBI signal for use during low power modes.

## 11.3 Pin Descriptions

The detailed signal name and pin assignment information for the MC1322x 64 signals that can be used as parallel I/O pins is displayed in Chapter 2, “Pins and Connections”. All of these pins are available for general-purpose I/O when they are not used by another on-chip peripheral or function.

The 64 GPIO pins have alternative functions as follows:

- 4 JTAG I/Os reconfigurable as GPIO
- 22 GPIOs reconfigurable as interfaces to the peripherals
- 8 dedicated KBI GPIOs, 4 of which are usable as keyboard interrupts (KBIs) or external wake-up signals to the CRM
- 8 GPIOs reconfigurable as ADC analog inputs

- 4 control pins for external RF RX/TX control
- 14 Nexus pins (JTAG also)

Table 11-1 lists the shared functionality of the pins and where the functions are described.

**Table 11-1. GPIO Shared Functionality**

| GPIO Pins | Alternate Function | Reference[1] |
|---|---|---|
| GPIO3-GPIO0 | SSI | Chapter 16, “Synchronous Serial Interface (SSI)” |
| GPIO7-GPIO4 | SPI | Chapter 15, “Serial Peripheral Interface Module (SPI)” |
| GPIO11-GPIO8 | TIMER | Chapter 12, “Timer Module (TMR)” |
| GPIO13-GPIO12 | $I^2C$ | Chapter 14, “I2C Module” |
| GPIO21-GPIO14 | UART | Chapter 13, “Universal Asynchronous Receiver/Transmitter Module (UART)” |
| GPIO29-GPIO22 | KBI | Chapter 5, “System Management (Including CRM)” |
| GPIO41-GPIO30 | ADC | Chapter 17, “Analog to Digital Converter Module (ADC)” |
| GPIO45-GPIO42 | EXT RF CTL | Chapter 6, “MC1322x IEEE 802.15.4 2.4 GHz ISM Band Transceiver” |
| GPIO63-GPIO46 | JTAG & Nexus | Chapter 18, “Development and Debug Support” |

[1] See this section for information about modules that share these pins.

## 11.4 Functional Description

Each input/output port is a single bit and is easily configured. It consists of all the control and data logic in the GPIO module port and any peripheral control associated with that bit. Figure 11-1 shows the typical block diagram of a single bit configuration.

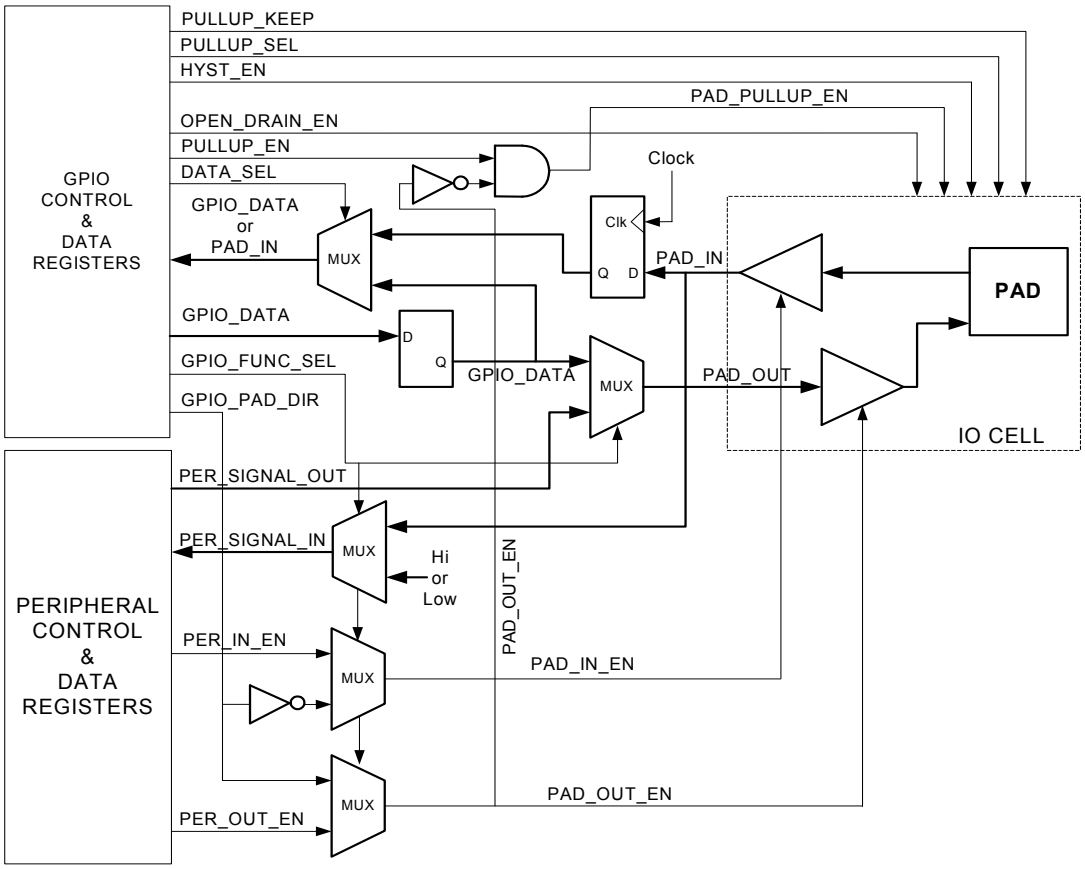



**Figure 11-1. GPIO Typical Single-Bit Block Diagram.**

Each single-bit port has a set of data and control register bits that determine the operation of that port. The GPIO data and control bits are organized such that all similar control or data are in a single register type. For 64 ports, two 32-bit wide registers of each type are typically used. An exception is the Function Select control which uses four registers total because the each select field is 2-bits wide. The GPIO register types are shown in Table 11-2.

**Table 11-2. GPIO Register Types**

| Register Type | Function | Access Type |
|---|---|---|
| GPIO FUNCTION SELECT | Function select (1 of 4 possible) | R/W |
| GPIO PAD DIRECTION | Set pad direction only for GPIO modes | R/W |
| GPIO PAD DIRECTION SET | Used with present value of GPIO PAD DIRECTION register to set new bit values in GPIO PAD DIRECTION | W-only |
| GPIO PAD DIRECTION RESET | Used with present value of GPIO PAD DIRECTION register to reset new bit values in GPIO PAD DIRECTION | W-only |
| GPIO DATA | Data register for GPIO modes only | R/W |
| GPIO DATA SET | Used with present value of GPIO DATA register to set new bit values in DATA REGISTER | W-only |

**Table 11-2. GPIO Register Types (continued)**

| Register Type | Function | Access Type |
|---|---|---|
| GPIO DATA RESET | Used with present value of GPIO DATA register to reset new bit values in DATA REGISTER | W-only |
| PAD PULLUP ENABLE | Pad pull-up (pull-down) enable | R/W |
| PAD PULLUP SELECT | Select between a pull-up and a pull-down when pull-up is enabled | R/W |
| READ DATA CONTROL | Selects between GPIO DATA register and input pad for read data | R/W |
| PAD HYSTERESIS ENABLE | Enable input pad hysteresis | R/W |
| PAD KEEPER | Pad keeper enable | R/W |

The GPIO block has overall control of the pads. However, the mode control will determine whether the pad is used with GPIO or a peripheral function. When the peripheral function is selected, that peripheral controls pad enables and data direction.

## 11.4.1 GPIO Function Select

The GPIO module has four registers dedicated to providing a 2-bit function select field for each GPIO (see Table 11-11). These control the four modes of operation: Normal Mode (default), Alternate1 Mode, Alternate2 Mode and Alternate3 Mode. Table 11-3 lists the GPIO functionality versus function select for the various categories of alternative or peripheral functions.

**NOTE**

The default modes shown in Table 11-3 occur upon exit of POR. These can be altered during the boot process, see Section 3.12.7, "GPIO Function Upon Boot Exit".

**Table 11-3. General GPIO Function vs. Function Select**

| Category | Alternative Function | Function 0 [00] (Default) | Function 1 [01] Alternate 1 | Function 2 [10] Alternate 2 | Function 3 [11] Alternate 3 |
|---|---|---|---|---|---|
| 1 | SSI | GPIO | SSI | GPIO | GPIO |
| 2 | SPI | GPIO | SPI | GPIO | GPIO |
| 3 | TIMER | GPIO | TIMER | GPIO | GPIO |
| 4 | $I^2C$ | GPIO | $I^2C$ | GPIO | GPIO |
| 5 | UART | GPIO | UART | GPIO | GPIO |
| 6 | KBI | GPIO/KBI_interrupt | GPIO/KBI_interrupt | GPIO/ KBI_interrupt | GPIO/ KBI_interrupt |
| 7 | ADC | GPIO except for ADC1 VREF pins and JTAG RTCK | ADC (analog) except ADC1 VREF H/L; GPIO for JTAG RTCK | GPIO | GPIO |

**Table 11-3. General GPIO Function vs. Function Select (continued)**

| Category | Alternative Function | Function 0 [00] (Default) | Function 1 [01] Alternate 1 | Function 2 [10] Alternate 2 | Function 3 [11] Alternate 3 |
|---|---|---|---|---|---|
| 8 | EXT RF CTL | GPIO | TX control | RX control | GPIO |
| 9 | JTAG & Nexus | JTAG & Nexus | GPIO | GPIO | GPIO |

The GPIO module always controls the selection and choice of pull-up or pull-down resistors and use of pad state keepers.

#### 11.4.1.1 Normal (Function 0) Mode

Function 0 mode is the POR default mode which puts all IO to GPIO mode except the JTAG/NEXUS pins and the ADC1_VREF pins. For default:

- On exiting reset, all the GPIO except JTAG/NEXUS pins and the ADC1_VREF pins are set to inputs with input pull-down resistors enabled
- For the JTAG/NEXUS and ADC1_VREF pins, Function 0 mode is the normal non-GPIO functionality of the pins. For the JTAG/NEXUS input pins, the default is to have weak pull-ups enabled instead of pull-downs. Commonly, the JTAG or Nexus functionality is not used in the target application and the initialization of the MC1322x should reconfigure these pins.
- In GPIO Mode, the GPIO module
  - Controls the pad direction (input or output)
  - Provides the output data when driving the pad
  - Selects the source of data (pad or register) when reading the port

#### 11.4.1.2 Alternate Modes

The function select provides three alternate modes, however they are highly redundant.

- For all GPIOs except the JTAG/NEXUS and ADC1_VREF pins, Function 1 mode can be thought of as “Peripheral Controlled Mode.” The peripheral function will control operation of the pad IF THE PERIPHERAL IS ENABLED.
- When enabled the ADC module puts its pads into analog mode and the digital pad function is totally disabled
- The external RF control (EXT RF CTL) has different functionality in Function 1 versus Function 2
- The peripheral module supplies the output enable to the I/O pad to control its direction, and the pad output data if its output enable is asserted. The peripheral can read the value of the data on the I/O pad if it is using the pin as an input.

### 11.4.2 Pull-up/Pull-down Resistors

Each GPIO pad has a selectable pull-up/pull-down resistor.

- The use of a pull-up/pull-down is selectable via the PAD PULLUP ENABLE registers
- The use of a pull-up versus a pull-down is selectable via the PAD PULLUP SELECT registers
- The pull-up/pull-down is only enabled when the pad is an input. It is disabled when the pad is programmed as an output.
- For a peripheral function, the pull-up/pull-down is selected via the GPIO module. If selected the pull-up/pull-down is active for a GPIO or peripheral input
- The POR GPIO default is as pull-down except for as noted in Table 11-14.
- Typical values are 50 Kohms for pull-ups and pull-downs

Table 11-4 shows the truth table for the pull-up/pull-down enable function.

**Table 11-4. GPIO Pull-up/Pull-down Configuration**

| Pull-up En | Pad Output Enable | Pull-ups |
|---|---|---|
| 0 | 0 | disabled |
| 1 | 0 | enabled |
| X | 1 | disabled |

### 11.4.3 Input Hysteresis

Each GPIO pad has a selectable input hysteresis function that is available when the pad is programmed as an input

- The use of hysteresis is selectable via the PAD HYSTERESIS ENABLE registers
- For a peripheral function, the input hysteresis is also selected via the GPIO module. If selected the hysteresis is active for a GPIO or peripheral input
- The POR GPIO default is with hysteresis disabled.

### 11.4.4 Pad Keeper Function

Each GPIO pad has a selectable pad keeper function

- The pad keeper will retain the last state held on the pad (when used as an input). The pad keeper is disabled if the pad is programmed as an output.
- The use of the pad keeper is selectable via the PAD KEEPER registers
- For a peripheral function, the pad keeper is selected via the GPIO module. If selected the keeper is active for the GPIO or peripheral input
- The POR GPIO default is with pad keeper disabled.

### 11.4.5 Port I/O Control

The GPIO module has primary control of the IO port. As stated in the previous sections the pull-up/pull-down, hysteresis, and pad keeper functions are controlled from the GPIO module independent of the function select state (operating mode). When GPIO mode is enabled the port I/O control uses the following functionality

#### 11.4.5.1 GPIO Pad Direction Control

Each GPIO pad has a pad direction control

- The pad direction is determined by the GPIO PAD DIRECTION registers
- The direction defaults to inputs enabled
- The PAD DIRECTION registers can be accessed directly for read and write.
- The PAD DIRECTION registers can also be modified to set or reset specific IO
  - For each PAD DIRECTION register, there is a write-only GPIO PAD DIRECTION SET register. Writing a “1” to any bit in that register will set the corresponding bit in the PAD DIRECTION register and retain all other 1’s previously set in the PAD DIRECTION register.
  - For each PAD DIRECTION register, there is a write-only GPIO PAD DIRECTION RESET register. Writing a “1” to any bit in that register will reset (set to “0”) the corresponding bit in the PAD DIRECTION register and retain all other 0’s previously set in the PAD DIRECTION register.
- If the function select for a given IO does not enable GPIO, pad direction reverts to the assigned peripheral function

#### 11.4.5.2 GPIO Data Control

Each GPIO pad has pad data control

- If the pad direction is selected as an output the data state of the pad is driven by the corresponding bit of the GPIO DATA register
  - The GPIO DATA registers default to all zeroes.
  - The GPIO DATA registers can be accessed directly for write.
  - The GPIO DATA registers can also be modified to set or reset specific IO
    - For each GPIO DATA register, there is a write-only GPIO DATA SET register. Writing a “1” to any bit in that register will set the corresponding bit in the GPIO DATA register and retain all other 1’s previously set in the GPIO DATA register.
    - For each GPIO DATA register, there is a write-only GPIO DATA RESET register. Writing a “1” to any bit in that register will reset (set to “0”) the corresponding bit in the GPIO DATA register and retain all other 0’s previously set in the PAD DIRECTION register.
- If the pad direction is selected as an input, data can be read from either the pad or the GPIO DATA register.
  - The corresponding READ DATA CONTROL register will determine for each IO whether an access of the GPIO DATA register will fetch a read of the stored internal DATA REGISTER bit or the pad input register.

— The default condition on the READ DATA CONTROL register is to read the pads.

#### 11.4.5.3 Peripheral Port Control

When not in GPIO mode, the pad direction control comes from the peripheral, and input and output data are interfaced with the peripheral port.

- When in GPIO Mode, the data inputs into the peripheral modules are driven high (except for SSI, whose peripheral inputs need to be 0 for non-peripheral and non-master modes) and the output data enables from the peripherals are ignored.
- As stated previously pull-up/pull-down, hysteresis, and keepers are determined by the GPIO module while the port is in peripheral mode.
- The $I^2C$ port is a special case
  - — Both the input pad and output pad are enabled for $I^2C$ operation
  - — $I^2C$ is the only case when the output pad is enabled for open-drain operation
  - — Pull-ups are always disabled for $I^2C$ pins in $I^2C$ operation

### 11.4.6 Pin Mapping within GPIO Control Registers

The GPIO port pins are bit mapped into the GPIO control registers. Table 11-5 contains a listing of all the GPIOs by port number, corresponding MC1322x pin name, pin number, and functional modes.

**Table 11-5. MC1322x GPIO Register Bit Mapping vs. Peripheral Function**

| Register X | | | Register X+4 | | |
|---|---|---|---|---|---|
| **Bit** | **Function** | **Pin** | **Bit** | **Function** | **Pin** |
| 0 | GPIO0 / SSI_TX | 34 | 0 | GPIO32 / ADC2[3] | 3 |
| 1 | GPIO1 / SSI_RX | 33 | 1 | GPIO33 / ADC3[3] | 4 |
| 2 | GPIO2 / SSI_FSYN | 32 | 2 | GPIO34 / ADC4[3] | 5 |
| 3 | GPIO3 / SSI_BITCK | 31 | 3 | GPIO35 / ADC5[3] | 6 |
| 4 | GPIO4 / SPI_SS | 30 | 4 | GPIO36 / ADC6[3] | 7 |
| 5 | GPIO5 / SPI_MISO | 29 | 5 | GPIO37 / ADC7_RTCK[3] | 8 |
| 6 | GPIO6 / SPI_MOSI | 28 | 6 | GPIO38 / ADC2_VREFH[3] | 64 |
| 7 | GPIO7 / SPI_SCK | 27 | 7 | GPIO39 / ADC2_VREFL[3] | 61 |
| 8 | GPIO8 / TMR0 | 26 | 8 | GPIO40 / ADC1_VREFH[3] | 63 |
| 9 | GPIO9 / TMR1 | 25 | 9 | GPIO41 / ADC1_VREFL[3] | 62 |
| 10 | GPIO10 / TMR2 | 24 | 10 | GPIO42 / ANT_1 | 56 |
| 11 | GPIO11 / TMR3 | 23 | 11 | GPIO43 / ANT_2 | 57 |
| 12 | GPIO12 / I2C_SCL[1] | 22 | 12 | GPIO44 / TX_ON | 52 |
| 13 | GPIO13 / I2C_SDA[1] | 21 | 13 | GPIO45 / RX_ON | 59 |
| 14 | GPIO14 / UART1_TX | 20 | 14 | GPIO46 / TMS | 12 |

**Table 11-5. MC1322x GPIO Register Bit Mapping vs. Peripheral Function (continued)**

| Register X | | | Register X+4 | | |
|---|---|---|---|---|---|
| **Bit** | **Function** | **Pin** | **Bit** | **Function** | **Pin** |
| 15 | GPIO15 / UART1_RX | 19 | 15 | GPIO47 / TCK | 11 |
| 16 | GPIO16 / UART1_CTS | 18 | 16 | GPIO48 / TDI | 10 |
| 17 | GPIO17 / UART1_RTS | 17 | 17 | GPIO49 / TDO | 9 |
| 18 | GPIO18 / UART2_TX | 16 | 18 | GPIO50 / MCKO | 131 |
| 19 | GPIO19 / UART2_RX | 15 | 19 | GPIO51 / MDO00 | 103 |
| 20 | GPIO20 / UART2_CTS | 14 | 20 | GPIO52 / MDO01 | 102 |
| 21 | GPIO21 / UART2_RTS | 13 | 21 | GPIO53 / MDO02 | 112 |
| 22 | GPIO22 / KBI_0_HST_WK[2] | 42 | 22 | GPIO54 / MDO03 | 111 |
| 23 | GPIO23 / KBI_1[2] | 41 | 23 | GPIO55 / MDO04 | 121 |
| 24 | GPIO24 / KBI_2[2] | 40 | 24 | GPIO56 / MDO05 | 120 |
| 25 | GPIO25 / KBI_3[2] | 39 | 25 | GPIO57 / MDO06 | 130 |
| 26 | GPIO26 / KBI_4[2] | 38 | 26 | GPIO58 / MDO07 | 129 |
| 27 | GPIO27 / KBI_5[2] | 37 | 27 | GPIO59 / MSEO0_B | 114 |
| 28 | GPIO28 / KBI_6[2] | 36 | 28 | GPIO60 / MSEO1_B | 113 |
| 29 | GPIO29 / KBI_7[2] | 35 | 29 | GPIO61 / RDY_B | 122 |
| 30 | GPIO30 / ADC0[3] | 1 | 30 | GPIO62 / EVTO_B | 123 |
| 31 | GPIO31 / ADC1[3] | 2 | 31 | GPIO63 / EVTI_B | 132 |

1 $I^2C$ pins are true bidirectional with open drain.

2 The 8 KBI pads are always active, even during low-power modes. When KBI_4:KBI_7 are being used as keyboard interrupts, they are controlled by the CRM module well as the GPIO

3 Analog-digital GPIO ports have their digital pads forced to tri-state when in analog functional modes. The digital pad is completely off. Pad keepers must be disabled.

## 11.5 Special Conditions and Signal Usage

Different power modes and special pin functions are discussed in the following sections.

### 11.5.1 Hardware Reset Active (Off Condition)

If the hardware reset is active, the MC1322x is in the lowest power mode, i.e., the reset condition. When input RESETB is asserted low, the device is held in reset and the "off" condition (there is no internal pull-up resistor on the reset input). The only circuitry that remains powered is the POR control. The GPIO buffers have the supply removed from them to eliminate leakage. As a result, no GPIO signal should be tied or driven high while the RESETB is active. This prevents a GPIO from forward biasing the ESD diode on the pad and causing excessive leakage through the IO structure. See Section 3.3, "System Reset and Low Power GPIO Signal Connections"

**NOTE**

If a GPIO is held or driven high during reset, no damage will be done. However, if reset is being used as a very low power state, this defeats that purpose.

The 8 KBI signals designated as KBI7-KBI0 are a special case; they are always powered and are in their reset/low power state.

### 11.5.2 Release From Hardware Reset

On release from reset all GPIO revert to default conditions. The GPIO are commonly forced to inputs with pull-downs activated and pad keepers disabled with the following exceptions (see Table 11-3):

- The JTAG/Nexus debug port signals are active
- The 8 KBI7-KBI0 signals are a special block (see Section 5.1.5, "KBI Interface Signals"). When exiting reset, KBI3-KBI0 are configured as outputs forced to the "1" state under GPIO control. The KBI7-KBI4 are configured as inputs with pull-downs enabled.

**NOTE**

See Section 5.9.2, "Wake-up Control (WU_CNTL)" for detailed information on control and use of the KBI signals.

- The shared analog signals for the ADCs
  - ADC7 defaults to debug port output signal RTCK
  - ADC6-ADC0, ADC2_VREFH, and ADC2_VREFL default to GPIO digital inputs with pull-downs enabled and with pad keepers enabled
  - ADC1_VREFH, and ADC1_VREFL default to their analog function as ADC reference voltages

**NOTE**

When these ports (ADC7-ADC0, ADCx_VREFH, and ADCxVREFL) are used as analog signals the bus keepers must be disabled, and alternately, the bus keepers must be enabled when the ports are used as digital inputs.

Table 11-6 lists the conditions of the GPIO as the MC1322x exits a hardware reset.

**NOTE**

Be aware that the boot process can affect the state of the GPIO and leave a GPIO in an altered state. The impact of the boot process on the GPIO is discussed in Section 3.12.7, "GPIO Function Upon Boot Exit".

**Table 11-6. GPIO Pin Default State**

| Item | Pin Function | Pin Name | Default Function | Reset Active | Default State (Upon exiting reset, before Boot) | Hibernate & Doze |
|---|---|---|---|---|---|---|
| 1 | Standard GPIO[1] | | GPIO | Unpowered | Inputs w / pull-downs enabled and bus keepers disabled | Unpowered |
| 2 | $I^2C$ Bus | I2C_SDA, I2C_SCL | GPIO | Unpowered | Input w / pull-ups enabled and bus keepers disabled | Unpowered |
| 3 | ADC Reference Voltage | ADC1_VREFH, ADC1_VREFL | Analog | Unpowered | Analog Reference Voltage Inputs (no pull-up/pull-down and bus keeper disabled) | Unpowered |
| 4 | ADC Reference Voltage | ADC2_VREFH, ADC2_VREFL | GPIO | Unpowered | 1) ADC2_VREFH - Input w / pull-up enabled<br>2) ADC2_VREFL - Input w / pull-down enabled[2]<br>3) Pad keepers enabled for both pins | Unpowered |
| 5 | ADC Analog Input Channel | ADC6 - ADC0 | GPIO | Unpowered | Input w / pull-downs enabled and bus keepers enabled | Unpowered |
| 6 | ADC7_RTCK Debug Port Clock | ADC7_RTCK | RTCK | Unpowered | JTAG RTCK output | Unpowered |
| 7 | JTAG | TDI, TMS, TCK, TDO | JTAG | Unpowered | 1) TDI, TMS, and TCK - Input w/pull-ups enabled and pad keepers disabled<br>2) TDO - Output | Unpowered |
| 7 | Nexus (includes JTAG and RTCK) | MDO07-MDO00, MSEO0_B, MSEO1_B, MCKO, EVTO_B, EVTI_B, RDY_B | Nexus | Unpowered | All Nexus signals are outputs with the exception of the input EVTI_B w / pull-up enabled | Unpowered |
| 8 | KBI | KBI_7-KBI_0 | GPIO | Powered (revert to default CRM KBI control) | 1) KBI_3:KBI_0 - outputs high<br>2) KBI_7:KBI_4 - inputs w / pull-downs enabled | Powered (revert to CRM KBI control) |

1 "Standard GPIO" refers to all GPIO not listed as special cases in this table.

2 ADC2_VREFL exits reset with pull-up enabled, but boot code changes to pull-down enabled for "clear FLASH" test. See Section 3.12.5, "FLASH Erase or Recovery Mode"

### 11.5.3 Low Power GPIO Operation

The MC1322x has been designed to provide extremely low power operation (see Section 3.3, “System Reset and Low Power GPIO Signal Connections”). Being held in hardware reset is the lowest power state, and in addition, the device can be put into two programmed low power modes:

- Hibernate mode - The MC1322x can be programmed to enter Hibernate as a very low power state. Similar to reset, all the GPIO with the exception of the eight KBI signals have the supply removed from them to eliminate leakage. The KBI signals IO remain powered and in their CRM programmed mode. The reference oscillator is stopped and there are a number of power savings options.
- Doze mode - The MC1322x can also be programmed to enter Doze as a second form of very low power state. Doze is the same as Hibernate with the exception of the reference oscillator remaining running.

The use of the GPIO during low power operation is extensively discussed in Section 3.3, “System Reset and Low Power GPIO Signal Connections”

### 11.5.4 KBI Signals Shared with GPIO

The KBI signals are very unique:

- The KBI signals are the only GPIO always powered - When the MC1322x is “awake”, the GPIO module is always in control of the pads. When Hibernate or Doze low power mode is activated, all the KBI signals come under control of the CRM module.
- The KBI can be used as standard GPIO if desired.
- KBI7-KBI4 are typically configured as inputs with pull-downs/pullups enabled. In low power mode a pull-up or pull-down is always enabled depending on the programmed interrupt request active sense.
- The KBI7-KBI4 external interrupt request capability is always available and is controlled by the CRM. However, the GPIO module must configure the GPIO for KBI7-KBI4 as an input(s) with a pull-up/pull-down for the interrupt request to be used in normal mode.
- KBI3-KBI0 are configured as outputs. Out-of-default (POR), they are forced to the “1” state. They can be programmed for state in the WU_CNTL Register.
- KBI0 can be enabled as an “out-of-wake-up” indicator.

**NOTE**

Use of the KBI is extensively discussed in the following to sections

- Section 3.4, “Using KBI Signals as Keypad/Pushbutton Interface”
- Section 5.1.5, “KBI Interface Signals”

### 11.5.5 Shared ADC Analog Signals

As seen in Table 11-6, the GPIO signals shared with the ADC module are a special case

- ADC Reference Voltage signals ADC1_VREFH & ADC1_VREFL initialize with their analog functionality
- ADC Reference Voltage signals ADC2_VREFH & ADC2_VREFL initialize with their digital functionality with pull-up/pull-down and pad keeper functions enabled. These signals are used as part of the boot process; see Section 3.12, "Bootloader".
- All the ADC channel inputs ADC6-ADC0 initialize as digital inputs with pull-downs and bus keepers enabled.
- ADC channel input ADC7 initializes as digital output RTCK for the debug port.
- The bus keepers must be enabled when any of these signals are used as digital inputs, and conversely the bus keepers must be disabled for analog mode.

**NOTE**

For more information on the analog use of these signals, see Section 17.4, "External Signal Description".

## 11.6 GPIO Module Register Memory Map

The GPIO module is programmed via a set of memory-mapped registers

- The base address is 0x8000_0000
- The register memory map for the module is listed in Table 11-7
- These registers should only be accessed as 32 bits

**Table 11-7. GPIO Module Register Memory Map**

| Address | Name | Access Type | Access Width |
|---|---|---|---|
| Base + 0x00 | GPIO Pad Direction for GPIO 00-31 (GPIO_PAD_DIR0) | R/W | 32 only |
| Base + 0x04 | GPIO Pad Direction for GPIO 32-63 (GPIO_PAD_DIR1) | R/W | 32 only |
| Base + 0x08 | GPIO Data for GPIO 00-31 (GPIO_DATA0) | R/W | 32 only |
| Base + 0x0C | GPIO Data for GPIO 32-63 (GPIO_DATA1) | R/W | 32 only |
| Base + 0x10 | GPIO Pad Pull-up Enable for GPIO 00-31 (GPIO_PAD_PU_EN0) | R/W | 32 only |
| Base + 0x14 | GPIO Pad Pull-up Enable for GPIO 32-63 (GPIO_PAD_PU_EN1) | R/W | 32 only |
| Base + 0x18 | GPIO Function Select for GPIO 00-15 (GPIO_FUNC_SEL0) | R/W | 32 only |
| Base + 0x1C | GPIO Function Select for GPIO 16-31 (GPIO_FUNC_SEL1) | R/W | 32 only |
| Base + 0x20 | GPIO Function Select for GPIO 32-47 (GPIO_FUNC_SEL2) | R/W | 32 only |
| Base + 0x24 | GPIO Function Select for GPIO 48-63 (GPIO_FUNC_SEL3) | R/W | 32 only |
| Base + 0x28 | GPIO Data Select for GPIO 00-31 (GPIO_DATA_SEL0) | R/W | 32 only |
| Base + 0x2C | GPIO Data Select for GPIO 32-63 (GPIO_DATA_SEL1) | R/W | 32 only |

**Table 11-7. GPIO Module Register Memory Map (continued)**

| Address | Name | Access Type | Access Width |
|---|---|---|---|
| Base + 0x30 | GPIO Pad Pull-up Select for GPIO 00-31 (GPIO_PAD_PU_SEL0) | R/W | 32 only |
| Base + 0x34 | GPIO Pad Pull-up Select for GPIO 32-63 (GPIO_PAD_PU_SEL1) | R/W | 32 only |
| Base + 0x38 | GPIO Pad Hysteresis Enable for GPIO 00-31 (GPIO_PAD_HYST_EN0) | R/W | 32 only |
| Base + 0x3C | GPIO Pad Hysteresis Enable for GPIO 32-63 (GPIO_PAD_HYST_EN1) | R/W | 32 only |
| Base + 0x40 | GPIO Pad Keeper Enable for GPIO 00-31 (GPIO_PAD_KEEP0) | R/W | 32 only |
| Base + 0x44 | GPIO Pad Keeper Enable for GPIO 32-63 (GPIO_PAD_KEEP1) | R/W | 32 only |
| Base + 0x48 | GPIO Data Set for GPIO 00-31 (GPIO_DATA_SET0) | W | 32 only |
| Base + 0x4C | GPIO Data Set for GPIO 32-63 (GPIO_DATA_SET1) | W | 32 only |
| Base + 0x50 | GPIO Data Reset for GPIO 00-31 (GPIO_DATA_RESET0) | W | 32 only |
| Base + 0x54 | GPIO Data Reset for GPIO 32-63 (GPIO_DATA_RESET1) | W | 32 only |
| Base + 0x58 | GPIO Pad Direction Set for GPIO 00-31 (GPIO_PAD_DIR_SET0) | W | 32 only |
| Base + 0x5C | GPIO Pad Direction Set for GPIO 32-63 (GPIO_PAD_DIR_SET1) | W | 32 only |
| Base + 0x60 | GPIO Pad Direction Reset for GPIO 00-31 (GPIO_PAD_DIR_RESET0) | W | 32 only |
| Base + 0x64 | GPIO Pad Direction Reset for GPIO 32-63 (GPIO_PAD_DIR_RESET1) | W | 32 only |

## 11.7 GPIO Module Registers and Control Bits

The following sections describe the functional registers in the GPIO module.

### 11.7.1 GPIO Pad Direction Registers (GPIO_PAD_DIR0 and GPIO_PAD_DIR1)

The GPIO Pad Direction Registers control the direction of the IO when the GPIO mode is selected. Two 32-bit registers (GPIO_PAD_DIR0 and GPIO_PAD_DIR1) are required to control the 64 possible GPIO.

| GPIO_PAD_DIR0 | | | | FOR GPIO MODES ONLY - GPIOs GPIO_00 - GPIO_31 | | | | | | | | | Addr Base+0x00 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $Offset_{16}$ | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| Base+0x00 | GPIO_31 | GPIO_30 | GPIO_29 | GPIO_28 | GPIO_27 | GPIO_26 | GPIO_25 | GPIO_24 | GPIO_23 | GPIO_22 | GPIO_21 | GPIO_20 | GPIO_19 | GPIO_18 | GPIO_17 | GPIO_16 |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| $Offset_{16}$ | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| Base+0x00 | GPIO_15 | GPIO_14 | GPIO_13 | GPIO_12 | GPIO_11 | GPIO_10 | GPIO_09 | GPIO_08 | GPIO_07 | GPIO_06 | GPIO_05 | GPIO_04 | GPIO_03 | GPIO_02 | GPIO_01 | GPIO_00 |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| GPIO_PAD_DIR1 | | | | FOR GPIO MODES ONLY - GPIOs GPIO_32 - GPIO_63 | | | | | | | | | Addr Base+0x00-0x04 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $Offset_{16}$ | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| Base+0x04 | GPIO_63 | GPIO_62 | GPIO_61 | GPIO_60 | GPIO_59 | GPIO_58 | GPIO_57 | GPIO_56 | GPIO_55 | GPIO_54 | GPIO_53 | GPIO_52 | GPIO_51 | GPIO_50 | GPIO_49 | GPIO_48 |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| $Offset_{16}$ | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| Base+0x04 | GPIO_47 | GPIO_46 | GPIO_45 | GPIO_44 | GPIO_43 | GPIO_42 | GPIO_41 | GPIO_40 | GPIO_39 | GPIO_38 | GPIO_37 | GPIO_36 | GPIO_35 | GPIO_34 | GPIO_33 | GPIO_32 |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 11-8. GPIO Direction Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 0-31 | **GPIO_PAD_DIRx[31:0]** - These bits control the directions of the pins when in GPIO Mode only. In Peripheral Mode, these bits have no effect on the output enables or pull-up enables (for GPIOs which have no Peripheral mode, the bits always take effect). | 0 (default) = Pin is an input. Pull-ups are dependent on the value of the GPIO_PAD_PU_EN registers. Exceptions: GPIOs 22-25 (KBI0-3).<br><br>1 = Pin is an output; pull-ups are disabled. |

## 11.7.2 GPIO Data Registers (GPIO_DATA1 and GPIO_DATA0)

The GPIO Data Registers supply the data when the output is enabled and GPIO mode is selected. Two 32-bit registers (GPIO_DATA0 and GPIO_DATA1) are required to control the 64 possible GPIO.

| GPIO_DATA0 | | | | FOR GPIO MODES ONLY - GPIOs GPIO_00 - GPIO_31 | | | | | | | | | Addr Base+0x08 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $Offset_{16}$ | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| Base+0x08 | GPIO_31 | GPIO_30 | GPIO_29 | GPIO_28 | GPIO_27 | GPIO_26 | GPIO_25 | GPIO_24 | GPIO_23 | GPIO_22 | GPIO_21 | GPIO_20 | GPIO_19 | GPIO_18 | GPIO_17 | GPIO_16 |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| $Offset_{16}$ | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| Base+0x08 | GPIO_15 | GPIO_14 | GPIO_13 | GPIO_12 | GPIO_11 | GPIO_10 | GPIO_09 | GPIO_08 | GPIO_07 | GPIO_06 | GPIO_05 | GPIO_04 | GPIO_03 | GPIO_02 | GPIO_01 | GPIO_00 |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| GPIO_DATA1 | | | | FOR GPIO MODES ONLY - GPIOs GPIO_32 - GPIO_63 | | | | | | | | | Addr Base+0x0C | | | |
| $Offset_{16}$ | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| Base+0x0C | GPIO_63 | GPIO_62 | GPIO_61 | GPIO_60 | GPIO_59 | GPIO_58 | GPIO_57 | GPIO_56 | GPIO_55 | GPIO_54 | GPIO_53 | GPIO_52 | GPIO_51 | GPIO_50 | GPIO_49 | GPIO_48 |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| $Offset_{16}$ | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| Base+0x0C | GPIO_47 | GPIO_46 | GPIO_45 | GPIO_44 | GPIO_43 | GPIO_42 | GPIO_41 | GPIO_40 | GPIO_39 | GPIO_38 | GPIO_37 | GPIO_36 | GPIO_35 | GPIO_34 | GPIO_33 | GPIO_32 |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 11-9. GPIO Data Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 0-31 | **GPIO_DATA1x[31:0]** - These bits contain the output data for the GPIOs when in GPIO Mode. | Default = 0<br>except GPIOs 22-25<br>(KBI0-3) |

## 11.7.3 GPIO Pull-up Enable Registers (GPIO_PAD_PU_EN1 and GPIO_PAD_PU_EN0)

The GPIO pull-up enable registers select if a pull-up or pull-down is enabled if the pad is an input (for all modes). A pull-up or pull-down can be selected via the PAD_PULLUP_SELECT registers. Two 32-bit registers (GPIO_PAD_PU_EN0 and GPIO_PAD_PU_EN1) are required to control the 64 possible GPIO.

| GPIO_PAD_PU_EN0 | | | | GPIOs GPIO_00 - GPIO_31 | | | | | | | | | Addr<br>Base+0x10 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $Offset_{16}$ | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| Base+0x10 | GPIO_31 | GPIO_30 | GPIO_29 | GPIO_28 | GPIO_27 | GPIO_26 | GPIO_25 | GPIO_24 | GPIO_23 | GPIO_22 | GPIO_21 | GPIO_20 | GPIO_19 | GPIO_18 | GPIO_17 | GPIO_16 |
| RESET | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | | | | | | | | | | | | | | | | |
| $Offset_{16}$ | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| Base+0x10 | GPIO_15 | GPIO_14 | GPIO_13 | GPIO_12 | GPIO_11 | GPIO_10 | GPIO_09 | GPIO_08 | GPIO_07 | GPIO_06 | GPIO_05 | GPIO_04 | GPIO_03 | GPIO_02 | GPIO_01 | GPIO_00 |
| RESET | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | | | | | | | | | | | | | | | | |
| GPIO_PAD_PU_EN1 | | | | GPIOs GPIO_32 - GPIO_63 | | | | | | | | | Addr<br>Base+0x14 | | | |
| $Offset_{16}$ | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| Base+0x14 | GPIO_63 | GPIO_62 | GPIO_61 | GPIO_60 | GPIO_59 | GPIO_58 | GPIO_57 | GPIO_56 | GPIO_55 | GPIO_54 | GPIO_53 | GPIO_52 | GPIO_51 | GPIO_50 | GPIO_49 | GPIO_48 |
| RESET | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | | | | | | | | | | | | | | | | |
| $Offset_{16}$ | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| Base+0x14 | GPIO_47 | GPIO_46 | GPIO_45 | GPIO_44 | GPIO_43 | GPIO_42 | GPIO_41 | GPIO_40 | GPIO_39 | GPIO_38 | GPIO_37 | GPIO_36 | GPIO_35 | GPIO_34 | GPIO_33 | GPIO_32 |
| RESET | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

**Table 11-10. GPIO Pull-up Enable Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 0-31 | **GPIO_PAD_PU_ENx[31:0]** - These bits control whether pull-ups/pull-downs are enabled for inputs in the various functional modes. Pull-ups/pull-downs are automatically disabled for outputs in these modes. | 0 = Pull-ups/pull-downs disabled for inputs<br>1 = (Default) Pull-ups/pull-downs enabled for inputs |

## 11.7.4 GPIO Function Select Registers (GPIO_FUNC_SEL3, GPIO_FUNC_SEL2, GPIO_FUNC_SEL1, and GPIO_FUNC_SEL0)

The GPIO Function Select Registers provide a 2-bit function select field for each GPIO. The 2-bit field selects one of 4 functions for each GPIO. Four32-bit registers (GPIO_FUNC_SEL0, GPIO_FUNC_SEL1, GPIO_FUNC_SEL2, and GPIO_FUNC_SEL3) are required to control the 64 possible GPIO.

| GPIO_FUNC_SEL0 | | | | GPIOs GPIO_15 - GPIO_0 | | | | | | | | | Addr Base+0x18 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $Offset_{16}$ | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| Base+0x18 | GPIO_15 | | GPIO_14 | | GPIO_13 | | GPIO_12 | | GPIO_11 | | GPIO_10 | | GPIO_9 | | GPIO_8 | |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| $Offset_{16}$ | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| Base+0x18 | GPIO_7 | | GPIO_6 | | GPIO_5 | | GPIO_4 | | GPIO_3 | | GPIO_2 | | GPIO_1 | | GPIO_0 | |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| GPIO_FUNC_SEL1 | | | | GPIOs GPIO_31 - GPIO_16 | | | | | | | | | Addr Base+0x1C | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $Offset_{16}$ | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| Base+0x1C | GPIO_31 | | GPIO_30 | | GPIO_28 | | GPIO_28 | | GPIO_27 | | GPIO_26 | | GPIO_25 | | GPIO_24 | |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| $Offset_{16}$ | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| Base+0x1C | GPIO_23 | | GPIO_22 | | GPIO_21 | | GPIO_20 | | GPIO_19 | | GPIO_18 | | GPIO_17 | | GPIO0_16 | |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| GPIO_FUNC_SEL2 | | | | GPIOs GPIO_47 - GPIO_32 | | | | | | | | Addr Base+0x20 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $Offset_{16}$ | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| Base+0x20 | GPIO_47 | | GPIO_46 | | GPIO_45 | | GPIO_44 | | GPIO_43 | | GPIO_42 | | GPIO_41 | | GPIO_40 | |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| $Offset_{16}$ | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| Base+0x20 | GPIO_39 | | GPIO_38 | | GPIO_37 | | GPIO_36 | | GPIO_35 | | GPIO_34 | | GPIO_33 | | GPIO0_32 | |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| GPIO_FUNC_SEL3 | | | | GPIOs GPIO_63 - GPIO_48 | | | | | | | | Addr Base+0x24 | | | | |
| $Offset_{16}$ | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| Base+0x24 | GPIO_63 | | GPIO_62 | | GPIO_61 | | GPIO_60 | | GPIO_59 | | GPIO_58 | | GPIO_57 | | GPIO_56 | |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| $Offset_{16}$ | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| Base+0x24 | GPIO_55 | | GPIO_54 | | GPIO_53 | | GPIO_52 | | GPIO_51 | | GPIO_50 | | GPIO_49 | | GPIO0_48 | |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 11-11. GPIO Functional Mode Select Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| For each GPIO | **GPIO_FUNC_SEL[1:0] -** There is a 2-bit field for each GPIO that selects function | 0 = Functional Mode 0 (default)<br>1 = Alternate Mode 1 (Function 1)<br>2 = Alternate Mode 2 (Function 2)<br>3 = Alternate Mode 3 (Function 3)<br><br>The Functional Modes are different for each grouping of GPIOs. See Table 11-12 for a complete listing of GPIO port assignments and functional modes. |

**Table 11-12. Pin Function vs. Function Select State**

| GPIO PORT | Chip Pin Name | Functional 0 (Default) | | Functional 1 | | Functional 2 | | Functional 3 | |
|---|---|---|---|---|---|---|---|---|---|
| | | FUNC_SEL = 0 | I/O | FUNC_SEL = 1 | I/O | FUNC_SEL = 2 | I/O | FUNC_SEL = 3 | I/O |
| **PERIPHERAL CONNECTIONS** | | | | | | | | | |
| GPIO_00 | SSI_TX | GPIO_00 | I/O | SSI_TX | Tri-O | GPIO_00 | I/O | GPIO_00 | I/O |
| GPIO_01 | SSI_RX | GPIO_01 | I/O | SSI_RX | I | GPIO_01 | I/O | GPIO_01 | I/O |
| GPIO_02 | SSI_FSYNC | GPIO_02 | I/O | SSI_MST_FSYNC<br>SSI_SLV_FSYNC | Tri-O<br>I | GPIO_02 | I/O | GPIO_02 | I/O |
| GPIO_03 | SSI_BITCLK | GPIO_03 | I/O | SSI_MST_BITCLK<br>SSI_SLV_BITCLK | Tri-O<br>I | GPIO_03 | I/O | GPIO_03 | I/O |
| GPIO_04 | SPI_SS | GPIO_04 | I/O | SPI_SS | I<br>Tri-O | GPIO_04 | I/O | GPIO_04 | I/O |
| GPIO_05 | SPI_MISO | GPIO_05 | I/O | SPI_MISO | Tri-O<br>I | GPIO_05 | I/O | GPIO_05 | I/O |
| GPIO_06 | SPI_MOSI | GPIO_06 | I/O | SPI_MOSI | I<br>Tri-O | GPIO_06 | I/O | GPIO_06 | I/O |
| GPIO_07 | SPI_SCK | GPIO_07 | I/O | SPI_SCK | I<br>Tri-O | GPIO_07 | I/O | GPIO_07 | I/O |
| GPIO_08 | TMR0 | GPIO_08 | I/O | TMR0_IN<br>TMR0_OUT | I<br>O | GPIO_08 | I/O | GPIO_08 | I/O |
| GPIO_09 | TMR1 | GPIO_09 | I/O | TMR1_IN<br>TMR1_OUT | I<br>O | GPIO_09 | I/O | GPIO_09 | I/O |
| GPIO_10 | TMR2 | GPIO_10 | I/O | TMR2_IN<br>TMR2_OUT | I<br>O | GPIO_10 | I/O | GPIO_10 | I/O |
| GPIO_11 | TMR3 | GPIO_11 | I/O | TMR3_IN<br>TMR3_OUT | I<br>O | GPIO_11 | I/O | GPIO_11 | I/O |
| GPIO_12 | I2C_SCL | GPIO_12 | I/O | I2C_SCL | I/O* | GPIO_12 | I/O | GPIO_12 | I/O |
| GPIO_13 | I2C_SDA | GPIO_13 | I/O | I2C_SDA | I/O* | GPIO_13 | I/O | GPIO_13 | I/O |
| GPIO_14 | UART1_TX | GPIO_14 | I/O | UART1_TX | Tri-O | GPIO_14 | I/O | GPIO_14 | I/O |
| GPIO_15 | UART1_RX | GPIO_15 | I/O | UART1_RX | I | GPIO_15 | I/O | GPIO_15 | I/O |
| GPIO_16 | UART1_CTS | GPIO_16 | I/O | UART1_CTS | O | GPIO_16 | I/O | GPIO_16 | I/O |
| GPIO_17 | UART1_RTS | GPIO_17 | I/O | UART1_RTS | I | GPIO_17 | I/O | GPIO_17 | I/O |
| GPIO_18 | UART2_TX | GPIO_18 | I/O | UART2_TX | Tri-O | GPIO_18 | I/O | GPIO_18 | I/O |
| GPIO_19 | UART2_RX | GPIO_19 | I/O | UART2_RX | I | GPIO_19 | I/O | GPIO_19 | I/O |
| GPIO_20 | UART2_CTS | GPIO_20 | I/O | UART2_CTS | O | GPIO_20 | I/O | GPIO_20 | I/O |
| GPIO_21 | UART2_RTS | GPIO_21 | I/O | UART2_RTS | I | GPIO_21 | I/O | GPIO_21 | I/O |
| **DEDICATED GPIO CONNECTIONS** | | | | | | | | | |
| GPIO_22 | KBI_0** | GPIO_22 | I/O | GPIO_22 | I/O | GPIO_22 | I/O | GPIO_22 | I/O |
| GPIO_23 | KBI_1** | GPIO_23 | I/O | GPIO_23 | I/O | GPIO_23 | I/O | GPIO_23 | I/O |
| GPIO_24 | KBI_2** | GPIO_24 | I/O | GPIO_24 | I/O | GPIO_24 | I/O | GPIO_24 | I/O |
| GPIO_25 | KBI_3** | GPIO_25 | I/O | GPIO_25 | I/O | GPIO_25 | I/O | GPIO_25 | I/O |
| GPIO_26 | KBI_4** | GPIO_26/<br>KBI_4_Wake_Int | I/O | GPIO_26/<br>KBI_4_Wake_Int | I | GPIO_26/<br>KBI_4_Wake_Int | I/O | GPIO_26/<br>KBI_4_Wake_Int | I/O |
| GPIO_27 | KBI_5** | GPIO_27/<br>KBI_5_Wake_Int | I/O | GPIO_27/<br>KBI_5_Wake_Int | I | GPIO_27/<br>KBI_5_Wake_Int | I/O | GPIO_27/<br>KBI_5_Wake_Int | I/O |
| GPIO_28 | KBI_6** | GPIO_28/<br>KBI_6_Wake_Int | I/O | GPIO_28/<br>KBI_6_Wake_Int | I | GPIO_28/<br>KBI_6_Wake_Int | I/O | GPIO_28/<br>KBI_6_Wake_Int | I/O |
| GPIO_29 | KBI_7** | GPIO_29/<br>KBI_7_Wake_Int | I/O | GPIO_29/<br>KBI_7_Wake_Int | I | GPIO_29/<br>KBI_7_Wake_Int | I/O | GPIO_29/<br>KBI_7_Wake_Int | I/O |

**Table 11-12. Pin Function vs. Function Select State (continued)**

| GPIO PORT | Chip Pin Name | Functional 0 (Default) | | Functional 1 | | Functional 2 | | Functional 3 | |
|---|---|---|---|---|---|---|---|---|---|
| **ADC CONNECTIONS** | | | | | | | | | |
| GPIO_30 | ADC_Chan0*** | GPIO_30 | I/O | ADC Analog IP (Ch. 0) | Tri | GPIO_30 | I/O | GPIO_30 | I/O |
| GPIO_31 | ADC_Chan1*** | GPIO_31 | I/O | ADC Analog IP (Ch. 1) | Tri | GPIO_31 | I/O | GPIO_31 | I/O |
| GPIO_32 | ADC_Chan2*** | GPIO_32 | I/O | ADC Analog IP (Ch. 2) | Tri | GPIO_32 | I/O | GPIO_32 | I/O |
| GPIO_33 | ADC_Chan3*** | GPIO_33 | I/O | ADC Analog IP (Ch. 3) | Tri | GPIO_33 | I/O | GPIO_33 | I/O |
| GPIO_34 | ADC_Chan4*** | GPIO_34 | I/O | ADC Analog IP (Ch. 4) | Tri | GPIO_34 | I/O | GPIO_34 | I/O |
| GPIO_35 | ADC_Chan5*** | GPIO_35 | I/O | ADC Analog IP (Ch. 5) | Tri | GPIO_35 | I/O | GPIO_35 | I/O |
| GPIO_36 | ADC_Chan6*** | GPIO_36 | I/O | ADC Analog IP (Ch. 6) | Tri | GPIO_36 | I/O | GPIO_36 | I/O |
| GPIO_37 | ADC_Chan7*** | JTAG RTCK | O | ADC Analog IP (Ch. 7) | Tri | GPIO_37 | I/O | GPIO_37 | I/O |
| GPIO_38 | ADC2_VrefH*** | GPIO_38 | I/O | ADC2_VrefH | Tri | GPIO_38 | I/O | GPIO_38 | I/O |
| GPIO_39 | ADC2_VrefL*** | GPIO_39 | I/O | ADC2_VrefL | Tri | GPIO_39 | I/O | GPIO_39 | I/O |
| GPIO_40 | ADC1_VrefH*** | ADC1_VrefH | Tri | GPIO_40 | I/O | GPIO_40 | I/O | GPIO_40 | I/O |
| GPIO_41 | ADC1_VrefL*** | ADC1_VrefL | Tri | GPIO_4 | I/O | GPIO_41 | I/O | GPIO_41 | I/O |
| **CONTROL PINS** | | | | | | | | | |
| GPIO_42 | TxRx_ANT_1 | GPIO_42 | I/O | TRX Seq Mgr GPO1 (antenna sw1) | O | GPIO_42 | - | GPIO_42 | I/O |
| GPIO_43 | TxRx_ANT_2 | GPIO_43 | I/O | TRX Seq Mgr GPO2 (antenna sw2) | O | GPIO_43 | I/O | GPIO_43 | I/O |
| GPIO_44 | TX_ON | GPIO_44 | I/O | TX Seq Mgr GPO1 (PA) | O | RX Seq Mgr GPO1 (LNA) | O | GPIO_44 | I/O |
| GPIO_45 | RX_ON | GPIO_45 | I/O | TX Seq Mgr GPO2 | O | RX Seq Mgr GPO2 | O | GPIO_45 | I/O |
| **JTAG CONNECTIONS (4)** | | | | | | | | | |
| GPIO_46 | TMS | JTA_TMS | I | GPIO_46 | I/O | GPIO_46 | I/O | GPIO_46 | I/O |
| GPIO_47 | TCK | JTA_TCK | I | GPIO_47 | I/O | GPIO_47 | I/O | GPIO_47 | I/O |
| GPIO_48 | TDI | JTA_TDI | I | GPIO_48 | I/O | GPIO_48 | I/O | GPIO_48 | I/O |
| GPIO_49 | TDO | JTA_TDO | O | GPIO_49 | I/O | GPIO_49 | I/O | GPIO_49 | I/O |
| **NEXUS PINS** | | | | | | | | | |
| GPIO_50 | MCKO | MCKO Msg Clk | O | GPIO_50 | I/O | GPIO_50 | I/O | GPIO_50 | I/O |
| GPIO_51 | MDO0 | Msg Data Out | O | GPIO_51 | I/O | GPIO_51 | I/O | GPIO_51 | I/O |
| GPIO_52 | MDO1 | Msg Data Out | O | GPIO_52 | I/O | GPIO_52 | I/O | GPIO_52 | I/O |
| GPIO_53 | MDO2 | Msg Data Out | O | GPIO_53 | I/O | GPIO_53 | I/O | GPIO_53 | I/O |
| GPIO_54 | MDO3 | Msg Data Out | O | GPIO_54 | I/O | GPIO_54 | I/O | GPIO_54 | I/O |
| GPIO_55 | MDO4 | Msg Data Out | O | GPIO_55 | I/O | GPIO_55 | I/O | GPIO_55 | I/O |
| GPIO_56 | MDO5 | Msg Data Out | O | GPIO_56 | I/O | GPIO_56 | I/O | GPIO_56 | I/O |
| GPIO_57 | MDO6 | Msg Data Out | O | GPIO_57 | I/O | GPIO_57 | I/O | GPIO_57 | I/O |
| GPIO_58 | MDO7 | Msg Data Out | O | GPIO_58 | I/O | GPIO_58 | I/O | GPIO_58 | I/O |

**Table 11-12. Pin Function vs. Function Select State (continued)**

| GPIO PORT | Chip Pin Name | Functional 0 (Default) | | Functional 1 | | Functional 2 | | Functional 3 | |
|---|---|---|---|---|---|---|---|---|---|
| GPIO_59 | MSEO0_B | Msg Start/End | O | GPIO_59 | I/O | GPIO_59 | I/O | GPIO_59 | I/O |
| GPIO_60 | MSEO1_B | Msg Start/End | O | GPIO_60 | I/O | GPIO_60 | I/O | GPIO_60 | I/O |
| GPIO_61 | RDY_B | Nexus Ready | O | GPIO_61 | I/O | GPIO_61 | I/O | GPIO_61 | I/O |
| GPIO_62 | EVTO_B | Event Out | O | GPIO_62 | I/O | GPIO_62 | I/O | GPIO_62 | I/O |
| GPIO_63 | EVTI_B | Event In | I | GPIO_63 | I/O | GPIO_63 | I/O | GPIO_63 | I/O |

## 11.7.5 GPIO Data Select Registers (GPIO_DATA_SEL1 and GPIO_DATA_SEL0)

The GPIO Data Select Registers determine the read data source for each GPIO when a read of the GPIO Data Register (GPIO_DATA1 and GPIO_DATA0) occurs. The data can be read from the pad or the GPIO Data Register. Two 32-bit registers (GPIO_DATA_SEL1 and GPIO_DATA_SEL0) are required to control the 64 possible GPIO.

| GPIO_DATA_SEL0 | | | | GPIOs GPIO_00 - GPIO_31 | | | | | | | | | Addr Base+0x28 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $Offset_{16}$ | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| Base+0x28 | GPIO_31 | GPIO_30 | GPIO_29 | GPIO_28 | GPIO_27 | GPIO_26 | GPIO_25 | GPIO_24 | GPIO_23 | GPIO_22 | GPIO_21 | GPIO_20 | GPIO_19 | GPIO_18 | GPIO_17 | GPIO_16 |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| $Offset_{16}$ | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| Base+0x28 | GPIO_15 | GPIO_14 | GPIO_13 | GPIO_12 | GPIO_11 | GPIO_10 | GPIO_09 | GPIO_08 | GPIO_07 | GPIO_06 | GPIO_05 | GPIO_04 | GPIO_03 | GPIO_02 | GPIO_01 | GPIO_00 |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| GPIO_DATA_SEL1 | | | | GPIOs GPIO_32 - GPIO_63 | | | | | | | | | Addr Base+0x2C | | | |
| $Offset_{16}$ | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| Base+0x2C | GPIO_63 | GPIO_62 | GPIO_61 | GPIO_60 | GPIO_59 | GPIO_58 | GPIO_57 | GPIO_56 | GPIO_55 | GPIO_54 | GPIO_53 | GPIO_52 | GPIO_51 | GPIO_50 | GPIO_49 | GPIO_48 |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| $Offset_{16}$ | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| Base+0x2C | GPIO_47 | GPIO_46 | GPIO_45 | GPIO_44 | GPIO_43 | GPIO_42 | GPIO_41 | GPIO_40 | GPIO_39 | GPIO_38 | GPIO_37 | GPIO_36 | GPIO_35 | GPIO_34 | GPIO_33 | GPIO_32 |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 11-13. GPIO Read Control Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 0-31 | **GPIO_DATA_SELx[31:0]** - Each bit controls whether data is read directly from the GPIO pad, or from the GPIO Data Register | 0 = (Default) Data is read from the pad.<br>1 = Data is read from the GPIO Data Register |

## 11.7.6 GPIO Pad Pull-up Select (GPIO_PAD_PU_SEL1 and GPIO_PAD_PU_SEL0)

The GPIO Pad Pull-up Select Registers determine whether a pull-up versus a pull-down resistor is used if the pad is an input and if the pull-up/pull-down is enabled via the GPIO pull-up enable registers. Two 32-bit registers (GPIO_PAD_PU_SEL1 and GPIO_PAD_PU_SEL0) are required to control the 64 possible GPIO.

| GPIO_PAD_PU_SEL0 | | | | GPIOs GPIO_00 - GPIO_31 | | | | | | | | | Addr Base+0x30 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $Offset_{16}$ | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| Base+0x30 | GPIO_31 | GPIO_30 | GPIO_29 | GPIO_28 | GPIO_27 | GPIO_26 | GPIO_25 | GPIO_24 | GPIO_23 | GPIO_22 | GPIO_21 | GPIO_20 | GPIO_19 | GPIO_18 | GPIO_17 | GPIO_16 |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| $Offset_{16}$ | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| Base+0x30 | GPIO_15 | GPIO_14 | GPIO_13 | GPIO_12 | GPIO_11 | GPIO_10 | GPIO_09 | GPIO_08 | GPIO_07 | GPIO_06 | GPIO_05 | GPIO_04 | GPIO_03 | GPIO_02 | GPIO_01 | GPIO_00 |
| RESET | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| GPIO_PAD_PU_SEL1 | | | | GPIOs GPIO_32 - GPIO_63 | | | | | | | | | | Addr Base+-0x34 | | |
| $Offset_{16}$ | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| Base+ 0x34 | GPIO_63 | GPIO_62 | GPIO_61 | GPIO_60 | GPIO_59 | GPIO_58 | GPIO_57 | GPIO_56 | GPIO_55 | GPIO_54 | GPIO_53 | GPIO_52 | GPIO_51 | GPIO_50 | GPIO_49 | GPIO_48 |
| RESET | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| | | | | | | | | | | | | | | | | |
| $Offset_{16}$ | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| Base+ 0x34 | GPIO_47 | GPIO_46 | GPIO_45 | GPIO_44 | GPIO_43 | GPIO_42 | GPIO_41 | GPIO_40 | GPIO_39 | GPIO_38 | GPIO_37 | GPIO_36 | GPIO_35 | GPIO_34 | GPIO_33 | GPIO_32 |
| RESET | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 11-14. GPIO PAD Pull-up Select Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 0-31 | **GPIO_PAD_PU_SELx[31:0]** - Each bit controls the selection of pull-up versus pull-down for an output pad. The corresponding GPIO_PAD_PU_EN bit must be asserted to activate either weak pull device. | 0 = Weak pull-down<br>(default for all except GPIOs 12-13 ($I^2C$), 63 (EVTI_B), 46 (TMS), 47 (TCK) and 48 (TDI))<br><br>1 = Weak pull-up<br>(default for GPIOs 12-13 ($I^2C$), 63 (EVTI_B), 46 (TMS), 47 (TCK) and 48 (TDI)) |

## 11.7.7 GPIO Pad Hysteresis Enable Registers (GPIO_PAD_HYST_EN1 and GPIO_PAD_HYST_EN0)

The GPIO Pad Hysteresis Enable Registers determine whether hysteresis is used if the pad is an input. Two 32-bit registers (GPIO_PAD_HYST_EN1 and GPIO_PAD_HYST_EN0) are required to control the 64 possible GPIO.

| GPIO_PAD_HYST_EN0 | | | | GPIOs GPIO_00 - GPIO_31 | | | | | | | | | Addr Base+0x38 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $Offset_{16}$ | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| Base+0x38 | GPIO_31 | GPIO_30 | GPIO_29 | GPIO_28 | GPIO_27 | GPIO_26 | GPIO_25 | GPIO_24 | GPIO_23 | GPIO_22 | GPIO_21 | GPIO_20 | GPIO_19 | GPIO_18 | GPIO_17 | GPIO_16 |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| $Offset_{16}$ | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| Base+0x38 | GPIO_15 | GPIO_14 | GPIO_13 | GPIO_12 | GPIO_11 | GPIO_10 | GPIO_09 | GPIO_08 | GPIO_07 | GPIO_06 | GPIO_05 | GPIO_04 | GPIO_03 | GPIO_02 | GPIO_01 | GPIO_00 |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| GPIO_PAD_HYST_EN1 | | | | GPIOs GPIO_32 - GPIO_63 | | | | | | | | | Addr Base+0x3C | | | |
| $Offset_{16}$ | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| Base+0x3C | GPIO_63 | GPIO_62 | GPIO_61 | GPIO_60 | GPIO_59 | GPIO_58 | GPIO_57 | GPIO_56 | GPIO_55 | GPIO_54 | GPIO_53 | GPIO_52 | GPIO_51 | GPIO_50 | GPIO_49 | GPIO_48 |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| $Offset_{16}$ | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| Base+0x3C | GPIO_47 | GPIO_46 | GPIO_45 | GPIO_44 | GPIO_43 | GPIO_42 | GPIO_41 | GPIO_40 | GPIO_39 | GPIO_38 | GPIO_37 | GPIO_36 | GPIO_35 | GPIO_34 | GPIO_33 | GPIO_32 |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 11-15. GPIO Pad Hysteresis Enable Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| Bit31-Bit0 | **GPIO_PAD_HYST_ENx[31:0]** - Each bit can enable hysteresis for a GPIO pad when used as an input. | 0 = (Default) Hysteresis disabled<br>1 = Hysteresis enabled |

## 11.7.8 GPIO Pad Keeper Enable Registers (GPIO_PAD_KEEP1 and GPIO_PAD_KEEP0)

The GPIO Pad Keeper Enable Registers determine whether a pad "keeper" is enabled if the pad is an input. The keeper will retain the last state on the pin (high or low) if the pin ceases to be driven. Two 32-bit registers (GPIO_PAD_KEEP1 and GPIO_PAD_KEEP0) are required to control the 64 possible GPIO.

| GPIO_PAD_KEEP0 | | | | GPIOs GPIO_31 - GPIO_0 | | | | | | | | | Addr Base+0x40 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $Offset_{16}$ | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| Base+0x40 | GPIO_31 | GPIO_30 | GPIO_29 | GPIO_28 | GPIO_27 | GPIO_26 | GPIO_25 | GPIO_24 | GPIO_23 | GPIO_22 | GPIO_21 | GPIO_20 | GPIO_19 | GPIO_18 | GPIO_17 | GPIO_16 |
| RESET | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| $Offset_{16}$ | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| Base+0x40 | GPIO_15 | GPIO_14 | GPIO_13 | GPIO_12 | GPIO_11 | GPIO_10 | GPIO_09 | GPIO_08 | GPIO_07 | GPIO_06 | GPIO_05 | GPIO_04 | GPIO_03 | GPIO_02 | GPIO_01 | GPIO_00 |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| GPIO_PAD_KEEP1 | | | | GPIOs GPIO_32 - GPIO_63 | | | | | | | | | Addr Base+0x44 | | | |
| $Offset_{16}$ | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| Base+0x44 | GPIO_63 | GPIO_62 | GPIO_61 | GPIO_60 | GPIO_59 | GPIO_58 | GPIO_57 | GPIO_56 | GPIO_55 | GPIO_54 | GPIO_53 | GPIO_52 | GPIO_51 | GPIO_50 | GPIO_49 | GPIO_48 |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| $Offset_{16}$ | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| Base+0x44 | GPIO_47 | GPIO_46 | GPIO_45 | GPIO_44 | GPIO_43 | GPIO_42 | GPIO_41 | GPIO_40 | GPIO_39 | GPIO_38 | GPIO_37 | GPIO_36 | GPIO_35 | GPIO_34 | GPIO_33 | GPIO_32 |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |

**Table 11-16. GPIO Pad Pull-up Keeper Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 0-63 | **GPIO_PAD_KEEPx[31:0]** - These bits enable pull-up keepers on the pads. | 0 = (Default) Pull-up keep disabled for all but GPIOs 30-36, 38-39<br>1 = Pull-up keep enabled |

**NOTE**

When the dual-type analog/digital GPIOS (GPIOs 30-41) are configured as inputs (e.g., GPIO inputs), the corresponding GPIO Pad Keeper Enable Register bits must be 1.

## 11.7.9 GPIO Data Set Registers (GPIO_DATA_SET1 and GPIO_DATA_SET0)

The GPIO Data Set Registers are used to set specific bits to “1” in the GPIO Data Registers. Two registers (GPIO_DATA_SET1 and GPIO_DATA_SET0) are required to control the 64 possible GPIO.

- The data set registers are write-only.
- Any bits that are “1” in the present value of GPIO_DATA0 or GPIO_DATA1 will still be “1” after a write to its corresponding data set register.
- Any bit written to a “1” in the data set register will cause that bit to be set to “1” in its data register.
- The final contents of the data register are the logic “OR” of the previous data register contents and the data set register write data.

**NOTE**

The GPIO_DATA0 and GPIO_DATA1 registers can be written directly to program all bits at one time.

| GPIO_DATA_SET0 | | | | FOR GPIO MODES ONLY - GPIOs GPIO_00 - GPIO_31 | | | | | | | | | Addr Base+0x48 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $Offset_{16}$ | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| Base+0x48 | GPIO_31 | GPIO_30 | GPIO_29 | GPIO_28 | GPIO_27 | GPIO_26 | GPIO_25 | GPIO_24 | GPIO_23 | GPIO_22 | GPIO_21 | GPIO_20 | GPIO_19 | GPIO_18 | GPIO_17 | GPIO_16 |
| | | | | | | | | | | | | | | | | |
| $Offset_{16}$ | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| Base+0x48 | GPIO_15 | GPIO_14 | GPIO_13 | GPIO_12 | GPIO_11 | GPIO_10 | GPIO_09 | GPIO_08 | GPIO_07 | GPIO_06 | GPIO_05 | GPIO_04 | GPIO_03 | GPIO_02 | GPIO_01 | GPIO_00 |
| | | | | | | | | | | | | | | | | |
| GPIO_DATA_SET1 | | | | FOR GPIO MODES ONLY - GPIOs GPIO_32 - GPIO_63 | | | | | | | | | Addr Base+0x4C | | | |
| $Offset_{16}$ | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| Base+0x4C | GPIO_63 | GPIO_62 | GPIO_61 | GPIO_60 | GPIO_59 | GPIO_58 | GPIO_57 | GPIO_56 | GPIO_55 | GPIO_54 | GPIO_53 | GPIO_52 | GPIO_51 | GPIO_50 | GPIO_49 | GPIO_48 |
| | | | | | | | | | | | | | | | | |
| $Offset_{16}$ | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| Base+0x4C | GPIO_47 | GPIO_46 | GPIO_45 | GPIO_44 | GPIO_43 | GPIO_42 | GPIO_41 | GPIO_40 | GPIO_39 | GPIO_38 | GPIO_37 | GPIO_36 | GPIO_35 | GPIO_34 | GPIO_33 | GPIO_32 |

**Table 11-17. GPIO Data Set Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 0-63 | **GPIO_DATA_SETx[31:0]** - These bits set corresponding bits in the GPIO_DATAx Registers. | Write-only. Writing a 1 sets the corresponding bit in the GPIO_DATAx Register. Any existing bits set to 1 in GPIO_DATAx remain at 1. |

## 11.7.10 GPIO Data Reset Registers (GPIO_DATA_RESET1 and GPIO_DATA_RESET0)

The GPIO Data Reset Registers are used to set specific bits to “0” in the GPIO Data Registers. Two registers (GPIO_DATA_RESET1 and GPIO_DATA_RESET0) are required to control the 64 possible GPIO.

- The data reset registers are write-only.
- Any bits that are “0” in the present value of GPIO_DATA0 or GPIO_DATA1 will still be “0” after a write to its corresponding data reset register.
- Any bit written to a “1” in the data reset register will cause that bit to be reset to “0” in its data register.

- The final contents of the data register are the logic “AND” of the previous data register contents and the bitwise inverse of the data reset register write data.

**NOTE**

The GPIO_DATA0 and GPIO_DATA1 registers can be written directly to program all bits at one time.

| GPIO_DATA_RESET0 | | | | FOR GPIO MODES ONLY - GPIOs GPIO_00 - GPIO_31 | | | | | | | | | Addr Base+0x50 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $Offset_{16}$ | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| Base+0x50 | GPIO_31 | GPIO_30 | GPIO_29 | GPIO_28 | GPIO_27 | GPIO_26 | GPIO_25 | GPIO_24 | GPIO_23 | GPIO_22 | GPIO_21 | GPIO_20 | GPIO_19 | GPIO_18 | GPIO_17 | GPIO_16 |
| | | | | | | | | | | | | | | | | |
| $Offset_{16}$ | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| Base+0x50 | GPIO_15 | GPIO_14 | GPIO_13 | GPIO_12 | GPIO_11 | GPIO_10 | GPIO_09 | GPIO_08 | GPIO_07 | GPIO_06 | GPIO_05 | GPIO_04 | GPIO_03 | GPIO_02 | GPIO_01 | GPIO_00 |
| | | | | | | | | | | | | | | | | |
| GPIO_DATA_RESET1 | | | | FOR GPIO MODES ONLY - GPIOs GPIO_32 - GPIO_63 | | | | | | | | | Addr Base+0x54 | | | |
| $Offset_{16}$ | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| Base+0x54 | GPIO_63 | GPIO_62 | GPIO_61 | GPIO_60 | GPIO_59 | GPIO_58 | GPIO_57 | GPIO_56 | GPIO_55 | GPIO_54 | GPIO_53 | GPIO_52 | GPIO_51 | GPIO_50 | GPIO_49 | GPIO_48 |
| | | | | | | | | | | | | | | | | |
| $Offset_{16}$ | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| Base+0x54 | GPIO_47 | GPIO_46 | GPIO_45 | GPIO_44 | GPIO_43 | GPIO_42 | GPIO_41 | GPIO_40 | GPIO_39 | GPIO_38 | GPIO_37 | GPIO_36 | GPIO_35 | GPIO_34 | GPIO_33 | GPIO_32 |

**Table 11-18. GPIO Data Reset Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 0-63 | **GPIO_DATA_RESETx[31:0]** - These bits reset corresponding bits in the GPIO_DATAx Registers. | Write-only. Writing a 1 resets the corresponding bit in the GPIO_DATAx Register. Any existing bits set to 0 in GPIO_DATAx remain at 0. |

### 11.7.11 GPIO Pad Direction Set Registers (GPIO_PAD_DIR_SET1 and GPIO_PAD_DIR_SET0)

The GPIO Pad Direction Set Registers are used to set specific bits to “1” in the GPIO Pad Direction Registers (set GPIO to output). Two registers (GPIO_PAD_DIR_SET1 and GPIO_PAD_DIR_SET0) are required to control the 64 possible GPIO.

- The pad direction set registers are write-only.
- Any bits that are “1” in the present value of GPIO_PAD_DIR0 or GPIO_PAD_DIR1 will still be “1” after a write to its corresponding pad direction set register.
- Any bit written to a “1” in the pad direction set register will cause that bit to be set to “1” in its pad direction register.
- The final contents of the pad direction register are the logic “OR” of the previous pad direction register contents and the pad direction set register write data.

**NOTE**

The GPIO_PAD_DIR0 and GPIO_PAD_DIR1 registers can be written directly to program all bits at one time.

| GPIO_DATA_DIR_SET0 | | | | FOR GPIO MODES ONLY - GPIOs GPIO_00 - GPIO_31 | | | | | | | | | Addr Base+0x58 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $Offset_{16}$ | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| Base+0x58 | GPIO_31 | GPIO_30 | GPIO_29 | GPIO_28 | GPIO_27 | GPIO_26 | GPIO_25 | GPIO_24 | GPIO_23 | GPIO_22 | GPIO_21 | GPIO_20 | GPIO_19 | GPIO_18 | GPIO_17 | GPIO_16 |
| | | | | | | | | | | | | | | | | |
| $Offset_{16}$ | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| Base+0x58 | GPIO_15 | GPIO_14 | GPIO_13 | GPIO_12 | GPIO_11 | GPIO_10 | GPIO_09 | GPIO_08 | GPIO_07 | GPIO_06 | GPIO_05 | GPIO_04 | GPIO_03 | GPIO_02 | GPIO_01 | GPIO_00 |
| | | | | | | | | | | | | | | | | |
| GPIO_DATA_DIR_SET1 | | | | FOR GPIO MODES ONLY - GPIOs GPIO_32 - GPIO_63 | | | | | | | | | Addr Base+0x5C | | | |
| $Offset_{16}$ | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| Base+0x5C | GPIO_63 | GPIO_62 | GPIO_61 | GPIO_60 | GPIO_59 | GPIO_58 | GPIO_57 | GPIO_56 | GPIO_55 | GPIO_54 | GPIO_53 | GPIO_52 | GPIO_51 | GPIO_50 | GPIO_49 | GPIO_48 |
| | | | | | | | | | | | | | | | | |
| $Offset_{16}$ | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| Base+0x5C | GPIO_47 | GPIO_46 | GPIO_45 | GPIO_44 | GPIO_43 | GPIO_42 | GPIO_41 | GPIO_40 | GPIO_39 | GPIO_38 | GPIO_37 | GPIO_36 | GPIO_35 | GPIO_34 | GPIO_33 | GPIO_32 |

**Table 11-19. GPIO Pad Direction Set Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 0-63 | **GPIO_DATA_DIR_SETx[31:0]** - These bits set corresponding bits in the GPIO_PAD_DIRx Registers.<br>**Note:** Setting a bit enables the GPIO as an output | Write-only. Writing a 1 sets the corresponding bit in the GPIO_DATAx Register. Any existing bits set to 1 in GPIO_DATAx remain at 1. |

## 11.7.12 GPIO Pad Direction Reset Registers (GPIO_PAD_DIR_RESET1 and GPIO_PAD_DIR_RESET0)

The GPIO Pad Direction Reset Registers are used to set specific bits to “0” in the GPIO Pad Direction Registers (set GPIO to input). Two registers (GPIO_PAD_DIR_RESET1 and GPIO_PAD_DIR_RESET0) are required to control the 64 possible GPIO.

- The pad direction reset registers are write-only.
- Any bits that are “0” in the present value of GPIO_PAD_DIR0 or GPIO_PAD_DIR1 will still be “0” after a write to its corresponding data reset register.
- Any bit written to a “1” in the pad direction reset register will cause that bit to be reset to “0” in its pad direction register.
- The final contents of the pad direction register are the logic “AND” of the previous pad direction register contents and the bitwise inverse of the pad direction reset register write data.

**NOTE**

The GPIO_PAD_DIR0 and GPIO_PAD_DIR1 registers can be written directly to program all bits at one time.

| GPIO_DATA_DIR_RESET0 | | | | FOR GPIO MODES ONLY - GPIOs GPIO_00 - GPIO_31 | | | | | | | | | Addr Base+0x64 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $Offset_{16}$ | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| Base+0x60 | GPIO_31 | GPIO_30 | GPIO_29 | GPIO_28 | GPIO_27 | GPIO_26 | GPIO_25 | GPIO_24 | GPIO_23 | GPIO_22 | GPIO_21 | GPIO_20 | GPIO_19 | GPIO_18 | GPIO_17 | GPIO_16 |
| | | | | | | | | | | | | | | | | |
| $Offset_{16}$ | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| Base+0x60 | GPIO_15 | GPIO_14 | GPIO_13 | GPIO_12 | GPIO_11 | GPIO_10 | GPIO_09 | GPIO_08 | GPIO_07 | GPIO_06 | GPIO_05 | GPIO_04 | GPIO_03 | GPIO_02 | GPIO_01 | GPIO_00 |
| | | | | | | | | | | | | | | | | |
| GPIO_DATA_DIR_RESET1 | | | | FOR GPIO MODES ONLY - GPIOs GPIO_32 - GPIO_63 | | | | | | | | | Addr Base+0x64 | | | |
| $Offset_{16}$ | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| Base+0x64 | GPIO_63 | GPIO_62 | GPIO_61 | GPIO_60 | GPIO_59 | GPIO_58 | GPIO_57 | GPIO_56 | GPIO_55 | GPIO_54 | GPIO_53 | GPIO_52 | GPIO_51 | GPIO_50 | GPIO_49 | GPIO_48 |
| | | | | | | | | | | | | | | | | |
| $Offset_{16}$ | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| Base+0x64 | GPIO_47 | GPIO_46 | GPIO_45 | GPIO_44 | GPIO_43 | GPIO_42 | GPIO_41 | GPIO_40 | GPIO_39 | GPIO_38 | GPIO_37 | GPIO_36 | GPIO_35 | GPIO_34 | GPIO_33 | GPIO_32 |

**Table 11-20. GPIO Pad Direction Reset Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 0-63 | **GPIO_DATA_DIR_RESETx[31:0]** - These bits reset corresponding bits in the GPIO_PAD_DIRx Registers.<br>**Note:** Resetting a bit enables the GPIO as an input | Write-only. Writing a 1 resets the corresponding bit in the GPIO_DATA_DIRx Register. Any existing bits set to 0 in GPIO_DATA_DIRx remain at 0. |

# Chapter 12
# Timer Module (TMR)

## 12.1 Overview

The Timer Module contains four identical counter/timer groups. Each 16-bit counter/timer group contains a prescaler, a counter, a load register, a hold register, a capture register, two compare registers, two status and control registers, and one control register. Each counter/timer can do input capture, output compare, and pulse-width modulation (PWM). There are four external pins (channels) each of which is mapped to a particular counter. These pins can be used as inputs or outputs as required by the selected mode of operation.

**NOTE**

- This description uses the terms “Timer” and “Counter” interchangeably because the counter/timers may perform either or both tasks.
- Although this chapter provides reference information on the TMR module, the user/programmer is directed toward the software utility entitled the “Timer (TMR) Driver”, see Appendix B, “MC1322x Software Driver Utilities”

## 12.2 Features

The TMR module has the following features:

- Four - 16 bit counters/timers.
- Count up/down.
- Counters can be cascaded.
- Programmable count modulo.
- Max count rate equals peripheral clock/2 for external clocks.
- Max count rate equals peripheral clock for internal clocks.
- Count once or repeatedly.
- Counters can be preloaded.
- Compare Registers can be preloaded with Compare Load feature
- Counters can share available 4 external input/output pins.
- Separate prescaler for each counter.
- Each counter has capture and compare capability.
- Programmable operation during debug mode.
- Input glitch filter (optional)

- Counting start can be synchronized across counters.

## 12.3 Block Diagram

A single channel timer/counter group within the TMR module is shown in Figure 12-1.

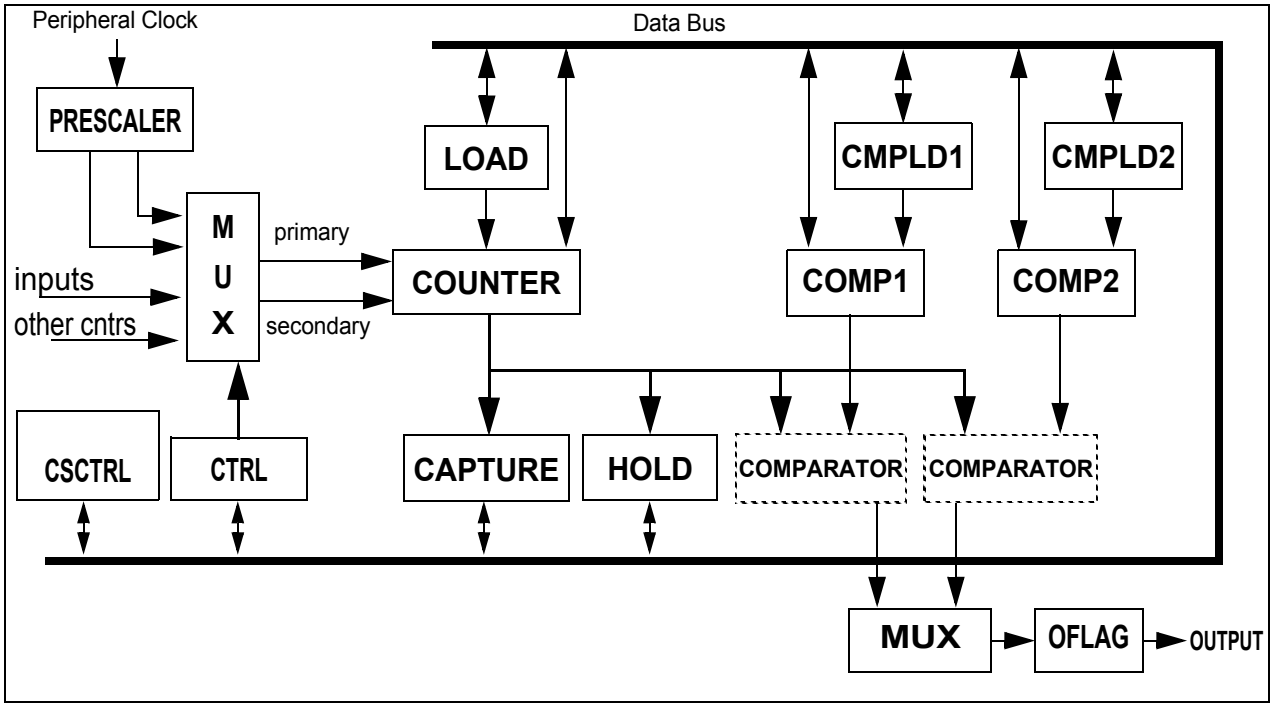



**Figure 12-1. Simplified Timer Block Diagram**

## 12.4 Functional Description

Figure 12-1 shows a TMR group block diagram. Each 16-bit counter/timer group contains a prescaler, a counter, a load register, a hold register, a capture register, two compare registers, two status and control registers, and one control register.

- The Counter block is the core of the group and provides the ability to count internal or external events.
  - The Load register provides the initialization value to the counter when the counter's terminal value has been reached.
  - The Hold Register captures the counter's value when other counters are being read. This feature supports the reading of cascaded counters.
  - The Capture register enables an external signal to take a "snap shot" of the counter's current value.
- Comparators provide control of the counter based on preset compare values. The COMP1 and COMP2 registers provide the values to which the counter is compared.
  - If a match occurs, the OFLAG signal can be set, cleared, or toggled.

  - At match time, an interrupt is generated if enabled, and the new compare value is loaded into the COMP1 or COMP2 registers from register CMPLD1 or CMPLD2 if enabled.
- The Prescaler provides different time bases useful for clocking the counter/timer from the Peripheral Clock
- The counter can also be clocked from the external TIO[3:0] IO pins or other channel counters
  - The TIO[3:0] IO pins are shareable
  - The counters can be cascaded for longer modulo counters.

The counter/timer has two basic modes of operation: It can count internal or external events, or it can count an internal clock source while an external input signal is asserted, thus timing the width of the external input signal.

- The counter can count the rising, falling, or both edges of the selected input pin.
- The counter can decode and count quadrature encoded input signals.
- The counter can count up and down using dual inputs in a “count with direction” format.
- The counter’s terminal count value (modulo) is programmable.
- The value that is loaded into the counter after reaching its terminal count is programmable.
- The counter can count repeatedly, or it can stop after completing one count cycle.
- The counter can be programmed to count to a programmed value and then immediately re initialize, or it can count through the compare value until the count “rolls over” to zero.

The external inputs to each counter/timer are shareable among each of the four counter/timers within the module. The external inputs can be used as:

- Count commands
- Timer commands
- They can trigger the current counter value to be “captured”
- They can be used to generate interrupt requests

The polarity of the external inputs are selectable.

The primary output of each timer/counter is the output signal OFLAG. The OFLAG output signal can be:

- Set, cleared, or toggled when the counter reaches the programmed value.
- The OFLAG output signal may be output to an external pin instead of having that pin serve as a timer input.
- The OFLAG output signal enables each counter to generate square waves, PWM, or pulse stream outputs.
- The polarity of the OFLAG output signal is selectable.

Any counter/timer can be assigned as a “Master”. A master’s compare signal can be broadcast to the other counter/timers within the module. The other counters can be configured to re initialize their counters and/or force their OFLAG output signals to predetermined values when a Master’s counter/timer compare event occurs.

### 12.4.1 External Signal Description

These pins can be independently configured to be either timer input sources or output flags.

The TMR module has 4 external Timer Input/Output signals TIO[3:0] that can be used as either inputs or outputs. These pins can be independently configured to be either timer input sources or output flags as suited to the selected timer function.

**Table 12-1. External Signal Properties**

| Name | I/O Type | Function | Reset State | Comment |
|---|---|---|---|---|
| TIO0 | Input/Output | Timer Input/Output | Input | Mapped to Timer 0. Can be used as input to any timer. Can be Timer 0 output only. |
| TIO1 | Input/Output | Timer Input/Output | Input | Mapped to Timer 1. Can be used as input to any timer. Can be Timer 1 output only. |
| TIO2 | Input/Output | Timer Input/Output | Input | Mapped to Timer 2. Can be used as input to any timer. Can be Timer 2 output only. |
| TIO3 | Input/Output | Timer Input/Output | Input | Mapped to Timer 3. Can be used as input to any timer. Can be Timer 3 output only. |

### 12.4.2 Clock Sources

The counters in the TMR module can have several clock sources as selected by the PRIMARY_CNT_SOURCE[3:0] field:

- The internal clock to the TMR blocks is the Peripheral Clock as determined by the XTAL_CLKDIV[5:0] field of the CRM System Control (SYS_CNTL) Register, see Section 5.9.1, “System Control (SYS_CNTL)”
  - Typical frequency is 24 MHz (same as the reference oscillator)
  - The PRIMARY_CNT_SOURCE[3:0] field allows the Peripheral Clock to be pre-scaled by a factor of 1 to 128
- The external IO pins TIO[3:0] can be selected as clock sources
- One counter output can be selected to toggle a second counter forming a longer concantenated counter. In this manner, 32-bit, 48-bit, or 64-bit counter configurations are possible.

### 12.4.3 Operational Modes

A counter/timer block has a plethora of operational modes. To best provide examples and usage guidelines, the modes are described in Section 12.7, “TMR Functional Modes”, after the register models are described in Section 12.6, “Register Descriptions”.

### 12.4.4 TMR Interrupt Requests

The TMR module has a single interrupt request signal (IRQ) that is connected to the interrupt controller. The TMR IRQ can be generated 20 individual sources, five from each of the four counters/channels. These are OR’ed together such that any can generate the interrupt request if enabled.

Table 12-2 lists the TMR interrupt sources and their characteristics.

**Table 12-2. TMR Interrupt Sources**

| Item | Ch | Status Bit (Register) | Mask Bit (Register) | Source Description | Interrupt Clear Mechanism |
|---|---|---|---|---|---|
| 1 | 0 | TCF (TMR0_SCTRL) | TCFIE (TMR0_SCTRL) | Successful compare occurred | Writing a “0” to the TCF bit clears the status. |
| 2 | 0 | TCF2 (TMR0_SCTRL) | TCF2EN (TMR0_SCTRL) | Successful Compare 2 occurred (Also reflected in TCF bit) | Writing a “0” to the TCF2 bit clears the status. |
| 3 | 0 | TCF1 (TMR0_SCTRL) | TCF1EN (TMR0_SCTRL) | Successful Compare 1 occurred (Also reflected in TCF bit) | Writing a “0” to the TCF1 bit clears the status. |
| 4 | 0 | TOF (TMR0_SCTRL) | TOFIE (TMR0_SCTRL) | Timer overflow occurred | Writing a “0” to the TOF bit clears the status. |
| 5 | 0 | IEF (TMR0_SCTRL) | IEFIE (TMR0_SCTRL) | Input clock edge has occurred | Writing a “0” to the IEF bit clears the status. |
| | | | | | |
| 6 | 1 | TCF (TMR1_SCTRL) | TCFIE (TMR1_SCTRL) | Successful compare occurred | Writing a “0” to the TCF bit clears the status. |
| 7 | 1 | TCF2 (TMR1_SCTRL) | TCF2EN (TMR1_SCTRL) | Successful Compare 2 occurred (Also reflected in TCF bit) | Writing a “0” to the TCF2 bit clears the status. |
| 8 | 1 | TCF1 (TMR1_SCTRL) | TCF1EN (TMR1_SCTRL) | Successful Compare 1 occurred (Also reflected in TCF bit) | Writing a “0” to the TCF1 bit clears the status. |
| 9 | 1 | TOF (TMR1_SCTRL) | TOFIE (TMR1_SCTRL) | Timer overflow occurred | Writing a “0” to the TOF bit clears the status. |
| 10 | 1 | IEF (TMR1_SCTRL) | IEFIE (TMR1_SCTRL) | Input clock edge has occurred | Writing a “0” to the IEF bit clears the status. |
| | | | | | |
| 11 | 2 | TCF (TMR2_SCTRL) | TCFIE (TMR2_SCTRL) | Successful compare occurred | Writing a “0” to the TCF bit clears the status. |
| 12 | 2 | TCF2 (TMR2_CSCTRL) | TCF2EN (TMR2_CSCTRL) | Successful Compare 2 occurred (Also reflected in TCF bit) | Writing a “0” to the TCF2 bit clears the status. |
| 13 | 2 | TCF1 (TMR2_SCTRL) | TCF1EN (TMR2_SCTRL) | Successful Compare 1 occurred (Also reflected in TCF bit) | Writing a “0” to the TCF1 bit clears the status. |
| 14 | 2 | TOF (TMR2_SCTRL) | TOFIE (TMR2_SCTRL) | Timer overflow occurred | Writing a “0” to the TOF bit clears the status. |
| 15 | 2 | IEF (TMR2_SCTRL) | IEFIE (TMR2_SCTRL) | Input clock edge has occurred | Writing a “0” to the IEF bit clears the status. |
| | | | | | |
| 16 | 3 | TCF (TMR3_SCTRL) | TCFIE (TMR3_SCTRL) | Successful compare occurred | Writing a “0” to the TCF bit clears the status. |
| 17 | 3 | TCF2 (TMR3_CSCTRL) | TCF2EN (TMR3_CSCTRL) | Successful Compare 2 occurred (Also reflected in TCF bit) | Writing a “0” to the TCF2 bit clears the status. |

| Item | Ch | Status Bit (Register) | Mask Bit (Register) | Source Description | Interrupt Clear Mechanism |
|---|---|---|---|---|---|
| 18 | 3 | TCF1 (TMR3_CSCTRL) | TCF1EN (TMR3_CSCTRL) | Successful Compare 1 occurred (Also reflected in TCF bit) | Writing a “0” to the TCF1 bit clears the status. |
| 19 | 3 | TOF (TMR3_SCTRL) | TOFIE (TMR3_SCTRL) | Timer overflow occurred | Writing a “0” to the TOF bit clears the status. |
| 20 | 3 | IEF (TMR3_SCTRL) | IEFIE (TMR3_SCTRL) | Input clock edge has occurred | Writing a “0” to the IEF bit clears the status. |

#### 12.4.4.1 Timer Compare Interrupts

These interrupts (TCF) are generated when a successful compare occurs between a counter and one of it’s compare registers while the Timer Compare Flag Interrupt Enable is set in the TMR_SCTRL. These interrupts are cleared by writing a zero to the TCF bit in the appropriate TMR_SCTRL.

When a timer compare interrupt is set in the TMR_SCTRL, one of the following two interrupts will also be asserted.

##### 12.4.4.1.1 Timer Compare 1 Interrupts (with Compare Load Feature)

These interrupts are generated when a successful compare occurs between a counter and its COMP1 register while the Timer Compare 1 Interrupt Enable is set in the CSCTRL register. These interrupts are cleared by writing a zero to the TCF1 bit in the appropriate CSCTRL.

##### 12.4.4.1.2 Timer Compare 2 Interrupts (with Compare Load Feature)

These interrupts are generated when a successful compare occurs between a counter and its COMP2 register while the Timer Compare 2 Interrupt Enable is set in the CSCTRL register. These interrupts are cleared by writing a zero to the TCF1 bit in the appropriate CSCTRL.

### 12.4.5 Timer Overflow Interrupts

These interrupts are generated when a counter rolls over its maximum value while the Timer Overflow Flag Interrupt Enable bit is set in the TMR_SCTRL. These interrupts are cleared by writing zero to the TOF bit of the appropriate TMR_SCTRL.

### 12.4.6 Timer Input Edge Interrupts

These interrupts are generated by a transition of the input signal (either positive or negative depending on IPS setting) while the Input Edge Flag Interrupt Enable bit is set in the TMR_SCTRL. These interrupts are cleared by writing a zero to the IEF bit of the appropriate TMR_SCTRL.

## 12.5 TMR Register Memory Map

The TMR module is programmed via a set of memory-mapped registers and their respective offset addresses are listed in Table 12-3.

- Base address for the TMR Module is **0x8000_7000**
- All registers are 16 bits wide with 16-bit access only. Note register address offset is 0x2, not 0x4.

**Table 12-3. Timer Module Memory Map**

| Address | Register Name | Access Type | Access Width |
|---|---|---|---|
| Base + 0x0 | Timer Channel 0 Compare Register 1 (TMR0_COMP1) | Read/Write | 16-bit only |
| Base + 0x2 | Timer Channel 0 Compare Register 2 (TMR0_COMP2) | Read/Write | 16-bit only |
| Base + 0x4 | Timer Channel 0 Capture Register (TMR0_CAPT) | Read/Write | 16-bit only |
| Base + 0x6 | Timer Channel 0 Load Register (TMR0_LOAD) | Read/Write | 16-bit only |
| Base + 0x8 | Timer Channel 0 Hold Register (TMR0_HOLD) | Read/Write | 16-bit only |
| Base + 0xa | Timer Channel 0 Counter Register (TMR0_CNTR) | Read/Write | 16-bit only |
| Base + 0xc | Timer Channel 0 Control Register (TMR0_CTRL) | Read/Write | 16-bit only |
| Base + 0xe | Timer Channel 0 Status and Control Register (TMR0_SCTRL) | Read/Write | 16-bit only |
| Base + 0x10 | Timer Channel 0 Comparator Load Register 1 (TMR0_CMPLD1) | Read/Write | 16-bit only |
| Base + 0x12 | Timer Channel 0 Comparator Load Register 2 (TMR0_CMPLD2) | Read/Write | 16-bit only |
| Base + 0x14 | Timer Channel 0 Comparator Status and Control Register (TMR0_CSCTRL) | Read/Write | 16-bit only |
| Base + 0x16 - Base + 0x1C | Reserved | | |
| Base + 0x1E | Timer Channel Enable Register (TMR0_ENBL) | Read/Write | 16-bit only |
| Base + 0x20 | Timer Channel 1 Compare Register 1 (TMR1_COMP1) | Read/Write | 16-bit only |
| Base + 0x22 | Timer Channel 1 Compare Register 2 (TMR1_COMP2) | Read/Write | 16-bit only |
| Base + 0x24 | Timer Channel 1 Capture Register (TMR1_CAPT) | Read/Write | 16-bit only |
| Base + 0x26 | Timer Channel 1 Load Register (TMR1_LOAD) | Read/Write | 16-bit only |
| Base + 0x28 | Timer Channel 1 Hold Register (TMR1_HOLD) | Read/Write | 16-bit only |
| Base + 0x2A | Timer Channel 1 Counter Register (TMR1_CNTR) | Read/Write | 16-bit only |
| Base + 0x2C | Timer Channel 1 Control Register (TMR1_CTRL) | Read/Write | 16-bit only |
| Base + 0x2E | Timer Channel 1 Status and Control Register (TMR1_SCTRL) | Read/Write | 16-bit only |
| Base + 0x30 | Timer Channel 1 Comparator Load Register 1 (TMR1_CMPLD1) | Read/Write | 16-bit only |
| Base + 0x32 | Timer Channel 1 Comparator Load Register 2 (TMR1_CMPLD2) | Read/Write | 16-bit only |
| Base + 0x34 | Timer Channel 1 Comparator Status and Control Register (TMR1_CSCTRL) | Read/Write | 16-bit only |
| Base + 0x36 - Base + 0x3C | Reserved | | |
| Base + 0x40 | Timer Channel 2 Compare Register 1 (TMR2_COMP1) | Read/Write | 16-bit only |
| Base + 0x42 | Timer Channel 2 Compare Register 2 (TMR2_COMP2) | Read/Write | 16-bit only |
| Base + 0x44 | Timer Channel 2 Capture Register (TMR2_CAPT) | Read/Write | 16-bit only |
| Base + 0x46 | Timer Channel 2 Load Register (TMR2_LOAD) | Read/Write | 16-bit only |
| Base + 0x48 | Timer Channel 2 Hold Register (TMR2_HOLD) | Read/Write | 16-bit only |
| Base + 0x4A | Timer Channel 2 Counter Register (TMR2_CNTR) | Read/Write | 16-bit only |
| Base + 0x4C | Timer Channel 2 Control Register (TMR2_CTRL) | Read/Write | 16-bit only |
| Base + 0x4e | Timer Channel 2 Status and Control Register (TMR2_SCTRL) | Read/Write | 16-bit only |
| Base + 0x50 | Timer Channel 2 Comparator Load Register 1 (TMR2_CMPLD1) | Read/Write | 16-bit only |
| Base + 0x52 | Timer Channel 2 Comparator Load Register 2 (TMR2_CMPLD2) | Read/Write | 16-bit only |
| Base + 0x54 | Timer Channel 2 Comparator Status and Control Register (TMR2_CSCTRL) | Read/Write | 16-bit only |
| Base + 0x56 - Base + 0x5C | Reserved | | |

**Table 12-3. Timer Module Memory Map**

| Address | Register Name | Access Type | Access Width |
|---|---|---|---|
| Base + 0x60 | Timer Channel 3 Compare Register 1 (TMR3_COMP1) | Read/Write | 16-bit only |
| Base + 0x62 | Timer Channel 3 Compare Register 2 (TMR3_COMP2) | Read/Write | 16-bit only |
| Base + 0x64 | Timer Channel 3 Capture Register (TMR3_CAPT) | Read/Write | 16-bit only |
| Base + 0x66 | Timer Channel 3 Load Register (TMR3_LOAD) | Read/Write | 16-bit only |
| Base + 0x68 | Timer Channel 3 Hold Register (TMR3_HOLD) | Read/Write | 16-bit only |
| Base + 0x6A | Timer Channel 3 Counter Register (TMR3_CNTR) | Read/Write | 16-bit only |
| Base + 0x6C | Timer Channel 3 Control Register (TMR3_CTRL) | Read/Write | 16-bit only |
| Base + 0x6E | Timer Channel 3 Status and Control Register (TMR3_SCTRL) | Read/Write | 16-bit only |
| Base + 0x70 | Timer Channel 3 Comparator Load Register 1 (TMR3_CMPLD1) | Read/Write | 16-bit only |
| Base + 0x72 | Timer Channel 3 Comparator Load Register 2 (TMR3_CMPLD2) | Read/Write | 16-bit only |
| Base + 0x74 | Timer Channel 3 Comparator Status and Control Register (TMR3_CSCTRL) | Read/Write | 16-bit only |
| Base + 0x76 - Base + 0x7E | Reserved | | |

## 12.6 Register Descriptions

The following sections provide descriptions of the TMR registers. Each timer counter/timer has a set of control and status registers. These register sets are symmetric in that each set has the same registers and model.

### 12.6.1 TMR Compare Registers

Each counter has two 16-bit registers that are used to compare count values, i.e., Compare Register 1 and Compare Register 2:

- Compare Register 1 is used when the counter is *counting up*.
- Compare Register 2 is used when the counter is *counting down*.

These registers provide the values to which the counter is compared. If a match occurs, the OFLAG signal can be set, cleared, or toggled. At match time, an interrupt is generated if enabled, and the new compare value is loaded into the COMP1 or COMP2 registers from CMPLD1 and CMPLD2 if enabled.

#### 12.6.1.1 TMR Compare Registers 1 (TMR0_COMP1:TMR3_COMP1)

The Timer Compare Registers 1 are count up compare and include:

- Timer Channel 0 Compare Register 1 (TMR0_COMP1) - Address Base + 0x00
- Timer Channel 1 Compare Register 1 (TMR1_COMP1) - Address Base + 0x20
- Timer Channel 2 Compare Register 1 (TMR2_COMP1) - Address Base + 0x40
- Timer Channel 3 Compare Register 1 (TMR3_COMP1) - Address Base + x60

| TMRX_COMP1 | | | | | | | | | | | | | Addr Base+0xX0 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | COMPARE_VALUE | | | | | | | | | | | | | | | |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 12-4. TMRX_COMP1 Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 15-0 | **COMPARE_VALUE[15:0] -** This field is a 16-bit compare value used with the counter when counting up | |

### 12.6.1.2 TMR Compare Registers 2 (TMR0_COMP2:TMR3_COMP2)

The Timer Compare Registers 2 are count down compare and include:

- Timer Channel 0 Compare Register 2 (TMR0_COMP2) - Address Base + 0x02
- Timer Channel 1 Compare Register 2 (TMR1_COMP2) - Address Base + 0x22
- Timer Channel 2 Compare Register 2 (TMR2_COMP2) - Address Base + 0x42
- Timer Channel 3 Compare Register 2 (TMR3_COMP2) - Address Base + 0x62

| TMRX_COMP2 | | | | | | | | | | | | | Addr Base+0xX2 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | COMPARE_VALUE | | | | | | | | | | | | | | | |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 12-5. TMRX_COMP2 Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 15-0 | **COMPARE_VALUE[15:0] -** This field is a 16-bit compare value used with the counter when counting down | |

## 12.6.2 TMR Capture Registers (TMRX_CAPT)

Each counter has a 16-bit timer capture register used to enable an external signal to take a “snapshot” of the counter’s current value:

- Timer Channel 0 Capture Register (TMR0_CAPT) - Address Base + 0x04

- Timer Channel 1 Capture Register (TMR1_CAPT) - Address Base + 0x24
- Timer Channel 2 Capture Register (TMR2_CAPT) - Address Base + 0x44
- Timer Channel 3 Capture Register (TMR3_CAPT) - Address Base + 0x64

| TMRX_CAPT | | | | | | | | | | | | | Addr Base+0xX4 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | CAPTURE_VALUE | | | | | | | | | | | | | | | |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 12-6. TMRX_CAPT Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 15-0 | **CAPTURE_VALUE[15:0] -** This field is a 16-bit value used to re-enable an external signal to take a “snapshot” of the counter’s current value | |

## 12.6.3 TMR Load Registers (TMRX_LOAD)

Each counter has a 16-bit timer load register used to provide the initialization value to the counter when the counter’s terminal value has been reached:

- Timer Channel 0 Load Register (TMR0_LOAD) - Address Base + 0x06
- Timer Channel 1 Load Register (TMR1_LOAD) - Address Base + 0x26
- Timer Channel 2 Load Register (TMR2_LOAD) - Address Base + 0x46
- Timer Channel 3 Load Register (TMR3_LOAD) - Address Base + 0x66

| TMRX_LOAD | | | | | | | | | | | | | Addr Base+0xX6 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | LOAD_VALUE | | | | | | | | | | | | | | | |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 12-7. TMRX_LOAD Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 15-0 | **LOAD_VALUE[15:0] -** This field is a 16-bit value used to load/initialize the counter | |

## 12.6.4 TMR Hold Registers (TMRX_HOLD)

Each counter has a 16-bit timer hold register used to capture the counter's value when other counters are being read. This feature supports the reading of cascaded counters:

- Timer Channel 0 Hold Register (TMR0_HOLD) - Address Base + 0x08
- Timer Channel 1 Hold Register (TMR1_HOLD) - Address Base + 0x28
- Timer Channel 2 Hold Register (TMR2_HOLD) - Address Base + 0x48
- Timer Channel 3 Hold Register (TMR3_HOLD) - Address Base + 0x68

**NOTE**

- The hold registers should only be used when COUNT_MODE[2:0] = 111 is enabled, i.e., synchronous counter mode.
- To enable latching of the counters' HOLD registers, read a counter CNTR register. This action activates a load of the HOLD registers which can then be accessed as required to get a strobed value of the full multi-stage cascaded counter.

| **TMRX_HOLD** | | | | | | | | | | | | | **Addr Base+0xX8** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | HOLD_VALUE | | | | | | | | | | | | | | | |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 12-8. TMRX_HOLD Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 15-0 | **HOLD_VALUE[15:0] -** This field is a 16-bit value used to capture the counter's value when other counters are being read | |

## 12.6.5 TMR Counter Registers (TMRX_CNTR)

Each counter has a 16-bit counter register used to count internal or external events:

- Timer Channel 0 Counter Register (TMR0_CNTR) - Address Base + 0x0A
- Timer Channel 1 Counter Register (TMR1_CNTR) - Address Base + 0x2A
- Timer Channel 2 Counter Register (TMR2_CNTR) - Address Base + 0x4A
- Timer Channel 3 Counter Register (TMR3_CNTR) - Address Base + 0x6A

| TMRX_CNTR | | | | | | | | | | | | | Addr Base+0xXA | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | COUNTER | | | | | | | | | | | | | | | |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 12-9. TMRX_CNTR Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 15-0 | **COUNTER[15:0] -** This field is a 16-bit value used to provide a count of internal or external events | |

## 12.6.6 TMR Control Registers (TMRX_CTRL)

Each counter has a 16-bit primary control register. These fields control operation and mode of the counter:

- Timer Channel 0 Control Register (TMR0_CTRL) - Address Base + 0x0C
- Timer Channel 1 Control Register (TMR1_CTRL) - Address Base + 0x2C
- Timer Channel 2 Control Register (TMR2_CTRL) - Address Base + 0x4C
- Timer Channel 3 Control Register (TMR3_CTRL) - Address Base + 0x6C

| TMRX_CTRL | | | | | | | | | | | | | Addr Base+0xXC | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | COUNT_MODE | | | PRIMARY_CNT_ SOURCE | | | | SECONDARY _CNT_ SOURCE | | ONCE | LENGTH | DIR | CO_ INIT | OUTPUT_MODE | | |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 12-10. TMRX_CTRL Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 15-13 | **COUNT_MODE[2:0] -** This 3-bit field controls the basic count mode and behavior of the counter | See Table 12-11 |
| 12-9 | **PRIMARY_CNT_SOURCE[3:0] -** This 4-bit field selects the primary count source. The selected input can be:<br>• External pins<br>• Other counter outputs (for concantenating counters)<br>• Peripheral Clock (set by CRM)<br>• Prescaled Peripheral Clock | See Table 12-12 |
| 8-7 | **SECONDARY_CNT_SOURCE[1:0] -** This field identifies the external input pin to be used as a count command, or timer command.<br>• The selected input can trigger the timer to capture the current value of the CNTR register.<br>• The selected input can also be used to specify the count direction.<br>• The polarity of the signal can be inverted by the IPS bit of the SCTRL register. | See Table 12-13 |
| 6 | **ONCE -** Count Once bit. This bit selects continuous or one shot counting mode. | 1 = Count until compare and then stop -<br>• If counting up, successful compare occurs when the counter reaches a COMP1 value.<br>• If counting down, successful compare occurs when the counter reaches a COMP2 value<br>0 = Count repeatedly (default) |
| 5 | **LENGTH -** Count Length bit. This bit determines whether the counter counts to the compare value and then re-initializes itself to the value specified in the load register, or the counter continues counting past the compare value, to the binary rollover. | 1 = Count until compare, then re initialize. -<br>• If counting up, successful compare occurs when the counter reaches a COMP1 value.<br>• If counting down, successful compare occurs when the counter reaches a COMP2 value<br>• When OUTPUT_MODE 0x4 is used, alternating values of COMP1 and COMP2 are used to generate successful comparisons. For example, the counter counts until COMP1 value is reached, re-initializes, then counts until COMP2 value is reached, re-initializes, then counts until COMP1 value is reached, etc.<br>0 = Rollover (default) |
| 4 | **DIR -** Count Direction bit. This bit selects either the normal count direction up, or the reverse direction, down. | 1 = Count down<br>0 = Count up (default) |
| 3 | **CO_INIT -** Co-Channel Initialization bit. This bit enables another counter/timer within the module to force the re-initialization of this counter/timer when it has an active compare event. | 1 = Co-Channel counter/timers force a re-initialization of this counter/timer<br>0 = Co-Channel counter/timers can not force a re-initialization of this counter/timer |
| 2-0 | **OUTPUT_MODE[2:0] -** This field determines the mode of operation for the OFLAG output signal. | See Table 12-14 |

**Table 12-11. Timer Count Mode**

| COUNT_MODE[2:0] Value | Operation |
|---|---|
| 000 | No Operation |
| 001 | Count rising edges of primary source[1] |
| 010 | Count rising and falling edges of primary source[2] |
| 011 | Count rising edges of primary source while secondary input high active |
| 100 | Quadrature Count mode, uses primary and secondary sources |
| 101 | Count primary source rising edges, secondary source specifies direction (1 = minus)[3] |
| 110 | Edge of secondary source triggers primary count till compare |
| 111 | Synchronous counter mode, up/down[4] |

1 Rising Edges counted only when IPS = 0. Falling edges counted when IPS = 1. If primary count source is IP bus clock divide by 1 only rising edges are counted regardless of IPS value.

2 Peripheral Bus clock divide by 1 can not be used as a primary count source in rising/falling edge count mode.

3 Rising Edges counted only when IPS = 0. Falling edges counted when IPS = 1.

4 Primary count source must be set to one of the counter outputs.

**Table 12-12. Timer Primary Count Source**

| Value [1] | Meaning |
|---|---|
| 0000 | Counter 0 input pin |
| 0001 | Counter 1 input pin |
| 0010 | Counter 2 input pin |
| 0011 | Counter 3 input pin |
| 0100 | Counter 0 output |
| 0101 | Counter 1 output |
| 0110 | Counter 2 output |
| 0111 | Counter 3 output |
| 1000 | Peripheral Clock divided by 1 prescaler |
| 1001 | Peripheral Clock divided by 2 prescaler |
| 1010 | Peripheral Clock divided by 4 prescaler |
| 1011 | Peripheral Clock divided by 8 prescaler |
| 1100 | Peripheral Clock divided by 16 prescaler |
| 1101 | Peripheral Clock divided by 32 prescaler |
| 1110 | Peripheral Clock divided by 64 prescaler |
| 1111 | Peripheral Clock divided by 128 prescaler |

1 A timer selecting its own output for input is not a legal choice. The result is no counting.

**Table 12-13. Timer Secondary Count Source**

| Value | Meaning |
|---|---|
| 00 | Counter 0 input pin |
| 01 | Counter 1 input pin |
| 10 | Counter 2 input pin |
| 11 | Counter 3 input pin |

**Table 12-14. Timer Output Mode**

| Value | Meaning |
|---|---|
| 000 | Asserted while counter is active |
| 001 | Clear OFLAG output on successful compare |
| 010 | Set OFLAG output on successful compare |
| 011 | Toggle OFLAG output on successful compare |
| 100 | Toggle OFLAG output using alternating compare registers |
| 101 | Set on compare, cleared on secondary source input edge |
| 110 | Set on compare, cleared on counter rollover |
| 111 | Enable gated clock output while counter is active |

## 12.6.7 TMR Status and Control Registers (TMRX_SCTRL)

Each counter has a 16-bit status and secondary control register. These bits primarily report status of the counter and enable interrupt request sources:

- Timer Channel 0 Status and Control Register (TMR0_SCTRL) - Address Base + 0x0E
- Timer Channel 1 Status and Control Register (TMR1_SCTRL) - Address Base + 0x2E
- Timer Channel 2 Status and Control Register (TMR2_SCTRL) - Address Base + 0x4E
- Timer Channel 3 Status and Control Register (TMR3_SCTRL) - Address Base + 0x6E

| **TMRX_SCTRL** | | | | | | | | | | | | | **Addr Base+0xXE** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | TCF | TCFIE | TOF | TOFIE | IEF | IEFIE | IPS | INPUT | CAPTURE_MODE | | MSTR | EEOF | VAL | FORCE | OPS | OEN |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 12-15. TMRX_SCTRL Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 15 | TCF - Timer Compare Flag. This bit indicates when a successful compare occurred.<br>**Note:** TCF1 and/or TCF2 in CSCTRL Register will also be set. See Section 12.6.9, "TMR Comparator Status and Control Registers (TMRX_CSCTRL)" | 1 = Successful compare - this bit is cleared by writing a zero to this bit location.<br>0 = No compare |
| 14 | **TCFIE -** Timer Compare Flag Interrupt Enable. This bit enables an interrupt request when the TCF bit is set. | 1 = Interrupt request enabled<br>0 = Interrupt request disabled (default) |
| 13 | **TOF -** Timer Overflow Flag. This bit indicates when the counter rolls over its maximum value 0xFFFF or 0x0000 (depending on count direction). | 1 = Rollover - - this bit is cleared by writing a zero to this bit location.<br>0 = No rollover |
| 12 | **TOFIE -** Timer Overflow Flag Interrupt Enable. This bit enables an interrupt request when the TOF bit is set. | 1 = Interrupt request enabled<br>0 = Interrupt request disabled (default) |
| 11 | **IEF -** Input Edge Flag. This bit indicates when a positive input transition occurs (on an input selected as a secondary count source) while the counter is enabled | 1 = Positive transition - this bit is cleared by writing a zero to this bit location.<br>• Setting the input polarity select (IPS) bit enables the detection of negative input edge transitions detection.<br>• The control register's secondary count source determines which external input pin is monitored by the detection circuitry.<br>0 = No transition |
| 10 | **IEFIE -** Input Edge Flag Interrupt Enable. This bit enables an interrupt request when the IEF bit is set. | 1 = Interrupt request enabled<br>0 = Interrupt request disabled (default) |
| 9 | **IPS -** Input Polarity Select. This bit enables inversion the input signal polarity. | 1 = Inversion enabled<br>0 = Inversion disabled (default) |
| 8 | **INPUT -** External Input Signal. This bit reflects the current state of the external input pin selected via the Secondary Count Source after application of the IPS bit. | Read only |
| 7-6 | **CAPTURE_MODE[1:0] -** Input Capture Mode. This field specifies the operation of the capture register as well as the operation of the input edge flag. The input source is the Secondary Count Source. | See Table 12-16 |
| 5 | **MSTR -** Master Mode enable. This bit enables the compare function's output to be broadcast to the other counter/timers in the module. This signal then can be used to re-initialize the other counters and/or force their OFLAG signal outputs | 1 = Master mode set<br>0 = Master Mode not set (default) |
| 4 | **EEOF -** Enable External OFLAG Force bit. This bit enables the compare from another counter/timer within the same module to force the state of this counter's OFLAG output signal. | 1 = OFLAG Force enabled<br>0= OFLAG Force not enabled (default) |

| Bit Number | Description | Operation |
|---|---|---|
| 3 | **VAL -** Forced OFLAG Value. This bit determines the value of the OFLAG output signal when a software triggered FORCE command. | Default = 0 |
| 2 | **FORCE -** Force the OFLAG output. This bit forces the current value of the VAL bit to be written to the OFLAG output.<br>• The VAL and FORCE bits can be written simultaneously in a single write operation.<br>• Write this bit only if the counter is disabled.<br>• Setting this bit while the counter is enabled may yield unpredictable results. | • Write only.<br>• Always reads as a zero. |
| 1 | **OPS -** Output Polarity Select. This bit determines the polarity of the OFLAG output signal. | 1 = Inverted polarity<br>0 = True polarity (default) |
| 0 | **OEN -** Output Enable. This bit determines the direction of the external pin mapped to this counter. | 1 = OFLAG output signal will be driven on the external pin.<br>• Other timer groups using this external pin as their input will see the driven value.<br>• The polarity of the signal will be determined by the OPS bit.<br>0 = The external pin is configured as an input (default). |

**Table 12-16. Timer Input Capture Mode**

| Capture_Mode{1:0] | IPS | Meaning |
|---|---|---|
| 00 | X | Capture function is disabled. |
| 01 | 0 | Load Capture register on rising edge of input |
| | 1 | Load Capture register on falling edge of input |
| 10 | 0 | Load Capture register on falling edge of input |
| | 1 | Load Capture register on rising edge of input |
| 11 | X | Load Capture register on both edges of input |

## 12.6.8 TMR Comparator Load Registers

Each counter has two comparators, and each comparator has a 16-bit “Comparator Load” register that provides a preset value that allows direct updating of the compare value without software intervention:

- Compare Load Register 1 is used for Comparator 1.
- Compare Load Register 2 is used for Comparator 2.

These registers offer a high degree of flexibility for loading compare registers with user-defined values on different compare events, and provide the values to be loaded. If a match occurs, an interrupt is generated if enabled, and the new compare value is loaded into the COMP1 or COMP2 registers from CMPLD1 and CMPLD2 if enabled.

The compare load feature is useful in variable frequency PWM mode.

### 12.6.8.1 TMR Comparator Load Register 1 (TMRX_CMPLD1)

Each Timer Compare Load Register 1 is the load value for Comparator 1 of the associated channel:

- Timer Channel 0 Comparator Load Register 1 (TMR0_CMPLD1) - Address Base + 0x10
- Timer Channel 1 Comparator Load Register 1 (TMR1_CMPLD1) - Address Base + 0x30
- Timer Channel 2 Comparator Load Register 1 (TMR2_CMPLD1) - Address Base + 0x50
- Timer Channel 3 Comparator Load Register 1 (TMR3_CMPLD1) - Address Base + 0x70

| **TMRX_CMPLD1** | | | | | | | | | | | | | **Addr Base+0xX0** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | COMPARE_LOAD1 | | | | | | | | | | | | | | | |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 12-17. TMRX_CMPLD1 Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 15-0 | **COMPARE_LOAD1[15:0] -** This field is a 16-bit compare value to be loaded into Comparator 1 based on a programmed timer event | See _____ |

### 12.6.8.2 TMR Comparator Load Register 2 (TMRX_CMPLD2)

Each Timer Compare Load Register 2 is the load value for Comparator 2 of the associated channel:

- Timer Channel 0 Comparator Load Register 1 (TMR0_CMPLD2) - Address Base + 0x12
- Timer Channel 1 Comparator Load Register 1 (TMR1_CMPLD2) - Address Base + 0x32
- Timer Channel 2 Comparator Load Register 1 (TMR2_CMPLD2) - Address Base + 0x52
- Timer Channel 3 Comparator Load Register 1 (TMR3_CMPLD2) - Address Base + 0x72

| **TMRX_CMPLD2** | | | | | | | | | | | | | **Addr Base+0xX0** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | COMPARE_LOAD2 | | | | | | | | | | | | | | | |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 12-18. TMRX_CMPLD2 Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 15-0 | **COMPARE_LOAD2[15:0] -** This field is a 16-bit compare value to be loaded into Comparator 2 based on a programmed timer event. | See _____ |

## 12.6.9 TMR Comparator Status and Control Registers (TMRX_CSCTRL)

Each counter has a 16-bit status and control register for the comparators:

- Timer Channel 0 Comparator Status and Control Register (TMR0_CSCTRL) - Addr Base + 0x14
- Timer Channel 1 Comparator Status and Control Register (TMR1_CSCTRL) - Addr Base + 0x34
- Timer Channel 2 Comparator Status and Control Register (TMR2_CSCTRL) - Addr Base + 0x54
- Timer Channel 3 Comparator Status and Control Register (TMR3_CSCTRL) - Addr Base + 0x74

| **TMRX_CSCTRL** | | | | | | | | | | | | | **Addr Base+0xX4** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | DBG_EN | | FILT_EN | Reserved | | | | | TCF2EN | TCF1EN | TCF2 | TCF1 | CL2 | | CL1 | |
| TYPE | rw | rw | rw | r | r | r | r | r | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 12-19. TMRX_CSCTRL Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 15-14 | **DBG_EN[1:0] -** Debug Actions Enable. This field enables the TMR module to perform certain actions in response to the chip entering debug mode. | 00 = Normal operation (during debug; default)<br>01 = Halt TMR counter during debug<br>10 = Force TMR output to logic 0 (prior to consideration of the OPS bit)<br>11 = Do Both: halt counter and force output to 0 during debug mode. |
| 13 | **FILT_EN -** Input Filter Enable. Each of the four external inputs to the TMR has a 4-bit shift register to act as a glitch filter:<br>• Three consecutive samples must match before the value is passed on to the counter.<br>• Assuming 24MHz operation, this will filter out pulses less than 83 nsecs wide.<br>• This control bit enables all four filters. | 1 = Input filters enabled<br>0 = Input filters disabled (default) |
| 12-8 | **Reserved** | Read only |

| Bit Number | Description | Operation |
|---|---|---|
| 7 | **TCF2EN -** Timer Compare 2 Interrupt Enable. This bit enables an interrupt request when the TCF2 bit is set.. | 1 = Interrupt request enabled<br>0 = Interrupt request disabled (default) |
| 6 | **TCF1EN -** Timer Compare 1 Interrupt Enable. This bit enables an interrupt request when the TCF1 bit is set. | 1 = Interrupt request enabled<br>0 = Interrupt request disabled (default) |
| 5 | **TCF2 -** Timer Compare 2 Status. This bit indicates a successful comparison between the timer and COMP2 register has occurred.<br>**Note:** This bit will also set TCF in SCTRL Register. | 1 = Successful compare - this bit is cleared by writing a zero to this bit location.<br>0 = No compare |
| 4 | **TCF1 -** Timer Compare 1Status. This bit indicates a successful comparison between the timer and COMP1 register has occurred.<br>**Note:** This bit will also set TCF in SCTRL Register. | 1 = Successful compare - this bit is cleared by writing a zero to this bit location.<br>0 = No compare |
| 3-2 | **CL2[1:0] -** Compare Load Control 2. This field controls when COMP2 is preloaded with the value from CMPLD2. | 00 = No preload (default)<br>01 = Load w/successful compare with the value in COMP1<br>10 = Load w/successful compare with the value in COMP2<br>11 = Reserved |
| 1-0 | **CL1[1:0] -** Compare Load Control 1. This field controls when COMP1 is preloaded with the value from CMPLD1. | 00 = No preload (default)<br>01 = Load w/successful compare with the value in COMP1<br>10 = Load w/successful compare with the value in COMP2<br>11 = Reserved |

## 12.6.10 TMR Channel Enable Register (TMR0_ENBL)

Timer Channel Enable Register (TMR0_ENBL) enables/disables each counter/channel independently through a control bit.

| **TMR0_ENBL** | | | | | | | | | | | | | **Addr Base+0x1E** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | Reserved | | | | | | | | | | | | ENBL | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

**Table 12-20. TMR0_ENBL Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 15-4 | **Reserved** | Read only |
| 3-0 | **ENBL[3:0] -** Timer Channel Enable. These bits enable the prescaler (if it is being used) and counter in each channel.<br>• Multiple ENBL bits can be set at the same time to synchronize the start of separate counters.<br>• If an ENBL bit is set, the corresponding channel will start the counter as soon as the COUNT_MODE field has a value other than 0.<br>• When an ENBL bit is clear, the corresponding counter maintains its current value. | 1= Timer Channel enabled (default)<br>0 = Timer Channel disabled<br><br>• ENBL3 enables Channel 3<br>• ENBL2 enables Channel 2<br>• ENBL1 enables Channel 1<br>• ENBL0 enables Channel 0 |

## 12.7 TMR Functional Modes

The operation of a counter/timer block is controlled primarily through its CTRL and CSCTRL control registers:

- The CTRL Register fields control count mode, count sources, continuous vs. single-shot operation, count-to-compare vs. continuous, count direction, co-channel operation, and operation of the OFLAG.
- The CSCTRL Register field control comparator operation; loading of COMP1 and COMP2
- Counters must also be enabled in the TMR0_ENBL Register.

Additional functionality includes:

- Input filters are enabled in the CSCTRL register
- If external IO are selected as count source
  - The selected external count signals are sampled at the TMR's base clock rate and then run to the input filter/transition detector.
  - The maximum count rate is one-half of the TMR's Peripheral Clock rate.
- Internal clock sources can be used to clock the counters at the Peripheral Clock rate.
- If a counter is programmed to count to a specific value and then stop, the Count Mode in the CTRL register is cleared when the count terminates.

Descriptions of functional modes are given in the following sections. Code snippets are used for illustration.

**NOTE**

Again, the user/programmer is directed toward the software utility entitled the "Timer (TMR) Driver", see Appendix B, "MC1322x Software Driver Utilities"

### 12.7.1 STOP Mode

If the COUNT_MODE[2:0] field is set to ‘000’, the counter is disabled:

- No counting occurs.
- This also disables the interrupt requests caused by input transitions on a selected input pin.

### 12.7.2 Single-Edge Count Mode

If the COUNT_MODE[2:0] field is set to ‘001’, the counter will count the rising edges of the selected primary clock source.

- This mode is useful for generating periodic interrupts for timing purposes, or counting external events (based on transitions on the external pins).
- If the selected input is inverted by setting the IPS bit (Input Polarity Select), then the negative edge of the selected external input signal is counted.

Two code examples are given for using single-edge count mode:

- Figure 12-2 counts pulses from an external source
- Figure 12-3 generates a periodic interrupt from the internal clock

```
//  This example uses TMR1 to count pulses (actually counts rising edges of the pulses)
//    from an external source (Pin TIO3).
//
void Pulse_Init(void)
{
  /* TMR1_CTRL: CM=0,PCS=3,SCS=0,ONCE=0,LENGTH=0,DIR=0,Co_INIT=0,OM=0 */
  setReg(TMR1_CTRL,0x0600);           /* Set up mode */
  /* TMR1_SCTRL: TCF=0,TCFIE=0,TOF=0,TOFIE=0,IEF=0,IEFIE=0,IPS=0,INPUT=0,
                 Capture_Mode=0,MSTR=0,EEOF=0,VAL=0,FORCE=0,OPS=0,OEN=0 */
  setReg(TMR1_SCTRL,0x00);
  setReg(TMR1_CNTR,0x00);             /* Reset counter register */
  setReg(TMR1_LOAD,0x00);             /* Reset load register */
  setRegBitGroup(TMR1_CTRL,CM,0x01);  /* Run counter */
}
```

**Figure 12-2. Count Pulses from External Source**

```
// This example generates an interrupt every 100ms (10Hz),
//   assuming the Peripheral Clock is operating at 24 MHz.
//
// It does this by using the Peripheral Clock divided by 128 as the counter clock source.
// The counter then counts to 18750 (dec) where it matches the COMP1 value.
// At that time an interrupt is generated, the counter is reloaded and
// the next COMP1 value is loaded from CMPLD1.
//
void TimerInt_Init(void)
{
  /* TMR0_CTRL: CM=0,PCS=0,SCS=0,ONCE=0,LENGTH=1,DIR=0,Co_INIT=0,OM=0 */
  setReg(TMR0_CTRL,0x20);              /* Stop all functions of the timer */
  /* TMR0_SCTRL: TCF=0,TCFIE=0,TOF=0,TOFIE=0,IEF=0,IEFIE=0,IPS=0,INPUT=0,
                 Capture_Mode=0,MSTR=0,EEOF=0,VAL=0,FORCE=0,OPS=0,OEN=0 */
  setReg(TMR0_SCTRL,0x00);
  setReg(TMR0_LOAD,0x00);              /* Reset load register */
  setReg(TMR0_COMP1,18750;             /* Set up compare 1 register */
  setReg(TMR0_CMPLD1,18750);           /* Also set the compare preload register */
  /* TMR0_CSCTRL: ??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,TCF2EN=0,TCF1EN=1,
                  TCF2=0,TCF1=0,CL2=0,CL1=1 */
  setReg(TMR0_CSCTRL,0x41);            /* Enable compare 1 interrupt and */
                                        /*   compare 1 preload            */
  setRegBitGroup(TMR0_CTRL,PCS,0xF);   /* Primary Count Source to Peri_Clk/ 128 */
  setReg(TMR0_CNTR,0x00);              /* Reset counter register */
  setRegBitGroup(TMR0_CTRL,CM,0x01);   /* Run counter */
}
```

**Figure 12-3. Generate Periodic Interrupt By Counting Internal Clocks**

## 12.7.3 Both-Edge Count Mode

If the COUNT_MODE[2:0] field is set to ‘010’, the counter will count both edges of the selected external clock source. This mode is useful for counting the changes in the external environment such as a simple encoder wheel. Figure 12-4 code example counts both pulse edges from an external source.

```
//  This example uses TMR1 to count pulse edges(actually counts both edges of the pulse)
//    from an external source (Pin TIO3).
//
void Pulse_Init(void)
{
  /* TMR1_CTRL: CM=0,PCS=3,SCS=0,ONCE=0,LENGTH=0,DIR=0,Co_INIT=0,OM=0 */
  setReg(TMR1_CTRL,0x0600);            /* Set up mode */
  /* TMR1_SCTRL: TCF=0,TCFIE=0,TOF=0,TOFIE=0,IEF=0,IEFIE=0,IPS=0,INPUT=0,
                 Capture_Mode=0,MSTR=0,EEOF=0,VAL=0,FORCE=0,OPS=0,OEN=0 */
  setReg(TMR1_SCTRL,0x00);
  setReg(TMR1_CNTR,0x00);              /* Reset counter register */
  setReg(TMR1_LOAD,0x00);              /* Reset load register */
  setRegBitGroup(TMR1_CTRL,CM,0x02);   /* Run counter */
}
```

**Figure 12-4. Count Both Edges of External Source Signal**

## 12.7.4 Gated-Count Mode

If the COUNT_MODE[2:0] field is set to ‘011’, the counter will count while the selected secondary input signal is high. This mode is used to time the duration of external events. If the selected input is inverted

by setting the IPS bit (Input Polarity Select), then the counter will count while the selected secondary input is low.

The Figure 12-5 code example is a gated-count mode where it captures the duration of an external active high pulse on Pin TIO1.

```
//  This example uses TMR1 to determine the duration of an external pulse.
//
//  The Peripheral Clock (24MHz) is used as the counter source.  If the duration of the
//    external pulse is longer than ~0.0027 seconds, a prescaled clock can be used
//    with a sacrifice of granularity.
//
void Pulse1_Init(void)
{
  /* TMR1_CTRL: CM=0,PCS=8,SCS=1,ONCE=0,LENGTH=0,DIR=0,Co_INIT=0,OM=0 */
  setReg(TMR1_CTRL,0x1080);             /* Set up mode */

  /* TMR1_SCTRL: TCF=0,TCFIE=0,TOF=0,TOFIE=0,IEF=0,IEFIE=0,IPS=0,INPUT=0,
                 Capture_Mode=0,MSTR=0,EEOF=0,VAL=0,FORCE=0,OPS=0,OEN=0 */
  setReg(TMR1_SCTRL,0x00);
  setReg(TMR1_CNTR,0x00);             /* Reset counter register */
  setReg(TMR1_LOAD,0x00);             /* Reset load register */
  setRegBitGroup(TMR1_CTRL,CM,0x03);  /* Run counter */
}
```

**Figure 12-5. Capture Duration of External Pulse**

## 12.7.5 Quadrature-Count Mode

If the COUNT_MODE[2:0] field is set to ‘100’, the counter will decode the primary and secondary external inputs as quadrature encoded signals. Quadrature signals are usually generated by rotary or linear sensors used to monitor movement of motor shafts or mechanical equipment. The quadrature signals are square waves that are 90 degrees out of phase. The decoding of quadrature signal provides both count and direction information.

Figure 12-6 shows a timing diagram illustrating the basic operation of a quadrature incremental position encoder.

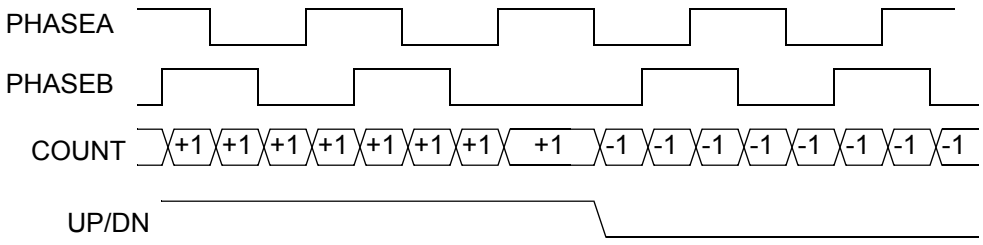



**Figure 12-6. Quadrature Incremental Position Encoder**

The Figure 12-7 code example is a quadrature-count mode where Pin TIO0 is Phase A and Pin TIO1 is Phase B.

```
//  This example uses TMR0 for counting states of a quadrature position encoder.
//
//  Timer input TIO0 is used as the primary count source (PHASEA).
//  Timer input TIO1 is used as the secondary count source (PHASEB).
//
void Pulse_Init(void)
{
  /* TMR0_CTRL: CM=0,PCS=0,SCS=1,ONCE=0,LENGTH=0,DIR=0,Co_INIT=0,OM=0 */
  setReg(TMRC0_CTRL,0x80);              /* Set up mode */

  /* TMR0_SCTRL: TCF=0,TCFIE=0,TOF=0,TOFIE=0,IEF=0,IEFIE=0,IPS=0,INPUT=0,
                 Capture_Mode=0,MSTR=0,EEOF=0,VAL=0,FORCE=0,OPS=0,OEN=0 */
  setReg(TMR0_SCTRL,0x00);

  setReg(TMR0_CNTR,0x00);              /* Reset counter register */
  setReg(TMR0_LOAD,0x00);              /* Reset load register */
  setReg(TMR0_COMP1,0xFFFF);            /* Set up compare 1 register */
  setReg(TMR0_COMP2,0x00);              /* Set up compare 2 register */

  /* TMR0_CSCTRL: ??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,
                  TCF2EN=0,TCF1EN=0,TCF2=0,TCF1=0,CL2=0,CL1=0 */
  setReg(TMR0_CSCTRL,0x00);

  setRegBitGroup(TMR0_CTRL,CM,0x04);  /* Run counter */
}
```

**Figure 12-7. Quadrature Count Mode Example**

## 12.7.6 SIgned-Count Mode

If the COUNT_MODE[2:0] field is set to ‘101’, the counter counts the primary clock source while the selected secondary source provides the selected count direction (up/down).

The Figure 12-8 code example is a signed-count mode where Pin TIO2 is primary (count input) and Pin TIO1 is secondary (count direction).

```
//
//  This example uses TMR0 for signed mode counting.
//
//  Timer input TIO2 is used as the primary count source.
//  Timer input TIO1 is used to determine the count direction.
//
void Pulse_Init(void)
{
  /* TMR0_CTRL: CM=0,PCS=2,SCS=1,ONCE=0,LENGTH=0,DIR=0,Co_INIT=0,OM=0 */
  setReg(TMR0_CTRL,0x0480);             /* Set up mode */

  /* TMR0_SCTRL: TCF=0,TCFIE=0,TOF=0,TOFIE=1,IEF=0,IEFIE=0,IPS=0,INPUT=0,
                 Capture_Mode=0,MSTR=0,EEOF=0,VAL=0,FORCE=0,OPS=0,OEN=0 */
  setReg(TMR0_SCTRL,0x1000);

  setReg(TMR0_CNTR,0x00);              /* Reset counter register */
  setReg(TMR0_LOAD,0x00);              /* Reset load register */

  setRegBitGroup(TMR0_CTRL,CM,0x05);  /* Run counter */
}
```

**Figure 12-8. Signed Count Mode Example**

### 12.7.7 Triggered-Count Mode

If the COUNT_MODE[2:0] field is set to ‘110’, the counter will begin counting the primary clock source after a positive transition (negative edge if IPS = 1) of the secondary source occurs.

- The counting will continue until a compare event occurs (Comp1 sets terminal count) or another positive input transition is detected.
- If a second source input transition occurs before a terminal count was reached, counting will stop and the TCF bit (timer compare flag) will be set.
- Subsequent secondary input transitions will continue to restart and stop the counting until a compare event occurs.

**NOTE**

The TCF bit is set upon the second positive edge.

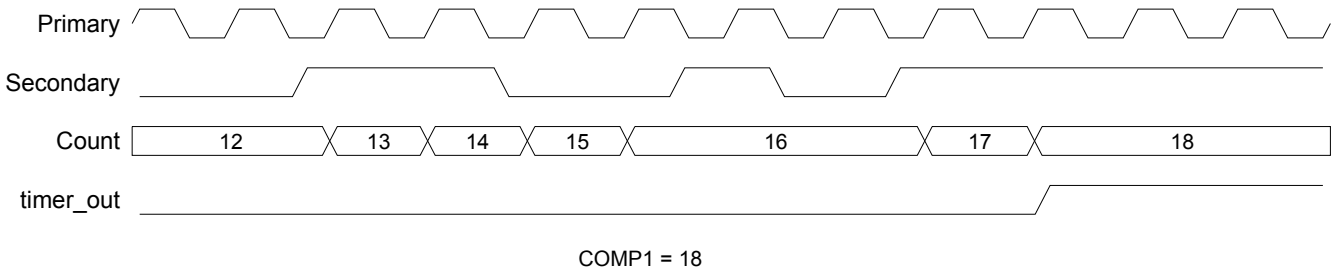



**Figure 12-9. Triggered Count Mode (Length=0)**

The Figure 12-10 code example is a triggered-count mode where Pin TIO3 is primary (count input) and Pin TIO2 is secondary (count trigger).

```
//  This example uses TMR1 for triggered mode counting.
//
//  Timer input TIO3 is used as the primary count source.
//  Timer input TIO2 is used for the trigger input.
//
void Pulse_Init(void)
{
  /* TMR1_CTRL: CM=0,PCS=3,SCS=2,ONCE=0,LENGTH=0,DIR=0,Co_INIT=0,OM=0 */
  setReg(TMR1_CTRL,0x0700);           /* Set up mode */

  /* TMR1_SCTRL: TCF=0,TCFIE=0,TOF=0,TOFIE=1,IEF=0,IEFIE=0,IPS=0,INPUT=0,
                 Capture_Mode=0,MSTR=0,EEOF=0,VAL=0,FORCE=0,OPS=0,OEN=0 */
  setReg(TMR1_SCTRL,0x1000);

  setReg(TMR1_CNTR,0x00);             /* Reset counter register */
  setReg(TMR1_LOAD,0x00);             /* Reset load register */
  setReg(TMR1_COMP1,0x0012);           /* Set up compare 1 register */

  /* TMR1_CSCTRL: ??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,
                  TCF2EN=0,TCF1EN=0,TCF2=0,TCF1=0,CL2=0,CL1=0 */
  setReg(TMR1_CSCTRL,0x00);

  setRegBitGroup(TMR1_CTRL,CM,0x06);  /* Run counter */
}
```

**Figure 12-10. Triggered Count Mode Example**

## 12.7.8 One-Shot Mode

If the COUNT_MODE[2:0] field is set to ‘110’, and the counter is set to re-initialize at a compare event (Count Length =1),and the OFLAG Output Mode is set to ‘101’ (Cleared on init, set on Compare), the counter works in a “One-Shot Mode”.

- An external event causes the counter to count.
- When terminal count is reached, the output is asserted.
  - — The Comp1 value determines the terminal count length
  - — Counter is re-initialized ready for another external event

This “delayed” output can be used to provide timing delays.

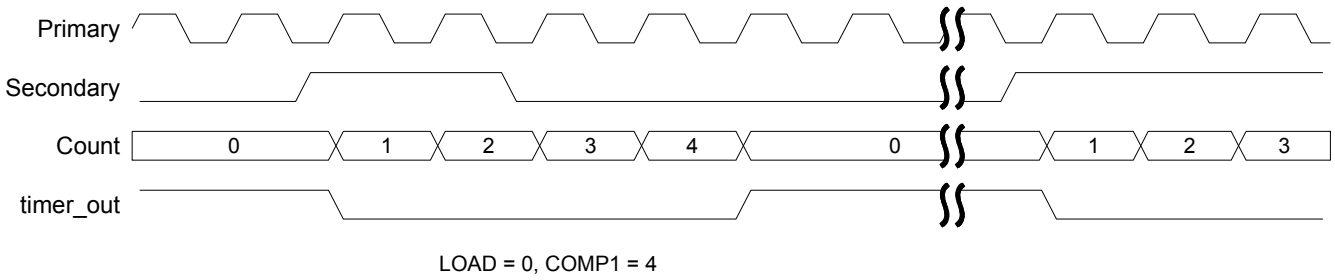



**Figure 12-11. One-Shot Mode (Length=0)**

The Figure 12-12 code example is a one-shot mode where Pin TIO3 is primary (count input) and Pin TIO2 is secondary (count trigger).

```
//  This example uses TMR1 for one-shot mode counting.
//
//  Timer input TIO3 is used as the primary count source.
//  Timer input TIO2 is used for the trigger input.
//
void Pulse_Init(void)
{
  /* TMR1_CTRL: CM=0,PCS=3,SCS=2,ONCE=0,LENGTH=1,DIR=0,Co_INIT=0,OM=5 */
  setReg(TMR1_CTRL,0x0725);           /* Set up mode */

  /* TMR1_SCTRL: TCF=0,TCFIE=0,TOF=0,TOFIE=1,IEF=0,IEFIE=0,IPS=0,INPUT=0,
                 Capture_Mode=0,MSTR=0,EEOF=0,VAL=0,FORCE=0,OPS=0,OEN=0 */
  setReg(TMR1_SCTRL,0x1000);

  setReg(TMR1_CNTR,0x00);             /* Reset counter register */
  setReg(TMR1_LOAD,0x00);             /* Reset load register */
  setReg(TMR1_COMP1,0x0004);           /* Set up compare 1 register */

  /* TMR1_CSCTRL: ??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,
                  TCF2EN=0,TCF1EN=0,TCF2=0,TCF1=0,CL2=0,CL1=0 */
  setReg(TMR1_CSCTRL,0x00);

  setRegBitGroup(TMR1_CTRL,CM,0x06);  /* Run counter */
}
```

**Figure 12-12. One-Shot Mode Example**

## 12.7.9 Cascade-Count Mode

In counter modes “001” to “110”, two, three or four counters can be "cascaded" to create very long time bases. In these modes the total counter operates as a multi-stage ripple counter. Each 16-bit counter is still synchronous but the output of one counter ripples into the next counter. The net effect is that it is possible to read the multiple counter values and get a invalid result. When cascading counters to generate long time bases and it is required to read all counters and get a correct value, cascade mode (COUNT_MODE[2:0] = 111) should be used. Up to four counters may be cascaded to create a 64-bit wide synchronous counter.

In cascade mode, a counter’s input is connected to the output of previous counter. The counter will count up and down as compare events occur in the previous counter. This “Cascade” mode enables multiple counters to be cascaded to yield longer counter lengths just as in the previous mode (001 to 110), but now the counter values can be read at any time via the HOLD registers.

Whenever any counter is read within the TMR module, all of the counters’ values are captured in their respective Hold registers. This action supports the reading of a cascaded counter chain. First read any counter of a cascaded counter chain, then read the hold registers of the other counters in the chain. The Cascaded counter mode is synchronous.

There are 6 possible combinations of counters in this "cascade" mode.

- Cascade 0 into 1. Counter 1 is set to cascade mode.
- Cascade 1 into 2. Counter 2 is set to cascade mode.
- Cascade 2 into 3. Counter 3 is set to cascade mode.
- Cascade 0 into 1 into 2. Counters 1 and 2 are set to cascade mode.

- Cascade 1 into 2 into 3. Counters 2 and 3 are set to cascade mode.
- Cascade 0 into 1 into 2 into 3. Counters 1, 2, and 3 are set to cascade mode.

In the previous modes (001 to 110) there are many more combinations. This is because in these ripple modes of cascading counters, the output of any counter can be used as the input to any counter accept itself.

The Figure 12-13 code example is a cascade-mode 2-stage counter.

```
// This example generates an interrupt every 30 seconds,
//   assuming the bus is operating at 24 MHz.
//
// To do this, Counter 2 is used to count 24,000 Peripheral Clk cycles, which means it
//   will compare and reload every 0.001 seconds.
// Counter 3 is cascaded and used to count the 0.001 second ticks and
//   generate the desired interrupt interval.
//
void TimerInt_Init(void)
{
// Set Counter 2 to count Peripheral_Clocks
  /* TMR2_CTRL: CM=0,PCS=8,SCS=0,ONCE=0,LENGTH=1,DIR=0,Co_INIT=0,OM=0 */
  setReg(TMR2_CTRL,0x1020);           /* Stop all functions of the timer */

// Set counter 3 as cascaded and to count counter 2 outputs
  /* TMR3_CTRL: CM=7,PCS=6,SCS=0,ONCE=0,LENGTH=1,DIR=0,Co_INIT=0,OM=0 */
  setReg(TMR3_CTRL,0xEC20);           /* Set up cascade counter mode */

 /* TMR3_SCTRL: TCF=0,TCFIE=0,TOF=0,TOFIE=0,IEF=0,IEFIE=0,IPS=0,INPUT=0,
                Capture_Mode=0,MSTR=0,EEOF=0,VAL=0,FORCE=0,OPS=0,OEN=0 */
  setReg(TMR3_SCTRL,0x00);
  /* TMR2_SCTRL: TCF=0,TCFIE=0,TOF=0,TOFIE=0,IEF=0,IEFIE=0,IPS=0,INPUT=0,
                Capture_Mode=0,MSTR=0,EEOF=0,VAL=0,FORCE=0,OPS=0,OEN=0 */
  setReg(TMR2_SCTRL,0x00);

  setReg(TMR3_CNTR,0x00);             /* Reset counter register */
  setReg(TMR2_CNTR,0x00);
  setReg(TMR3_LOAD,0x00);             /* Reset load register */
  setReg(TMR2_LOAD,0x00);
  setReg(TMR3_COMP1,  30000);          /* milliseconds in 30 seconds */
  setReg(TMR3_CMPLD1,30000);
  setReg(TMR2_COMP1,  24000);          // Set to cycle every millisecond
  setReg(TMR2_CMPLD1,24000);

  /* TMR3_CSCTRL: ??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,
                  TCF2EN=0,TCF1EN=1,TCF2=0,TCF1=0,CL2=0,CL1=1 */
  setReg(TMR3_CSCTRL,0x41);           /* Enable compare 1 interrupt and */
                                       /*   compare 1 preload            */
  /* TMR2_CSCTRL: ??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,
                  TCF2EN=0,TCF1EN=0,TCF2=0,TCF1=0,CL2=0,CL1=1 */
  setReg(TMR2_CSCTRL,0x01);           /* Enable Compare 1 preload  */

  setRegBitGroup(TMR2_CTRL,CM,0x01);  /* Run counter */
}
```

**Figure 12-13. Generate Periodic Interrupt Cascading Two Counters**

## 12.7.10 Pulse-Output Mode

If the counter is setup for COUNT mode (COUNT_MODE[2:0] = 001), and the OFLAG OUTPUT_MODE[2:0] is set to ‘111’ (Gated Clock Output), and the Count Once bit is set, then the counter will output a pulse stream of pulses that has the same frequency of the selected clock source, and the number of output pulses is equal to the compare value minus the init value. This mode is useful for driving step motor systems.

**NOTE**

This mode does not work if the Primary Count Source is set to “1000’ (Peripheral Clk/1).

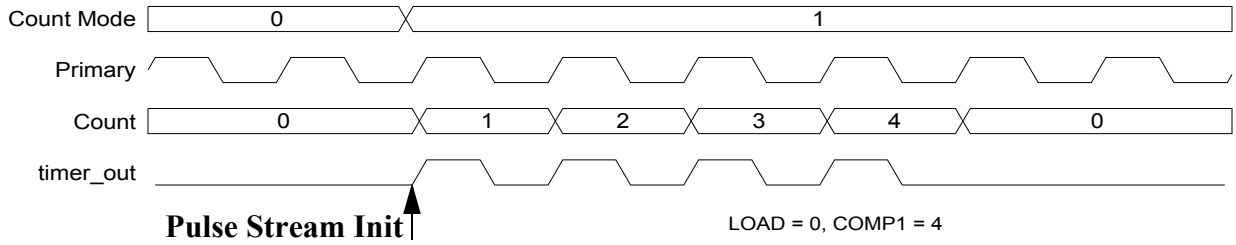



**Figure 12-14. Pulse Output Mode**

The Figure 12-15 code example is pulse-output mode example that outputs 4 pulses similar to Figure 12-14.

```
// This example generates four 10ms pulses, from TIO1 output.
// Assuming the peripheral clock is operating at 24 MHz.
//
// To do this, Timer 3 is used to generate a clock with a period of 10ms.
//
// Timer 1 is used to gate these clocks and count the number of pulses that have
//   been generated.
//
void PulseStream_Init(void)
{
// Select Peripheral_Clk/16 as the clock source for Timer 3
  /* TMR3_CTRL: CM=0,PCS=0x0C,SCS=0,ONCE=0,LENGTH=1,DIR=0,Co_INIT=0,OM=3 */
   setReg(TMR3_CTRL,0x1823);             /* Set up mode */
  /* TMR3_SCTRL: TCF=0,TCFIE=0,TOF=0,TOFIE=0,IEF=0,IEFIE=0,IPS=0,INPUT=0,
                 Capture_Mode=0,MSTR=0,EEOF=0,VAL=0,FORCE=0,OPS=0,OEN=0 */
   setReg(TMR3_SCTRL,0x00);
   setReg(TMR3_LOAD,0x00);             /* Reset load register */
   setReg(TMR3_COMP1,15000);            /* (16 * 15000 ) / 24e6 = 0.01 sec */
 /* TMR3_CSCTRL: ??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,
                   TCF2EN=0,TCF1EN=0,TCF2=0,TCF1=0,CL2=0,CL1=0 */
  setReg(TMR3_CSCTRL,0x00);           /* Set up comparator control register */


// Timer 3 output is the clock source for this timer.
  /* TMR1_CTRL: CM=0,PCS=7,SCS=0,ONCE=1,LENGTH=1,DIR=0,Co_INIT=0,OM=7 */
  setReg(TMR1_CTRL,0x0E67);           /* Set up mode */
  /* TMR1_SCTRL: TCF=0,TCFIE=0,TOF=0,TOFIE=0,IEF=0,IEFIE=0,IPS=0,INPUT=0,
                 Capture_Mode=0,MSTR=0,EEOF=0,VAL=0,FORCE=0,OPS=0,OEN=1 */
  setReg(TMR1_SCTRL,0x01);
  setReg(TMR1_CNTR,0x00);             /* Reset counter register */
  setReg(TMR1_LOAD,0x00);             /* Reset load register */
  setReg(TMR1_COMP1,0x04);             /* Set up compare 1 register */

// set to interrupt after the last pulse
  /* TMR1_CSCTRL: ??=0,??=0,??=0,??=0,??=0,??=0,??=0,??=0,
                   TCF2EN=0,TCF1EN=1,TCF2=0,TCF1=0,CL2=0,CL1=0 */
  setReg(TMR1_CSCTRL,0x40);           /* Set up comparator control register */

// Finally, start the counters running
  setReg(TMR3_CNTR,0);                /* Reset counter */

  setRegBitGroup(TMR3_CTRL,CM,0x01);  /* Run source clock counter */
  setRegBitGroup(TMR1_CTRL,CM,0x01);  /* Run counter */
}
```

**Figure 12-15. Pulse Outputs Using Two Counters**

## 12.7.11 Fixed-Frequency PWM Mode

If the counter is setup for COUNT mode (COUNT_MODE[2:0] = 001), count through roll-over (Count Length = 0), continuous count (Count Once = 0) and the OFLAG Output mode is ‘110’ (set on compare, cleared on counter roll-over) then the counter output yields a Pulse Width Modulated (PWM) signal with a frequency equal to the count clock frequency divided by 65,536 and a pulse width duty cycle equal to the compare value divided by 65,536. This mode of operation is often used to drive PWM amplifiers used to power motors and inverters.

The Figure 12-16 code example is a fixed-frequency PWM example

```
//  This example uses TMR0 for Fixed-Frequency PWM mode timing.
//
//  The timer will count Peripheral Clocks continuously until it rolls over.
//  This results in a PWM period of 65536 / 24e6 = 2730.667 usec
//
//  Initially, an output pulse width of 62.5 usec is generated ( 1500 / 24e6 )
//    giving a PWM ratio of 1500 / 65536 = 2.289%
//  This pulse width can be changed by changing the COMP1 register value (using CMPLD1).
//
void PWM1_Init(void)
{
  setReg(TMR0_CNTR,0);                 /* Reset counter */
  /* TMR0_SCTRL: TCF=0,TCFIE=0,TOF=0,TOFIE=0,IEF=0,IEFIE=0,IPS=0,INPUT=0,
                 Capture_Mode=0,MSTR=0,EEOF=0,VAL=0,FORCE=1,OPS=0,OEN=1 */
  setReg(TMR0_SCTRL,0x05);               /* Enable output */
  setReg(TMR0_COMP1,1500);           /* Store initial value to the duty-compare register */

  /* TMR0_CTRL: CM=1,PCS=8,SCS=0,ONCE=0,LENGTH=0,DIR=0,Co_INIT=0,OM=6 */
  setReg(TMR0_CTRL,0x3006);       /* Run counter */
}
```

**Figure 12-16. Fixed-Frequency PWM Mode Example**

## 12.7.12 Variable-Frequency PWM Mode

If the counter is setup for COUNT mode (COUNT_MODE[2:0] = 001), count till compare (Count Length = 1), continuous count (Count Once = 0) and the OFLAG Output mode is ‘100’ (toggle OFLAG and alternate compare registers) then the counter output yields a PWM signal whose frequency and pulse width is determined by the values programmed into the COMP1 and COMP2 registers, and the input clock frequency. This method of PWM generation has the advantage of allowing almost any desired PWM frequency and/or constant on or off periods. This mode of operation is often used to drive PWM amplifiers used to power motors and inverters. The CMPLD1 and CMPLD2 registers are especially useful for this mode, as they allow the programmer time to calculate values for the next PWM cycle while the PWM current cycle is underway.

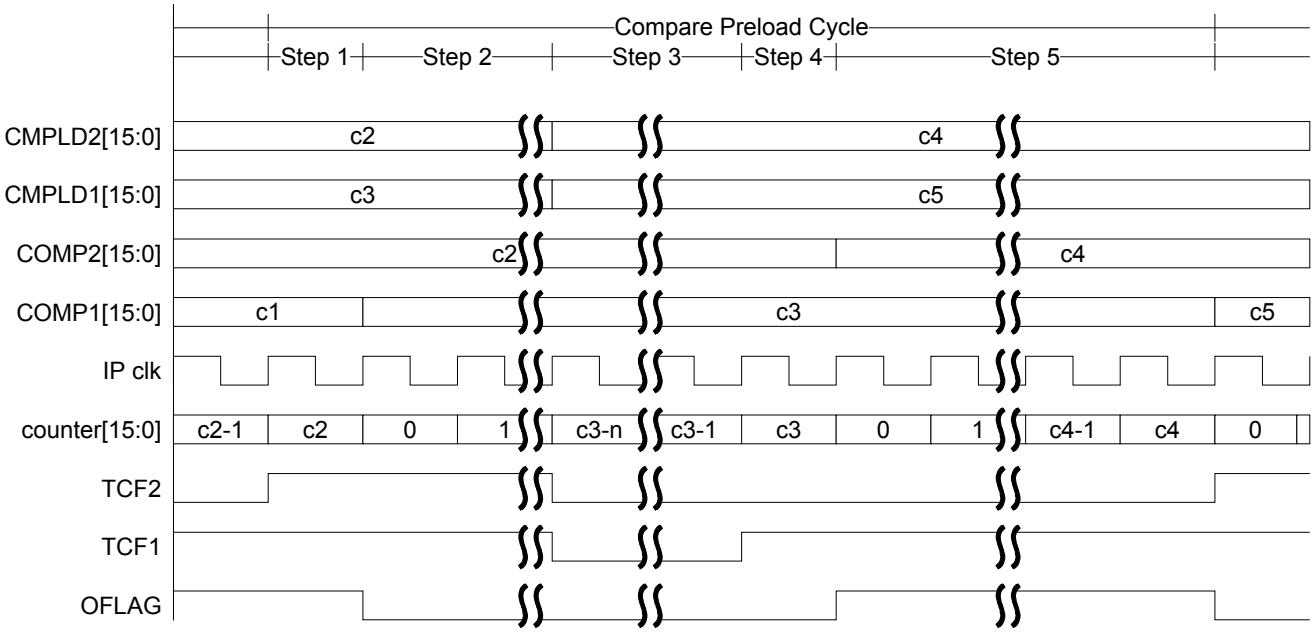
To set up the TMR to run in variable-frequency PWM mode with compare preload, use the following setup for the specific application counter.

**NOTE**

When performing the setup it is suggested that the TMR_CTRL register be updated last as the counter will start counting if the count mode is changed to any value other than 3'b000. (assuming the primary count source is already active)

- Timer Control Register (CTRL)
  - Count Mode = 3'b001 (count rising edges of primary source)
  - Primary Count Source = 4'b1000 (Peripheral Clock for best granularity for waveform timing)
  - Secondary Count Source = Any (ignored in this mode)
  - Count Once = 1'b0 (want to count repeatedly)
  - Count Length = 1'b1 (want to count until compare value is reached and re-initialize counter register)
  - Direction = Any (user's choice. The compare register values need to be chosen carefully to account for things like roll-under, etc.)
  - Co-channel Init = 1'b0 (user can set this if they need this function)
  - Output Mode = 3'b100 (Toggle OFLAG output using alternating compare registers)
- Timer Status and Control Register (SCTRL)
  - OEN = 1'b1 (Output enable to allow OFLAG output to be put on an external pin. Set this bit as needed)
  - OPS = Any (user's choice)
  - Make sure the rest of the bits are cleared for this register. Interrupts are enabled in the Comparator Status and Control Register instead of in this register.
- Comparator Status and Control Register (CSCTRL)
  - TCF2 enable = 1'b1 (allow interrupt to be issued when TCF2 is set)
  - TCF1 enable = 1'b0 (do not allow interrupt to be issued when TCF1 is set)
  - TCF1 = 1'b0 (clear timer Compare 1 interrupt source flag. This is set when counter register equals Comp 1 value and OFLAG is low)
  - TCF2 = 1'b0 (clear timer Compare 2 interrupt source flag. This is set when counter register equals Comp 2 value and OFLAG is high)
  - CL1[1:0] = 2'b10 (load compare register when TCF2 is asserted)
  - CL2[1:0] = 2'b01 (load compare register when TCF1 is asserted)
- Interrupt Service Routines -To service the TCF2 interrupts generated by the Timer, the interrupt controller must be configured to enable the interrupts for the particular timer being used. Additionally the user will need to write an interrupt service routine to do at a minimum the following:
  - Clear the TCF2 and TCF1 flags.
  - Calculate and write new values for both CMPLD1[15:0] and CMPLD2[15:0].

- Timing - Figure 12-17 contains the timing for using the compare preload feature. The compare preload cycle begins with a compare event on COMP2 causing TCF2 to be asserted. COMP1 is loaded with the value in the CMPLD1 (c3) one IP Bus clock later. In addition an interrupt is asserted by the timer and the interrupt service routine is executed during which both comparator load registers are updated with new values (c4 and c5). When TCF1 is asserted, COMP2 is loaded with the value in CMPLD2 (c4). And on the subsequent TCF2 event, COMP1 is loaded with the value in CMPLD1 (c5). The cycle starts over again as an interrupt is asserted and the interrupt service routine clears TCF1 and TCF2 and calculates new values for CMPLD1 and CMPLD2.




Step 1 -- CNTR matches COMP2 value. TCF2 is asserted and an interrupt request is generated.

Step 2 -- One clock later, OFLAG toggles, CMPLD1 is copied to COMP1, LOAD is copied to CNTR, the counter starts counting.

Step 3 -- The interrupt service routine clears TCF1 and TCF2 and the ISR loads CMPLD1 and CMPLD2 with the values for the next cycle. The counter continues counting until CNTR matches COMP1.

Step 4 -- TCF1 is asserted. One clock later, OFLAG toggles, CMPLD2 is copied to COMP2, LOAD is copied to CNTR and the counter starts counting.

Step 5 -- The counter continues counting until CNTR matches COMP2.

**Figure 12-17. Compare Load Timing**

```
//
// This example starts with an 11 ms with a 31 ms cycle.
// Assuming the chip is operating at 60 MHz, the timer use Peripheral_Clock/32 as its
//   clock source.
//
// Initial pulse period:  24e6/32 clocks/sec * 31 ms = 23250 total clocks in period
// Initial pulse width:   24e6/32 clocks/sec * 11 ms = 8250 clocks in pulse
//
//
// Once the initial values of COMP1/CMPLD1 and COMP2/CMPLD2 are set the pulse width
//   can be varied by load new values of CMPLD1 and CMPLD2 on each compare interrupt.
//
//
void PPG1_Init(void)
{
  setReg(TMR0_LOAD,0);                /* Clear load register */
  setReg(TMR0_CNTR,0);                /* Clear counter */

  /* TMR0_SCTRL: TCF=0,TCFIE=0,TOF=0,TOFIE=0,IEF=0,IEFIE=0,IPS=0,INPUT=0,
                Capture_Mode=0,MSTR=0,EEOF=0,VAL=0,FORCE=1,OPS=0,OEN=1 */
  setReg(TMR0_SCTRL,5);                 /* Set Status and Control Register */

// Set compare preload operation and enable an interrupt on compare2 events.
  /* TMR0_CSCTRL: TCF2EN=1,TCF1EN=0,TCF2=0,TCF1=0,CL21=0,CL20=1,CL11=1,CL10=0 */
  setReg(TMR0_CSCTRL,0x86);             /* Set Comparator Status and Control Register */

  setReg(TMR0_COMP1,8250);             /* Set the pulse width of the off time */
  setReg(TMR0_CMPLD1,8250);           /* Set the pulse width of the off time */
  setReg(TMR0_COMP2,23250-8250);       /* Set the pulse width of the on time  */
  setReg(TMR0_CMPLD2,23250-8250);     /* Set the pulse width of the on time  */

  /* TMR0_CTRL: CM=1,PCS=0xD,SCS=0,ONCE=0,LENGTH=1,DIR=0,Co_INIT=0,OM=4 */
  setRegBits(TMR0_CTRL,0x3A24);         /* Set variable PWM mode and run counter */
}
```

**Figure 12-18. Variable Frequency PWM Mode**

## 12.7.13 Usage of Compare Registers

The dual compare registers (COMP1 and COMP2) provide a bi-directional modulo count capability. The COMP1 register is used when the counter is *counting up*, and the COMP2 register is used when the counter is *counting down*. Alternating the compare mode is the only exception.

- The COMP1 register should be set to the desired maximum count value or 0xFFFF to indicate the maximum unsigned value prior to roll-over, and the COMP2 register should be set to the minimum count value or 0x0000 to indicate the minimum unsigned value prior to roll-under.
- If the output mode is set to 100, the OFLAG will toggle while using alternating compare registers. In the variable frequency PWM mode, the COMP2 value defines the desired pulse width of the on time, and the COMP1 register defines the off time. The variable frequency PWM mode is defined for positive counting only.
- Use caution when changing COMP1 and COMP2 while the counter is active. If the counter has already passed the new value, it will count to 0xFFFF or 0x0000, roll over, then begin counting toward the new value. The check is: Count = CMPx, *not* Count > COMP1 or Count < COMP2. Use of the CMPLD1 and CMPLD2 registers to compare values helps minimize this problem.

## 12.7.14 Usage of Compare Load Registers

The CMPLD1, CMPLD2 and CSCTRL registers offer a high degree of flexibility for loading compare registers with user-defined values on different compare events. To ensure correct functionality while using these registers, use the following method described in this section.

The purpose of the compare load feature is to allow quicker updating of the compare registers. In the past, a compare register could be updated using interrupts. However, because of the latency between an interrupt event occurring and the service of that interrupt, there was the possibility that the counter may have already counted past the new compare value by the time the compare register is updated by the interrupt service routine. The counter would then continue counting until it rolled over and reached the new compare value.

To address this, the compare registers are now updated in hardware in the same way the counter register is re-initialized to the value stored in the load register. The compare load feature allows the user to calculate new compare values and store them in to the comparator load registers. When a compare event occurs, the new compare values in the comparator load registers are written to the compare registers eliminating the use of software to do this.

The compare load feature is intended to be used in variable frequency PWM mode. The COMP1 register determines the pulse width for the logic low part of OFLAG and COMP2 determines the pulse width for the logic high part of OFLAG. The period of the waveform is determined by the COMP1 and COMP2 values and the frequency of the primary clock source. See Figure 12-19.

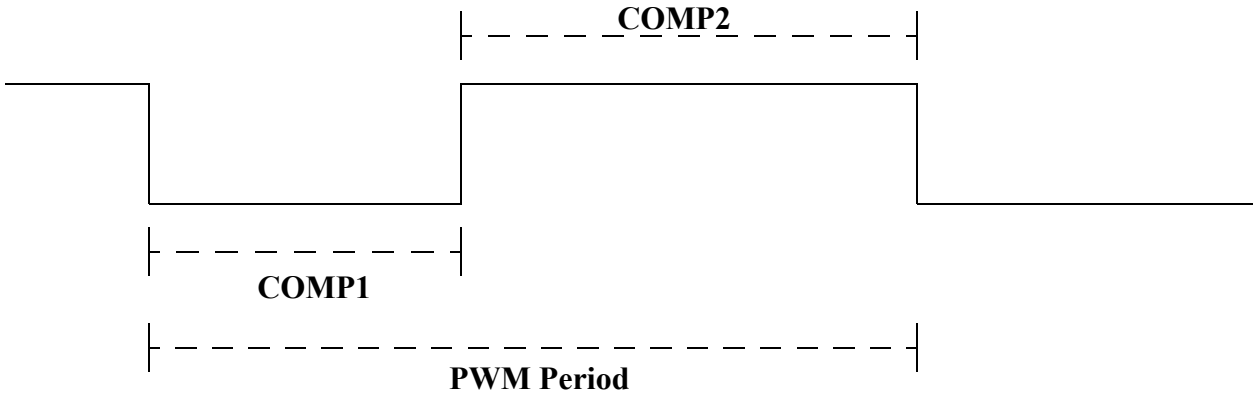



**Figure 12-19. Variable PWM Waveform**

To update the duty cycle or period of the waveform shown in Figure 12-19, the COMP1 and COMP2 values need updated using the compare load feature.

## 12.7.15 Usage of Capture Register

The Capture Register stores a copy of the counter's value when an input edge (positive, negative, or both) is detected. Once a capture event occurs, no further updating of the Capture register will occur until the IEF (Input Edge Flag) is cleared by writing a zero to the IEF.

# Chapter 13
# Universal Asynchronous Receiver/Transmitter Module (UART)

## 13.1 Overview

This chapter describes the Universal Asynchronous Receiver Transmitter (UART) module. The MC1322x has two independent UART modules designated UART1 and UART2. These can be used to connect to traditional RS232 serial ports or to communicate with other embedded controllers.

Each UART has an independent fractional divider, baud rate generator that is clocked by the peripheral bus clock (generated from the reference oscillator divided by the CRM prescaler; typically 24 MHz max) which enables a broad range of baud rates up to 1,843.2 kbaud. Transmit and receive use a common baud rate for each module.

Each UART has many advanced features useful for reliable, high-throughput serial communication including hardware flow control, independent 32-byte receive and transmit FIFOs with level indicators, high baud rate, and error detection.

## 13.2 Features

The UART module features include:

- 8-Bit only data
- Programmable one or two stop bits
- Programmable parity (even, odd, and none)
- Full duplex four-wire serial interface (RXD, TXD, RTS, and CTS)
- Programmable hardware flow control for RTS and CTS signals with programmable sense (high true / low true)
- Independent 32-byte receive FIFO and 32-byte transmit FIFO
- Status flags for flow control and FIFO states
- Receiver detects framing errors, start bit error, break characters, parity errors, and overrun errors.
- Voting logic for improved noise immunity (16X / 8X oversampling)
- Maskable interrupt request
- Time-out interrupt timer, which times out after eight non-present characters
- Receiver and transmitter enable/disable
- Baud rate generator to provide any multiple-of-2 baud rate between 1.2 kbaud and 1,843.2 kbaud
- Software reset

## 13.3 Block Diagram

The UART module is shown in Figure 13-1.

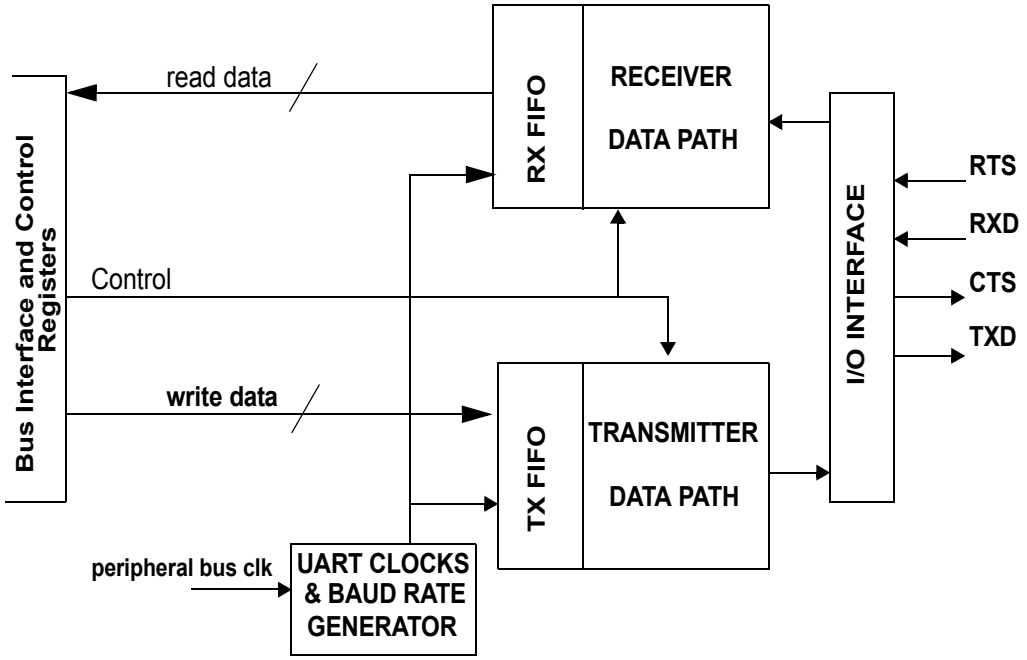



**Figure 13-1. Top Level UART Block Diagram**

## 13.4 Functional Description

The UART consists of control and status registers, a baud rate generator, TX logic with a 32-byte FIFO, and RX logic with a separate 32-byte FIFO. The transmitter and receiver operate independently, although they use the same baud rate generator. The following sections describe the functionality of the UART.

### 13.4.1 External Signal Descriptions

Table 13-1 lists UART signal names and their descriptions.

**Table 13-1. UART Signals**

| Signal Names | Direction | Active | Description |
|---|---|---|---|
| UARTx_RTS | Input | Programmable | Request to Send flow control input |
| UARTx_CTS | Output | Programmable | Clear to Send flow control output |
| UARTx_RX | Input | Low | Receive serial data input |
| UARTx_TX | Output | Low | Transmit serial data output |

## 13.4.2 Baud Rate Generation (Fractional Divider)

The high-speed UART is clocked with the peripheral bus clock (typically 24 MHz). This high frequency clock, plus an internal fractional divider, enables a wide range of frequencies calculated using the following equations:

***Eqn. 13-1***

$$baudrateX16 = \text{pb}clk \times \left[\frac{INC + 1}{MOD}\right]$$

$$\text{baudrate} = \frac{baudrateX16}{16}$$

or,

***Eqn. 13-2***

$$baudrateX8 = \text{pb}clk \times \left[\frac{INC + 1}{MOD}\right]$$

$$\text{baudrate} = \frac{baudrateX8}{8}$$

The internal baud rate generator is shown in Figure 13-2.

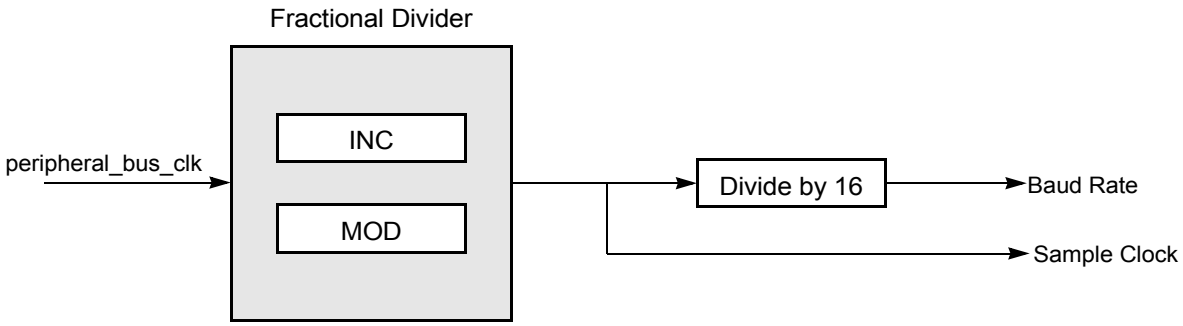



**Figure 13-2. UART Baud Rate Generator (16x oversampling)**

Table 13-2 and Table 13-3 show sample values for INC and MOD that yield standard baud rates for 8x and 16x oversampling at 13 MHz (minimum reference frequency), 24 MHz (standard reference frequency), and 26 MHz (maximum reference frequency). The INC and MOD values are programmed in the UART Baud Rate Divider Register (UBR).

**Table 13-2. Standard Baud Rates for 8x Oversampling**

| Baud Rate | Peripheral_Bus_Clk = 13Mhz | | | Peripheral_Bus_Blk = 24Mhz | | | Peripheral_Bus_Clk = 26Mhz | | |
|---|---|---|---|---|---|---|---|---|---|
| | MOD | INC | FREQ | MOD | INC | FREQ | MOD | INC | FREQ |
| 1,200 | 9,999 | 6 | 1,137.6 | 9,999 | 3 | 1,200.1 | 9,999 | 3 | 1,300.1 |
| 2,400 | 9,999 | 14 | 2,437.7 | 9,999 | 7 | 2,400.2 | 9,999 | 6 | 2,275.2 |
| 4,800 | 9,999 | 29 | 4,875.5 | 9,999 | 15 | 4,800.5 | 9,999 | 14 | 4,875.5 |
| 9,600 | 9,999 | 58 | 9,588.5 | 9,999 | 31 | 9,601.0 | 9,999 | 29 | 9,751.0 |
| 19,200 | 9,999 | 117 | 19,176.9 | 9,999 | 63 | 19,201.9 | 9,999 | 58 | 19,176.9 |
| 28,800 | 9999 | 176 | 28,765.4 | 9,999 | 95 | 28,802.9 | 9,999 | 88 | 28,927.9 |
| 38,400 | 9,999 | 235 | 38,353.8 | 9,999 | 127 | 38,403.8 | 9,999 | 117 | 38,353.8 |
| 57,600 | 9,999 | 353 | 57,530.8 | 9,999 | 191 | 57,605.8 | 9,999 | 176 | 57,530.8 |
| 115,200 | 9,999 | 708 | 115,224.0 | 9,999 | 383 | 115,211.5 | 9,999 | 353 | 115,061.5 |
| 230,400 | 9,999 | 1417 | 230,448.0 | 9,999 | 767 | 230,423.0 | 9,999 | 708 | 230,448.0 |
| 460,800 | 9,999 | 2834 | 460,733.6 | 9,999 | 1535 | 460,846.1 | 9,999 | 1417 | 460,896.1 |
| 921,600 | 9,999 | 5670 | 921,629.7 | 9,999 | 3071 | 921,692.2 | 9,999 | 2834 | 921,467.1 |
| 1,843,200 | 9,999 | 11341 | 1,843,259.3 | 9,999 | 6142 | 1,843,084.3 | 9,999 | 5670 | 1,843,259.3 |

**Table 13-3. Standard Baud Rates for 16x Oversampling**

| Baud Rate | Peripheral_Bus_Clk = 13 Mhz | | | Peripheral_Bus_Clk = 24 Mhz | | | Peripheral_Bus_Clk = 26 Mhz | | |
|---|---|---|---|---|---|---|---|---|---|
| | MOD | INC | FREQ | MOD | INC | FREQ | MOD | INC | FREQ |
| 1,200 | 9,999 | 14 | 1,218.9 | 9,999 | 7 | 1,200.1 | 9,999 | 6 | 1,300.1 |
| 2,400 | 9,999 | 29 | 2,437.7 | 9,999 | 15 | 2,400.2 | 9,999 | 14 | 2,275.2 |
| 4,800 | 9,999 | 58 | 4,794.2 | 9,999 | 31 | 4,800.5 | 9,999 | 29 | 4,875.5 |
| 9,600 | 9,999 | 117 | 9,588.5 | 9,999 | 63 | 9,601.0 | 9,999 | 58 | 9,751.0 |
| 19,200 | 9,999 | 235 | 19,176.9 | 9,999 | 95 | 19,201.9 | 9,999 | 88 | 19,176.9 |
| 28,800 | 9,999 | 353 | 28,765.4 | 9,999 | 127 | 28,802.9 | 9,999 | 117 | 28,927.9 |
| 38,400 | 9,999 | 472 | 38,435.1 | 9,999 | 191 | 38,403.8 | 9,999 | 176 | 38,353.8 |
| 57,600 | 9,999 | 708 | 57,612.0 | 9,999 | 383 | 57,605.8 | 9,999 | 353 | 57,530.8 |
| 115,200 | 9,999 | 1417 | 115,224.0 | 9,999 | 767 | 115,211.5 | 9,999 | 708 | 115,061.5 |
| 230,400 | 9,999 | 2834 | 230,366.8 | 9,999 | 1535 | 230,423.0 | 9,999 | 1417 | 230,448.0 |
| 460,800 | 9,999 | 5670 | 460,814.8 | 9,999 | 3071 | 460,846.1 | 9,999 | 2834 | 460,896.1 |
| 921,600 | 9,999 | 11341 | 921,629.7 | 9,999 | 6142 | 921,692.2 | 9,999 | 5670 | 921,467.1 |
| 1,843,200 | 9,999 | 22682 | 1,843,178.0 | 9,999 | 12286 | 1,843,234.3 | 9,999 | 11341 | 1,843,259 |

### 13.4.3 Basic Operation

The CPU accesses the UART through its data and control registers. The UART transmits and receives characters (bytes) of 8-bit length only. For transmission, data are first written to a 32-byte deep transmitter FIFO (first in, first out). This data is passed to the TX shift register and shifted serially out on the UART TXD pin. For reception, data are received serially from the UART RXD pin and stored in a 32-byte deep receiver FIFO. The received data are read from the receiver FIFO.

Figure 13-3 shows the UART data transfer format. The format supports 8-bit data with programmable odd, even, or no parity. One or two stop bits are also programmable. During transmission, a TX FIFO overrun error can be detected, and the receiver detects framing errors, start bit error, break characters, parity errors, and RX FIFO overrun errors.

The receiver and transmitter FIFOs contain a maskable interrupt that can be configured to interrupt when it reaches a certain level.

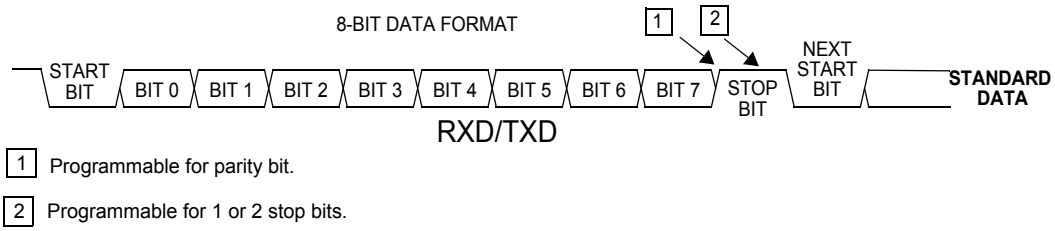



**Figure 13-3. Data Transfer Format**

The UART also supports hardware RTS and CTS flow control. See Section 13.5, “Flow Control”.

Each UART module supports an independent interrupt request to the CPU Interrupt Controller with a pre-determined priority level. The interrupt request can be from a number of sources.

The UART generated baud rate is based upon a configurable divisor and input clock and is common to both the RX and TX blocks. The fractional divider is setup by writing an INCrement and MODulo value to two registers. It is important that the UART is disabled (RxE and TxE equal 0) before writing new values to the INC/MOD registers. After writing to the registers, the receiver and/or transmitter can be enabled (RxE or TxE equal 1).

**NOTE**

When the UART is disabled (RxE and TxE equal 0) both the Rx and the Tx buffers are flushed, and the status register is updated.

### 13.4.4 FIFO Operation

Both the TX and RX FIFOs are accessed through the single 8-bit UART Data Register (UDATA). The register allows 8-bit, 16-bit and 32-bit access types; all of which use / yield a single byte transfer (the register is on a 32-bit word address boundary). The operation of the transmit and receive FIFO counters is shown in Figure 13-3.

The status of the RX FIFO is provided by the UART RxBuffer Control Register (URxCON), and the status of the TX FIFO is provided by the UART TxBuffer Control Register (UTxCON). Writing to the UDATA register loads a byte into the TX FIFO and reading the UDATA register unloads a byte from the RX FIFO

(if available). Each FIFO is 32 locations deep and the Rx_fifo_addr_diff[5:0] field of the URxCON register reports the number received bytes in the RX FIFO and the Tx_fifo_addr_diff[5:0] field in the UTxCON register reports the number of empty bytes in the TX FIFO.

The status of the FIFOs can also be used to generate an interrupt request for flow control and buffer management:

- The RxLevel[4:0] field in the URxCON register sets a watermark or threshold for the data available in the RX FIFO. If the contents of the RX FIFO meet or exceed this threshold, then an interrupt request can be enabled to empty a block of receive data.
- The TxLevel[4:0] field of the UTxCON register sets a threshold for the space (free bytes) available in the TX FIFO. If the space available meets or exceeds the threshold, then an interrupt request can be enabled to load a block of transmit data.
- Status and interrupt request generation is also supported for RX FIFO underrun, RX FIFO overrun, and TX FIFO overrun.

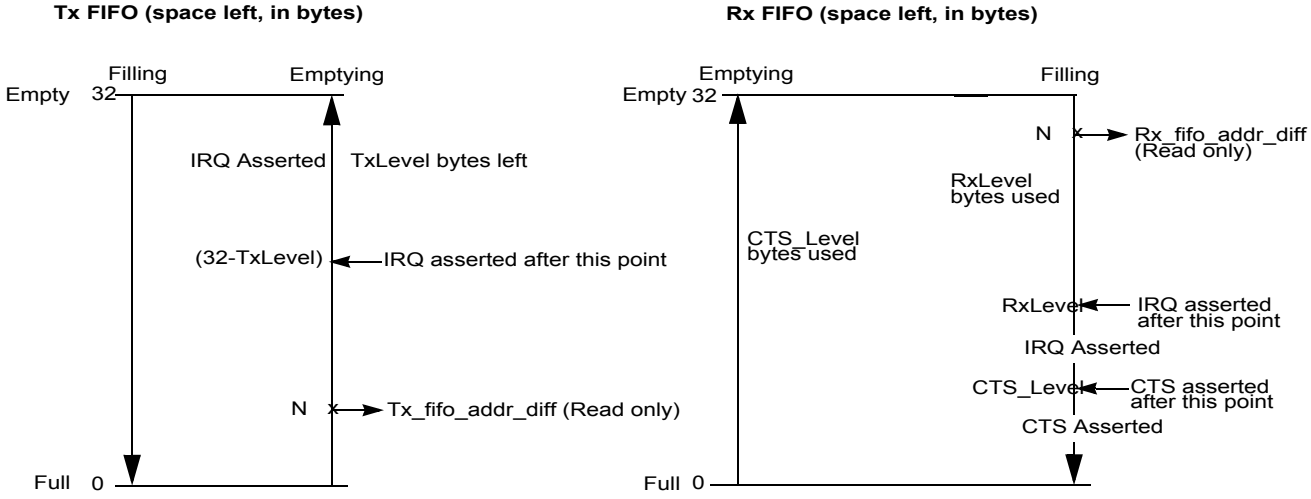
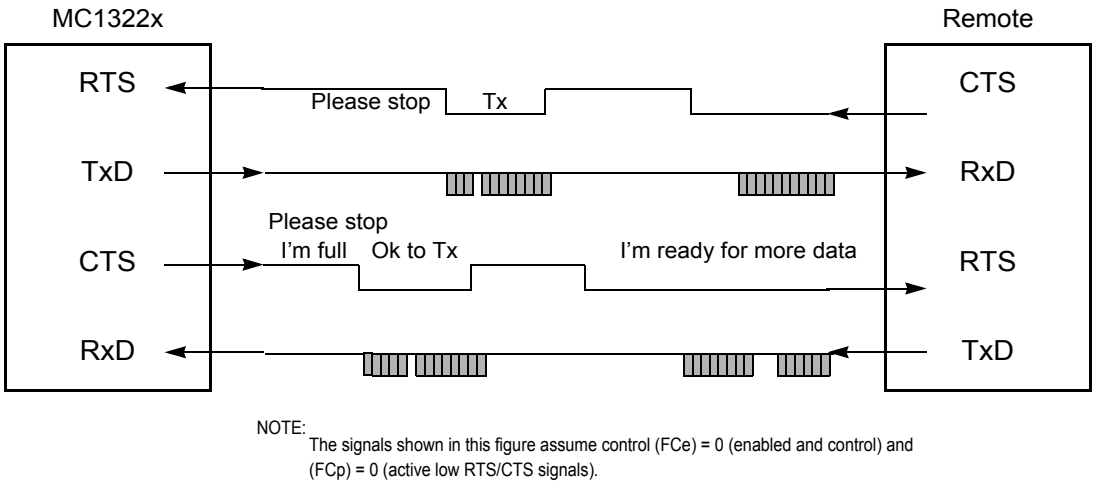



**Figure 13-4. TX and RX FIFO-Related Levels**

## 13.5 Flow Control

The UART has hardware support for:

- RTS (Ready To Send) input - The MC1322x is allowed to send because the remote unit is ready.
- CTS (Clear To Send) output - The MC1322x is ready to accept new data, so the remote unit can transmit when it has data.
- The polarity of RTS/CTS can be changed by changing the control[FCp] bit.
- When the control[FCe]=0 the flow control is disabled. This causes the UART to ignore the RTS input and asserts the CTS pin (that is, the CTS pin is therefore 0 when control[FCp]=0).




NOTE:
The signals shown in this figure assume control (FCe) = 0 (enabled and control) and (FCp) = 0 (active low RTS/CTS signals).

**Figure 13-5. Flow Control**

## 13.6 RX Timeout Counter

The UART contains a counter that provides a time-out function to indicate lack of receive activity.

- The time-out counter is always active after
  - — Receive mode is enabled (RxE bit is enabled),

  AND
  - — The first RX character has been received.
- The counter will time-out after lack of 8 characters. The wait time is set to 96 total data bits, which is derived as 8 characters * (1 start + 8 data + 1 parity + 2 stop) bits/character = 96 bits. The bit-time is determined by the programmed baud rate.
- The counter is reset to zero every time a stop bit has been processed correctly, which means, once a byte of data has been received into the RX FIFO.
- On time-out, the Rx FIFO Underrun Error Status (RUE) is set, and in turn, sets the RxRdy Status if it is not already set. An interrupt request (IRQ) will be generated if enabled.
- The time-out counter cannot be used to get a time-out IRQ before the receiver gets any data because the counter is not started until the first character is received.

## 13.7 Interrupts

Each UART module has a single interrupt request signal that is connected to the interrupt controller. The UART IRQ can be generated from two primary sources which are logically ORed such that either can generate the interrupt request if enabled. Table  lists the UART interrupt sources and their characteristics.

**Table 13-4. UART Interrupt Sources**

| Item | Status Bit | Mask Bit | Source Description | Interrupt Clear Mechanism |
|---|---|---|---|---|
| 1 | TxRdy | MTxR | Transmitter Is Causing An Interrupt - the transmitter can cause an interrupt request from two conditions<br>• TX FIFO empty space meets or exceeds TxLevel[4:0] set in UART TxBuffer Control Register (no additional status bit)<br>• TX FIFO Overrun Error - the incoming data overran the FIFO capacity. The TOE Status bit is also set | Clearing the interrupt is dependent upon the source<br>• TX FIFO empty interrupt request (MTxR) is disabled by writing to the TX FIFO<br>• TX FIFO Overrun Error TOE status is cleared by reading the status register |
| 2 | RxRdy | MRxR | Receiver Is Causing An Interrupt - the receiver can cause an interrupt request from multiple conditions<br>• Number of available bytes in RX FIFO meets or exceeds the value specified by RxLevel[4:0] set in UART RxBuffer Control Register (no additional status bit)<br>• RX FIFO Underrun Error - the RX time-out counter fired. The RUE Status bit is also set<br>• RX FIFO Overrun Error - the incoming data overran the FIFO capacity. The ROE Status bit is also set<br>• Frame/Stop Bit Error - the FE Status bit is also set<br>• Parity Error - the PE Status bit is also set<br>• Start Bit Error - the SE Status bit is also set | Clearing the interrupt is dependent upon the source<br>• RX FIFO data interrupt request (MTxR) is disabled by reading from the RX FIFO<br>• Error status are cleared by reading the status register |

### 13.7.1 Transmitter Interrupt Request Sources

As shown in Table , the transmitter can generate a request from two sources

- TX FIFO empty space meets or exceeds TxLevel[4:0] set in UART TxBuffer Control Register - this interrupt is used to ask for more data in the TX FIFO to keep the UART TX channel running at max capacity. The threshold is programmable.
- TX FIFO Overrun Error - the data written to the TX FIFO overran the FIFO capacity. The TOE Status bit is set. This should not occur is application software is written properly

### 13.7.2 Receiver Interrupt Request Sources

As shown in Table , the receiver can generate a request from multiple sources

- Number of available received bytes in RX FIFO meets or exceeds the value specified by RxLevel[4:0] set in UART RxBuffer Control Register - this interrupt is used to ask the CPU to fetch data from the RX FIFO. The threshold is programmable
- RX FIFO Underrun Error - the RX time-out counter fired. The RUE Status bit is also set.
- RX FIFO Overrun Error - the incoming RX data overran the FIFO capacity. The ROE Status bit is set.
- RX Frame/Stop Bit Error - the FE Status bit is set
- RX Parity Error - the PE Status bit is set
- RX Start Bit Error - the SE Status bit is set

## 13.8 UART Register Memory Map

Each UART module is programmed via a set of memory-mapped registers and their respective base addresses are listed in Table 13-5. The register memory map for each module related to the base address is listed in Table 13-6 and shows the 8-bit registers of the UART module.

The control and status registers should be accessed only as 32 bits. The data register may be accessed as 32, 16 or 8 bits.

**Table 13-5. Base Addresses of UART Modules**

| Base Address | UART Designation |
|---|---|
| 0x80005000 | UART1 |
| 0x8000B000 | UART2 |

**Table 13-6. UART Module Register Memory Map**

| Offset | Name | Access Type | Access Width |
|---|---|---|---|
| Base + 0x00 | UART Control Register (UCON) | R/W | 32-bit only |
| Base + 0x04 | UART Status Register (USTAT) | R | 32-bit only |
| Base + 0x08 | UART Data Register (UDATA) | R/W | 8-bit, 16-bit, or 32-bit |
| Base + 0x0C | UART RxBuffer Control Register (URxCON) | R/W | 32-bit only |
| Base + 0x10 | UART TxBuffer Control Register (UTxCON) | R/W | 32-bit only |
| Base + 0x14 | UART CTS Level Control Register (UCTS) | R/W | 32-bit only |
| Base + 0x18 | UART Baud Rate Divider Register (UBR) | R/W | 32-bit only |

# 13.9 UART Registers

The following sections provide descriptions of the UART registers.

## 13.9.1 UART Control Register (UCON)

The UART Control Register (UCON) is used to specify transmission parameters, such as flow control, stop bits, parity, etc.

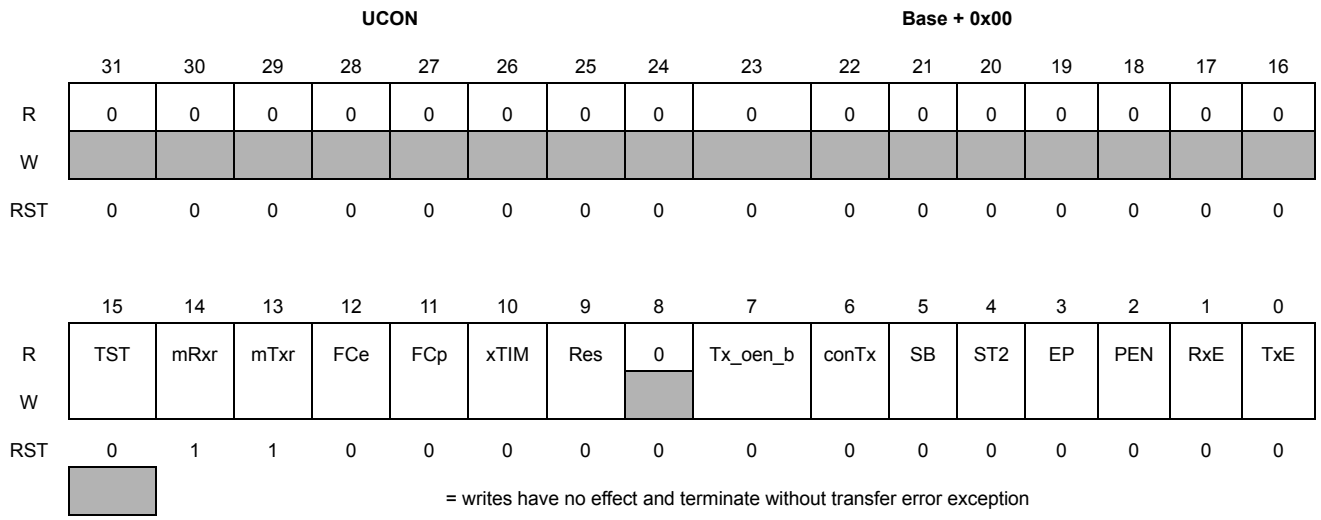



**Table 13-7. AGC Software Override Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-16 | Reserved | |
| 15 | **TST** — Test Loop-Back . When set, the transmitter is internally fed back to the receiver for a loop-back test. | 1 = Test mode<br>0 = Normal operation (default) |
| 14 | **MRxR** — Mask RxRDY Interrupt. Used to enable or disable (mask) receive interrupt requests | 1 = Masked<br>0 = Enabled (default) |
| 13 | **MTxR** — Mask TxRDY Interrupt. Used to enable or disable (mask) transmit interrupt requests | 1 = Masked<br>0 = Enabled (default) |
| 12 | **FCe** — Flow Control Enable | 1 = Enabled<br>0 = Disabled (default) |
| 11 | **FCp** — Flow Control polarity | 1 = Assert RTS to enable transmission<br>0 = Deassert RTS to enable transmission (default) |
| 10 | **xTIM** — Times Of Oversampling | 1 = 16 times oversampling<br>0 = 8 times oversampling (default) |
| 9-8 | Reserved | Read only |
| 7 | **Tx_oen_b** — TXD Output Enable. This bit is used to disable the TXD output. This typically done when in loopback mode and to not send external data. | 1 = Disable TXD output<br>0 = Enable TXD output (default) |

**Table 13-7. AGC Software Override Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 6 | **conTx** — Continuous Tx (Test Mode) | 1 = Enable<br>0 = Disable (default) |
| 5 | **SB** — Send Break | 1 = Send Break<br>0 = No Break (default) |
| 4 | **ST2** — Stop Bits | 1 = Two Stop Bits<br>0 = One Stop Bit (default) |
| 3 | **EP** — Even Parity | 1 = Odd<br>0 = Even (default) |
| 2 | **PEN** — Parity Enable | 1 = Enable<br>0 = Disable (default) |
| 1 | **RxE** — Rx Enable | 1 = Enable<br>0 = Disable (default) |
| 0 | **TxE** — Tx Enable | 1 = Enable<br>0 = Disable (default) |

## 13.9.2 UART Status Register (USTAT)

The UART Status Register (USTAT) indicates interrupt request status and any errors that have been detected during transmission, such as FIFO buffer overflow or underflow, parity error, and frame, start, or stop bit error. This register is read-only.

**NOTE**

- Bits[7:6] are cleared when the respective condition allows it.
- Status bits[5:0] are cleared when the register is read.

**USTAT** **Base + 0x04**

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| W | | | | | | | | | | | | | | | | |
| RST | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | TxRdy | RxRdy | RUE | ROE | TOE | FE | PE | SE |
| W | | | | | | | | | | | | | | | | |
| RST | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 13-8. USTAT Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-8 | Reserved | Read only |
| 7 | **TxRdy** — Transmitter Is Causing An Interrupt. | 1 = Interrupt pending<br>0 = No interrupt (default) |
| 6 | **RxRdy** — Receiver Is Causing An Interrupt | 1 = Error occurred<br>0 = No error (default) |
| 5 | **RUE** — Rx FIFO Underrun Error | 1 = Error occurred<br>0 = No error (default) |
| 4 | **ROE** — Rx FIFO Overrun Error | 1 = Error occurred<br>0 = No error (default) |
| 3 | **TOE** — Tx FIFO Overrun Error | 1 = Error occurred<br>0 = No error (default) |
| 2 | **FE** — Frame/Stop Bit Error | 1 = Error occurred<br>0 = No error (default) |
| 1 | **PE** — Parity Error | 1 = Error occurred<br>0 = No error (default) |
| 0 | **SE** — Start Bit Error | 1 = Error occurred<br>0 = No error (default) |

## 13.9.3 UART Data Register (UDATA)

The UART Data Register is used to write the UART transmit FIFO buffer with bytes that are to be transmitted, and to read the received bytes from the UART receive FIFO buffer.

**NOTE**

- All 8-bit, 16-bit, or 32-bit access types all allowed on the data fields
- All accesses yield only one valid data byte.

**UDATA** **Base + 0x08**

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| W | | | | | | | | | | | | | | | | |
| RST | | | | | | | | | | | | | | | | |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Rx_data | | | | | | | |
| W | | | | | | | | | Tx_Data | | | | | | | |
| RST | | | | | | | | | Undefined | | | | | | | |

**Table 13-9. UDATA Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-8 | Reserved | Read only |
| 7-0 | **Rx_data[7:0]** — Received Byte. This bit field contains the 8-bit data that has been received. | Read only |
| 7-0 | **Tx_data[7:0]** — Byte To Transmit. This bit field contains the 8-bit data to be transmitted. | Write only |

## 13.9.4 UART Buffer Control Registers

The UART buffers use threshold values to generate interrupts. These thresholds can be used to specify that an interrupt is generated before the transmit FIFO has become completely empty, or some time before the receive FIFO has been completely filled. These “early warnings” allow relaxed interrupt response times.

### 13.9.4.1 UART RxBuffer Control Register (URxCON)

The UART RxBuffer Control Register provides the level of data within the RX FIFO and also sets the threshold for the interrupt request that is generated when the receive buffer is considered sufficiently filled for retrieving data. Two fields are used:

- Rx_fifo_addr_diff[5:0] (read-only) - this value indicates the number of bytes currently buffered in the receive FIFO buffer. The number of bytes is between 0 (empty) and 32 (full) bytes. If this value goes negative (i.e., exceeding FIFO capacity), the write enable is deasserted and no more writes will occur.
- RxLevel[4:0] (write-only) -When the number of bytes filled in the receive FIFO meets or exceeds the value specified by RxLevel, an interrupt is generated.

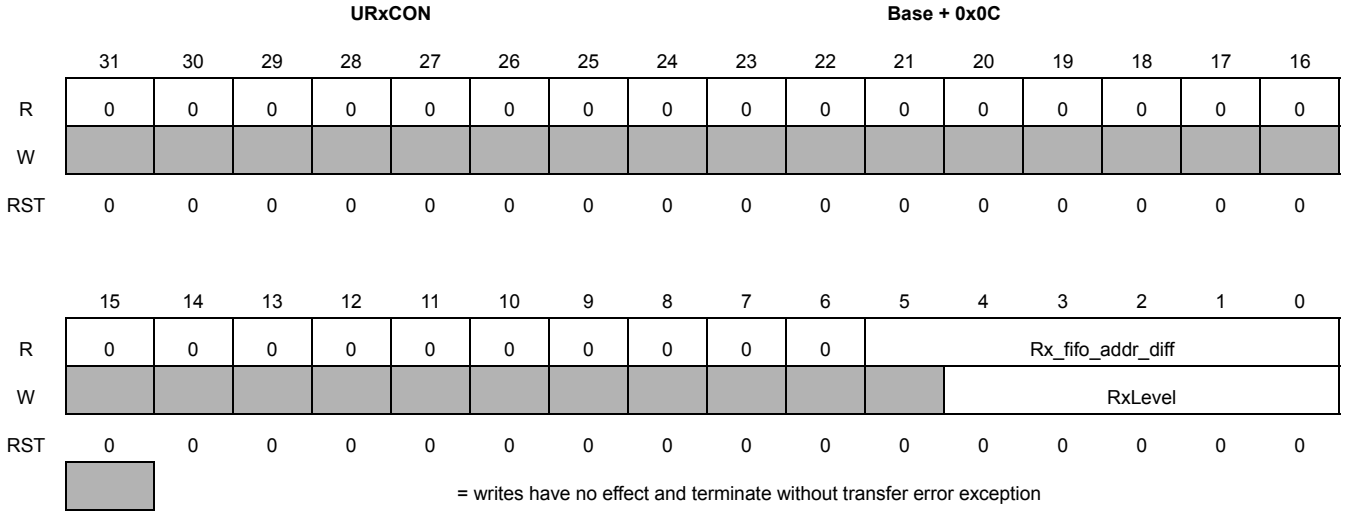



**Table 13-10. URxCON Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-6 | Reserved | |
| 5-0 | **Rx_fifo_addr_diff[5:0]** — Receive buffer full level. This value indicates the number of bytes currently buffered in the receive FIFO buffer. The number of bytes is between 0 (empty) and 32 (full) bytes. | Read only |
| 4-0 | **RxLevel[4:0]** — Receive buffer level . When the number of bytes filled in the receive FIFO meets or exceeds the value specified by RxLevel, an interrupt is generated. | Write only |

### 13.9.4.2 UART TxBuffer Control Register (UTxCON)

The UART TxBuffer Control Register provides the number of free bytes within the TX FIFO and also sets the threshold for the interrupt request that is generated when the transmit buffer is considered sufficiently empty for writing additional data. Two fields are used:

- Tx_fifo_addr_diff[5:0] (read-only) - this value indicates the number of empty bytes currently available in the transmit FIFO buffer. The number of free bytes is between 0 (full) and 32 (empty) bytes. Data can be written to the transmit buffer while this number is non-zero. If this value goes negative (i.e., exceeding FIFO capacity), the write enable is deasserted and no more writes will occur.
- TxLevel[4:0] (write-only) -When the number of bytes available in the transmit FIFO meets or exceeds the value specified by RxLevel, an interrupt is generated.

**UTxCON** **Base + 0x10**

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| W | | | | | | | | | | | | | | | | |
| RST | | | | | | | | | | | | | | | | |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Tx_fifo_addr_diff | | | | | |
| W | | | | | | | | | | | | TxLevel | | | | |
| RST | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 |

= writes have no effect and terminate without transfer error exception

**Table 13-11. UTxCON Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-6 | Reserved | |
| 5-0 | **Tx_fifo_addr_diff[5:0]** — Transmit Buffer Empty Level. This value indicates the number of empty bytes currently available in the transmit FIFO buffer. The number of free bytes is between 0 (full) and 32 (empty) bytes. | Read only |
| 4-0 | **TxLevel[4:0]** — Transmit Buffer Level. When the number of free bytes in the transmit FIFO buffer meets or exceeds the value specified by TxLevel, an interrupt is generated. | Write only |

### 13.9.4.3 UART CTS Level Control Register (UCTS)

UART CTS Level Control Register (UCTS) sets the threshold for the CTS flow control signal. If the remote UART continues to send limited amounts of data after detecting the deasserted CTS signal, the local receive buffer can handle the additional data if CTS_Level is set to a suitable value.


**Table 13-12. UCTS Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-5 | Reserved | Read only |
| 4-0 | **CTS_Level[4:0]** — CTS Buffer Level. When the number of bytes in the receiver (Rx) FIFO exceeds this value, the CTS signal is deasserted. | |

## 13.9.5 UART Baud Rate Divider Register (UBR)

The UART Baud Rate Divider Register contains the fractional divider register fields used to compute the receive and transmit baud rate.

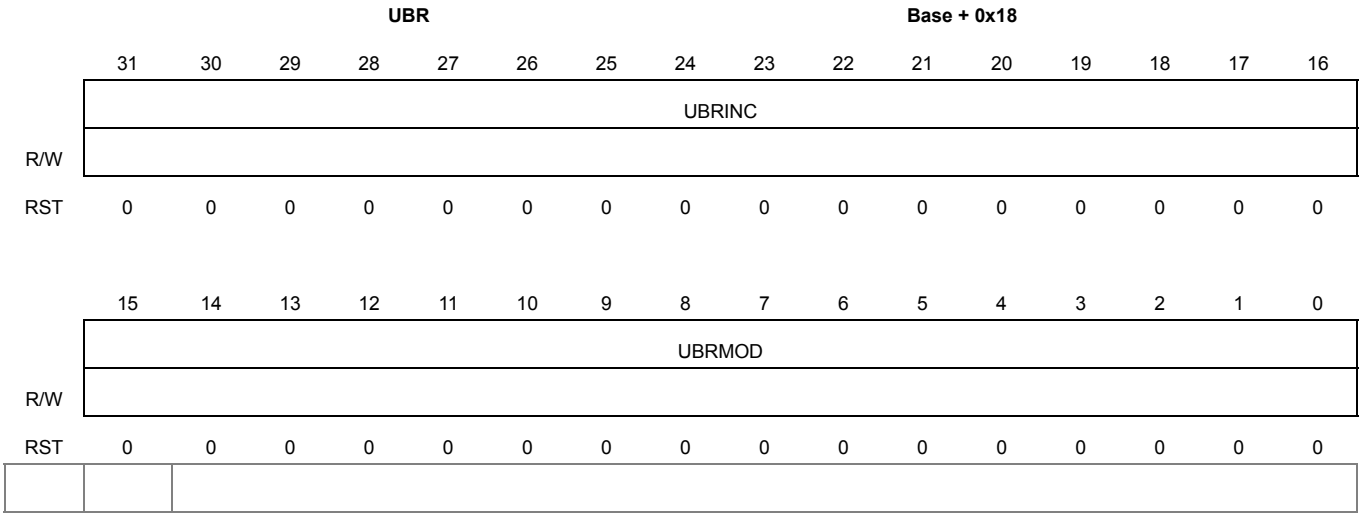



**Table 13-13. UBR Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-16 | **UBRINC[15:0]** — INC Value. The UBRINC field is used to select the increment value for the fractional division divider. See Section 13.4.2, “Baud Rate Generation (Fractional Divider)” for an explanation of the value to be written to the register. | |
| 15-0 | **UBRMOD[15:0]** — MOD Value. The UBRMOD field is used to select the modulus for the fractional division divider. See Section 13.4.2, “Baud Rate Generation (Fractional Divider)” for an explanation of the value to be written to the register. | |

# Chapter 14
# $I^2C$ Module

## 14.1 Overview

The inter-IC (IIC or $I^2C$) bus is a two-wire—serial data (SDA) and serial clock (SCL)— bidirectional serial bus that provides a simple efficient method of data exchange between the system and other devices, such as microcontrollers, EEPROMs, real-time clock devices, A/D converters, and LCDs. The two-wire bus minimizes the interconnections between devices. The synchronous, multi-master bus of the $I^2C$ allows the connection of additional devices to the bus for expansion and system development. The bus includes collision detection and arbitration that prevent data corruption if two or more masters attempt to control the bus simultaneously.

**NOTE**

- See Section 14.8.7, "I2C Clock Enable Register (I2CCKER)" for enabling the $I^2C$ module.
- Although this chapter provides reference information on the $I^2C$ module, the user/programmer is directed toward the software utility entitled the "Inter-integrated Circuit ($I^2C$) Driver", see Appendix B, "MC1322x Software Driver Utilities".

## 14.2 Features

The $I^2C$ interface includes the following features:

- Two-wire interface
- Multi-master operational
- Arbitration lost interrupt with automatic mode switching from master to slave
- Calling address identification interrupt
- START and STOP signal generation/detection
- Acknowledge bit generation/detection
- Bus busy detection
- Software-programmable bit clock frequency up to 150 kbps (module clock source is peripheral clock)
- 8-Bit control registers
- Software-selectable acknowledge bit
- On-chip filtering for spikes on the bus

## 14.3 Block Diagram

Figure 14-1 is a block diagram of the $I^2C$ block. The clock source to the module is the Peripheral Clock.

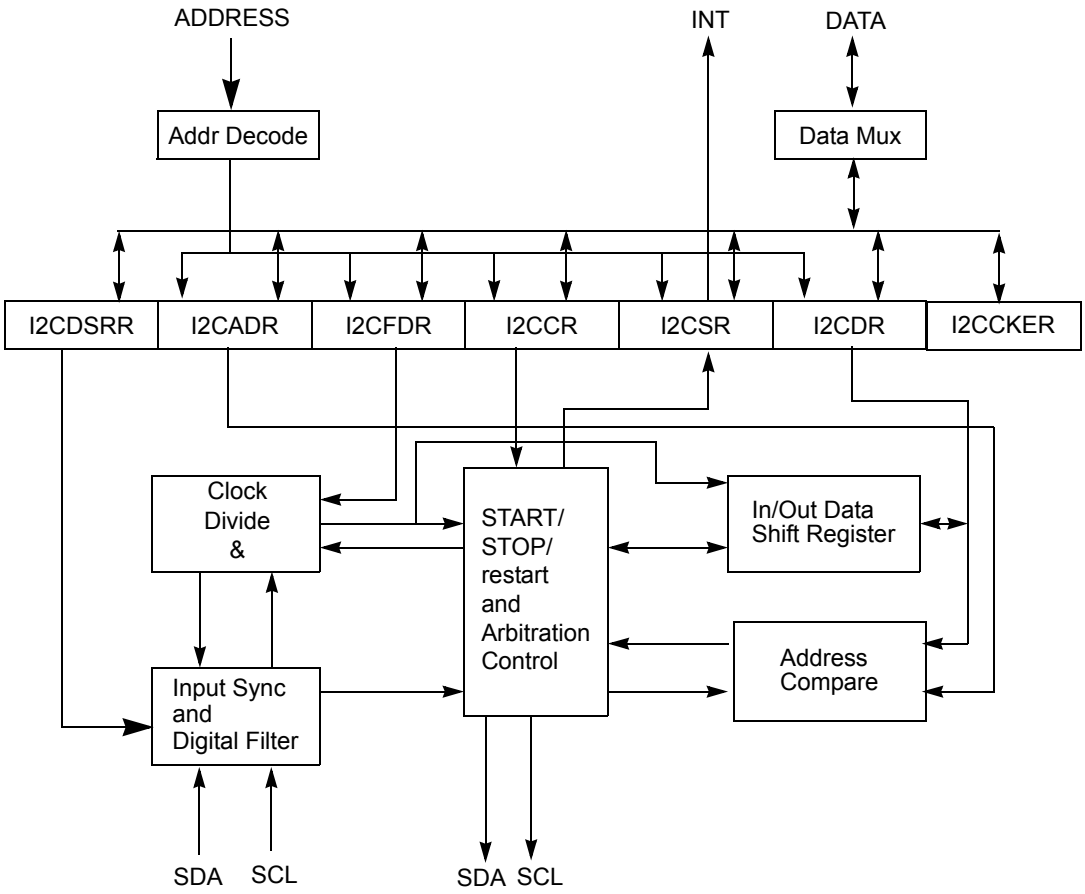



**Figure 14-1. $I^2C$ Controller Block Diagram**

## 14.4 Functional Description

The following sections describe the module functionality.

### 14.4.1 Signal Description

The $I^2C$ interface uses the SDA signal and SCL signal for data transfer. All devices that are connected to these two signals must have open drain or open collector outputs. A logical AND function is performed on both signals with external pull-up resistors.

Table 14-1 summarizes the signals that comprise the external $I^2C$ interface.

**Table 14-1. I²C Interface Signal Properties**

| Signal Names | Direction | Active | Description/Comments |
|---|---|---|---|
| SCL | Input/Output | 0 | Bus clock. Open drain output. External pull-up required |
| SDA | Input/Output | 0 | Bus clock. Open drain output. External pull-up required |

## 14.4.2 Modes of Operation

The following modes of operation are supported by the I²C controller:

- Master mode. The I²C is the driver of the SDA line. It cannot use its own slave address as a calling address. The I²C cannot be a master and a slave simultaneously.
- Slave mode. The I²C is not the driver of the SDA line. The module must be enabled before a START condition from a non-I²C master is detected.
- START condition. This condition denotes the beginning of a new data transfer (each data transfer contains several bytes of data) and awakens all slaves. This mode is I²C-specific.
- Repeated START condition. A START condition that is generated without a STOP condition to terminate the previous transfer. This mode is I²C-specific.
- STOP condition. The master can terminate the transfer by generating a STOP condition to free the bus. This mode is I²C-specific.
- Interrupt-driven byte-to-byte data transfer. When successful slave addressing is achieved (and SCL returns to zero), the data transfer can proceed on a byte-to-byte basis in the direction specified by the R/W bit sent by the calling master. Each byte of data must be followed by an acknowledge bit, which is signalled from the receiving device. Several bytes can be transferred during a data transfer session.

## 14.4.3 I²C Protocol Description

A standard I²C transfer consists of the following:

- START condition
- Slave target address transmission
- Data transfer
- STOP condition

Figure 14-2 shows the interaction of these four parts with the calling address, data byte, and new calling address components of the I²C protocol. The details of the protocol are described in the following subsections.

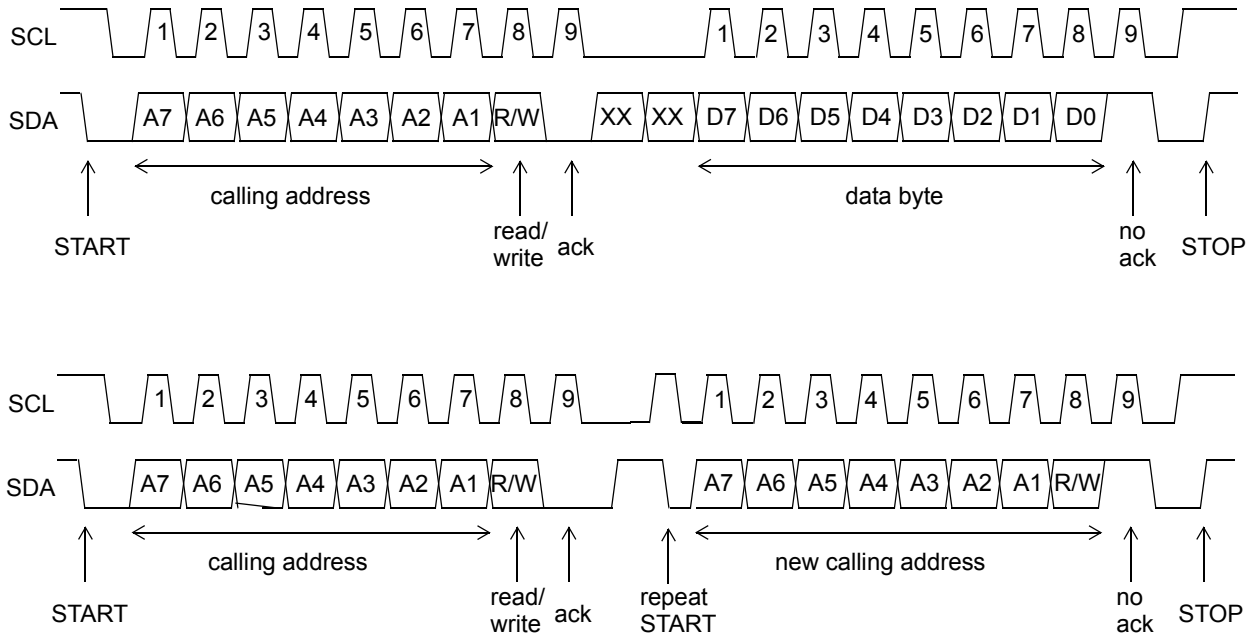



**Figure 14-2. I²C Interface Transaction Protocol**

### 14.4.3.1 START Condition

When the I²C bus is not engaged (both SDA and SCL lines are at logic high), a master can initiate a transfer by sending a START condition. As shown in Figure 14-2, a START condition is defined as a high-to-low transition of SDA while SCL is high. This condition denotes the beginning of a new data transfer. Each data transfer can contain several bytes and awakens all slaves. The START condition is initiated by a software write that sets the I2CCR[MSTA].

### 14.4.3.2 Slave Address Transmission

The first byte of data is transferred by the master immediately after the START condition is the slave address. This is a seven-bit calling address followed by a R/W bit, which indicates the direction of the data being transferred to the slave. Each slave in the system has a unique address. In addition, when the I²C module is operating as a master, it must not transmit an address that is the same as its slave address. An I²C device cannot be master and slave at the same time.

Only the slave with a calling address that matches the one transmitted by the master responds by returning an acknowledge bit (pulling the SDA signal low at the 9th clock) as shown in Figure 14-2. If no slave acknowledges the address, the master should generate a STOP condition or a repeated START condition.

When slave addressing is successful (and SCL returns to zero), the data transfer can proceed on a byte-to-byte basis in the direction specified by the R/W bit sent by the calling master.

The I²C module responds to a general call (broadcast) command when I2CCR[BCST] is set. A broadcast address is always zero, and the R/W bit is zero.

Each data byte is 8 bits long. Data bits can be changed only while SCL is low and must be held stable while SCL is high, as shown in Figure 14-2. There is one clock pulse on SCL for each data bit, and the most

significant bit (MSB) is transmitted first. Each byte of data must be followed by an acknowledge bit, which is signalled from the receiving device by pulling the SDA line low at the ninth clock. Therefore, one complete data byte transfer takes nine clock pulses. Several bytes can be transferred during a data transfer session.

If the slave receiver does not acknowledge the master, the SDA line must be left high by the slave. The master can then generate a stop condition to abort the data transfer or a START condition (repeated START) to begin a new calling.

If the master receiver does not acknowledge the slave transmitter after a byte of transmission, the slave interprets that the end-of-data has been reached. Then the slave releases the SDA line for the master to generate a STOP or a START condition.

#### 14.4.3.3 Repeated START Condition

Figure 14-2 shows a repeated START condition, which is generated without a STOP condition that can terminate the previous transfer. The master uses this method to communicate with another slave or with the same slave in a different mode (transmit/receive mode) without releasing the bus.

#### 14.4.3.4 STOP Condition

The master can terminate the transfer by generating a STOP condition to free the bus. A STOP condition is defined as a low-to-high transition of the SDA signal while SCL is high. For more information, see Figure 14-2. Note that a master can generate a STOP even if the slave has transmitted an acknowledge bit, at which point the slave must release the bus. The STOP condition is initiated by a software write that clears I2CCR[MSTA].

As described in Section 14.4.3.3, “Repeated START Condition”, the master can generate a START condition followed by a calling address without generating a STOP condition for the previous transfer. This is called a repeated START condition.

#### 14.4.3.5 Arbitration Procedure

The $I^2C$ interface is a true multiple master bus that allows more than one master device to be connected on it. If two or more masters simultaneously try to control the bus, each master’s (including the $I^2C$ module) clock synchronization procedure determines the bus clock—the low period is equal to the longest clock low period and the high is equal to the shortest one among the masters. A bus master loses arbitration if it transmits a logic 1 on SDA while another master transmits a logic 0. The losing masters immediately switch to slave-receive mode and stop driving the SDA line. In this case, the transition from master to slave mode does not generate a STOP condition. Meanwhile, the $I^2C$ unit sets the I2CSR[MAL] status bit to indicate the loss of arbitration and, as a slave, services the transaction if it is directed to itself.

Arbitration is lost (and I2CSR[MAL] is set) in the following circumstances:

- SDA sampled as low when the master drives a high during address or data-transmit cycle.
- SDA sampled as low when the master drives a high during the acknowledge bit of a data-receive cycle.
- A START condition is attempted when the bus is busy.

- A repeated START condition is requested in slave mode.

Note that the $I^2C$ module does not automatically retry a failed transfer attempt.

If the $I^2C$ module is enabled in the middle of an ongoing byte transfer, the interface behaves as follows:

- Slave mode—the $I^2C$ module ignores the current transfer on the bus and starts operating whenever a subsequent START condition is detected.
- Master mode—the $I^2C$ module cannot tell whether the bus is busy; therefore, if a START condition is initiated, the current bus cycle can be corrupted. This ultimately results in the current bus master of the $I^2C$ interface losing arbitration, after which bus operations return to normal.

#### 14.4.3.6 Clock Synchronization

Due to the wire AND logic on the SCL line, a high-to-low transition on the SCL line affects all devices connected on the bus. The devices begin counting their low period when the master drives the SCL line low. After a device has driven SCL low, it holds the SCL line low until the clock high state is reached. However, the change of low-to-high in a device clock may not change the state of the SCL line if another device 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. When all devices concerned have counted off their low period, the synchronized SCL line is released and pulled high. Then there is no difference between the devices' clocks and the state of the SCL line, and all the devices begin counting their high periods. The first device to complete its high period pulls the SCL line low again.

#### 14.4.3.7 Handshaking

The clock synchronization mechanism can be used as a handshake in data transfer. Slave devices can hold the SCL low after completion of a 1-byte transfer (9 bits). In such cases, it halts the bus clock and forces the master clock into wait states until the slave releases the SCL line.

#### 14.4.3.8 Clock Stretching

Slaves can use the clock synchronization mechanism to slow down the transfer bit rate. After the master has driven the SCL line 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.

## 14.4.4 Description of I²C Functional Blocks

The following sections provide additional information on the operation of the I²C module.

### 14.4.4.1 Clock Control

The clock control block handles requests from clock for transferring and controlling data for multiple tasks.

A 9-cycle data transfer clock is requested for the following conditions:

- Master mode
- Transmit slave address after START condition
- Transmit slave address after restart condition
- Transmit data
- Receive data
- Slave mode
- Transmit data
- Receive data
- Receive slave address after START or restart condition

### 14.4.4.2 Input Synchronization and Digital Filter

The input synchronization block synchronizes the input SCL and SDA signals to the system clock and detects transitions of these signals. In addition, the SCL and SDA input pins are filtered to eliminate noise. Three consecutive samples of the SCL and SDA lines are compared with a pre-determined sampling rate. If they are all high, the output of the filter is high. If they are all low, the output is low. If there are any combination of highs and lows, the output is whatever the value of the line was in the previous clock cycle.

The sampling rate is equal to a binary value stored in the frequency register. The duration of the sampling cycle is controlled by a down counter and a signal that sets at the end of the count. This allows a software write to the frequency register to control the filtered sampling rate.

#### 14.4.4.2.1 Transaction Monitoring

The different conditions of the I²C data transfers are monitored as follows:

- START conditions are detected when an SDA fall occurs while SCL is high.
- STOP conditions are detected when and SDA rise occurs while SCL is high.
- Data transfers in progress are cancelled when a STOP condition is detected or if there is a slave address mismatch. Cancellation of data transactions resets the clock module
- The bus is detected to be busy upon the detection of a START condition, and idle upon the detection of a STOP condition.

#### 14.4.4.3 Arbitration Control

The arbitration control block controls the arbitration procedure of the master mode. A loss of arbitration occurs whenever the master detects a 0 on the external SDA line while attempting to drive a 1, tries to generate a START or restart at an inappropriate time, or detects an unexpected STOP request on the line.

Arbitration by the master in master mode is lost under the following conditions—

- Low detected when high expected (transmit)
- Ack bit, low detected when high expected (receive)
- A START condition is attempted when the bus is busy
- A START condition is attempted when the bus is nearly busy
- A start condition is attempted when the requesting device is not the bus owner
- Unexpected STOP condition detected

#### 14.4.4.4 Control Transfer

The $I^2C$ contains logic that controls the output to the serial data (SDA) and serial clock (SCL) lines of the $I^2C$. The SCL output is pulled low as determined by the internal clock generated in the clock module. The SDA output can only change at the midpoint of a low cycle of the SCL, unless performing a START, STOP, or restart condition. Otherwise, the SDA output is held constant.

The SDA signal is pulled low when one or more of the following conditions are true in either master or slave mode—

- Master mode
- Data bit (transmit)
- Ack bit (receive)
- START condition
- STOP condition
- Restart condition
- Slave mode
- Acknowledging address match
- Data bit (transmit)
- Ack bit (receive)

The SCL signal corresponds to the internal SCL signal when one or more of the following conditions are true in either master or slave mode—

- Master mode
- Bus owner
- Lost arbitration
- START condition
- STOP condition
- Restart condition begin

- Restart condition end
- Slave mode
- Address cycle
- Transmit cycle
- Ack cycle

#### 14.4.4.5 In/Out Data Shift Register

This block controls the interface between the serial data line and the data register, both transmitting data to and receiving data from the $I^2C$.

In transmit mode, a write to I2CCR[MTX] sets the direction to transmit. The contents of yicp_tx_data are loaded into a shift register, which are then loaded to yici_rx_data. The contents of yici_rx_data are then shifted out to shift_sda, which eventually is routed out to yici_sda_out.

#### 14.4.4.6 Address Compare

Address compare block determines if a slave has been properly addressed, either by its slave address or by the general broadcast address (which addresses all slaves). The four performed address comparisons are described as follows:

- The address sent by the current master matches the general broadcast address.
- The address matches either the broadcast address or the slave address; in either case, slave mode results.
- Whether a broadcast message has been received, to update the I2CSR and to generate an interrupt.
- Whether the module has been addressed as a slave, to update the I2CSR and to generate an interrupt.

## 14.5 Reset

The reset state of the $I^2C$ is that the module is disabled. Once enabled, the default condition is that the interface will perform as a slave receiver, unless explicitly programmed to be a master or slave transmitter address.

## 14.6 Interrupts

There is a single interrupt request signal from the $I^2C$ module. The interrupt request is active if the interrupt status bit MIF (Register I2CSR, Bit 1) is set, and the request is enabled by bit MIEN enable (Register I2CCR, Bit 6).

MIF is set when one of the following events occurs (See Section 14.8.4, “I2C Status Register (I2CSR)” for more details):

- One byte of data is transferred (set at the falling edge of the 9th clock). Status bit MCF is set.
- The value in I2CADR matches with the calling address in slave-receive mode. Status bit MAAS is set.

- Arbitration is lost. Status bit MAL is set.

## 14.7 I²C Register Memory Map.

The I²C module is programmed via a set of memory-mapped registers and their respective offset addresses are listed in Table 14-2.

- The I²C base address is **0x8000_6000**
- All registers are 8 bits wide and are accessed only as byte-access on a long word boundary.

**Table 14-2. I²C Module Register Memory Map**

| Address | Register Name | Access Type | Access Width |
|---|---|---|---|
| Base + 0x00 | I²C Address Register (I2CADR) | R/W | 8-bit only |
| Base + 0x04 | I²C Frequency Divider Register (I2CFDR) | R/W | 8-bit only |
| Base + 0x08 | I²C Control Register (I2CCR) | R/W | 8-bit only |
| Base + 0x0C | I²C Status Register (I2CSR) | R/W | 8-bit only |
| Base + 0x10 | I²C Data Register (I2CDR) | R/W | 8-bit only |
| Base + 0x14 | I²C Digital Filter Sampling Rate Register (I2CDFSRR) | R/W | 8-bit only |
| Base + 0x18 | I²C Clock Enable Register (I2CCKER) | R/W | 8-bit only |

## 14.8 I²C Registers

The following sections provide descriptions of the I²C registers.

### 14.8.1 I²C Address Register (I2CADR)

The I²C Address Register (I2CADR) contains the address to which the I²C interface responds when addressed as a slave.

**NOTE**

This is not the address that is sent on the bus during the address-calling cycle when the I²C module is in master mode.

| **I2CADR** | | | | | **Addr Base+0x00** | | | |
|---|---|---|---|---|---|---|---|---|
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | ADDR | | | | | | | RSV |
| TYPE | rw | rw | rw | rw | rw | rw | rw | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 14-3. I2C Address Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 7-1 | **ADDR[6:0] -** Slave Address. This field contains the slave address that is used by the $I^2C$ interface. Note that the default mode of the $I^2C$ interface is slave mode for an address match. | |
| 0 | **Reserved** | Read only |

## 14.8.2 $I^2C$ Frequency Divider Register (I2CFDR)

The $I^2C$ Frequency Divider Register (I2CFDR) sets the sampling rate and the clock bit rate for the $I^2C$ interface. The clock source to the $I^2C$ is the Peripheral Clock which is controlled via the CRM Control Register. The bit rate clock is generated from the peripheral clock:

Bit Rate = Peripheral Clock / (Divider Ratio)

| **I2CFDR** | | | | | **Addr Base+0x04** | | | |
|---|---|---|---|---|---|---|---|---|
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | Reserved | | FDR | | | | | |
| TYPE | r | r | rw | rw | rw | rw | rw | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 14-4. $I^2C$ Address Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 7-6 | **Reserved** | Read only |
| 5-0 | **FDR[5:0] -** Frequency Divider Ratio. This field is used to prescale the clock for bit rate selection.<br>• The serial bit clock frequency of SCL is equal to the Peripheral Clock divided by the divider ratio.<br>• The frequency divider value can be changed at any point in a program. | See Table 14-5. |

Table 14-5 shows the divider ratio versus FDR[5:0] field setting. The incoming peripheral clock frequency to the module is the reference frequency divided by the XTAL_CLKDIV (CRM Control Register).

**Table 14-5. $I^2C$ Clock Divide Ratio vs. FDR Setting**

| FDR[5:0] | Divider (Decimal) | FDR[5:0] | Divider (Decimal) |
|---|---|---|---|
| 0x00 | 288 | 0x20 | 160 |
| 0x01 | 320 | 0x21 | 192 |

**Table 14-5. $I^2C$ Clock Divide Ratio vs. FDR Setting (continued)**

| FDR[5:0] | Divider (Decimal) | FDR[5:0] | Divider (Decimal) |
|---|---|---|---|
| 0x02 | 384 | 0x22 | 224 |
| 0x03 | 480 | 0x23 | 256 |
| 0x04 | 576 | 0x24 | 320 |
| 0x05 | 640 | 0x25 | 384 |
| 0x06 | 768 | 0x26 | 448 |
| 0x07 | 960 | 0x27 | 512 |
| 0x08 | 1152 | 0x28 | 640 |
| 0x09 | 1280 | 0x29 | 768 |
| 0x0A | 1536 | 0x2A | 896 |
| 0x0B | 1920 | 0x2B | 1024 |
| 0x0C | 2304 | 0x2C | 1280 |
| 0x0D | 2560 | 0x2D | 1536 |
| 0x0E | 3072 | 0x2E | 1792 |
| 0x0F | 3840 | 0x2F | 2048 |
| 0x10 | 4608 | 0x30 | 2560 |
| 0x11 | 5120 | 0x31 | 3072 |
| 0x12 | 6144 | 0x32 | 3584 |
| 0x13 | 7680 | 0x33 | 4096 |
| 0x14 | 9216 | 0x34 | 5120 |
| 0x15 | 10240 | 0x35 | 6144 |
| 0x16 | 12288 | 0x36 | 7168 |
| 0x17 | 15360 | 0x37 | 8192 |
| 0x18 | 18432 | 0x38 | 10240 |
| 0x19 | 20480 | 0x39 | 12288 |
| 0x1A | 24576 | 0x3A | 14336 |
| 0x1B | 30720 | 0x3B | 16384 |
| 0x1C | 36864 | 0x3C | 20480 |
| 0x1D | 40960 | 0x3D | 24576 |
| 0x1E | 49152 | 0x3E | 28672 |
| 0x1F | 61440 | 0x3F | 32768 |

## 14.8.3 I²C Control Register (I2CCR)

The I²C Control Register (I2CCR) sets the modes for the I²C interface.

| I2CCR | | | | | Addr Base+0x08 | | | |
|---|---|---|---|---|---|---|---|---|
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | MEN | MIEN | MSTA | MTX | TXAK | RSTA | RSV | BCST |
| TYPE | rw | rw | rw | rw | rw | w | r | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 14-6. I²C Control Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 7 | **MEN -** Module Enable. This bit enables the I²C module.<br>• This bit must be set before any other control register bits have any effect.<br>• All I²C registers for slave receive or master START can be initialized before setting this bit.<br>• When low, the interface is held in reset but the registers can still be accessed. | 1 = The I²C module is enabled.<br>0 = The module is disabled (default) |
| 6 | **MIEN -** Module Interrupt Mask. This bit enables interrupt requests. An interrupt occurs if I2CSR[MIF] is set. | 1 = Interrupt requests enabled<br>0 = Interrupt requests disabled (default) |
| 5 | **MSTA -** Master/Slave Mode START. Mode start bit. | 0 = When this bit is changed from a 1 to a 0, a STOP condition is generated and the mode changes from master to slave.<br>1 = When this bit is changed from a 0 to a 1, a START condition is generated on the bus, and the master mode is selected. The bit is cleared without generating a STOP condition when the master loses arbitration. |
| 4 | **MTX -** Transmit/Receive Mode Select. This bit selects the direction of the master and slave transfers.<br>• When configured as a slave, this bit should be set by software according to I2CSR[SRW].<br>• In master mode, the bit should be set according to the type of transfer required.<br>• Therefore, for address cycles, this bit will always be high.<br>• The MTX bit is cleared when the master loses arbitration. | 1 = Transmit mode<br>0 = Receive mode (default) |

| Bit Number | Description | Operation |
|---|---|---|
| 3 | **TXAK -** Transfer acknowledge. This bit specifies the value driven onto the SDA line during acknowledge cycles for both master and slave receivers.<br>• The value of this bit only applies when the $I^2C$ module is configured as a receiver, not a transmitter.<br>• It also does not apply to address cycles; when the core is addressed as a slave, an acknowledge is always sent. | 1 = No acknowledge signal response (high value on SDA) is sent.<br>0 = An acknowledge signal (low value on SDA) is sent out to the bus on the 9th clock bit after receiving one byte of data. |
| 2 | **RSTA -** Repeat START. Setting this bit always generates a repeated START condition on the bus and provides the core with the current bus master.<br>• Attempting a repeated START at the wrong time (or if the bus is owned by another master), results in loss of arbitration.<br>• This bit is write only. | 1 = Generates repeat START condition.<br>0 = No START condition is generated. |
| 1 | **Reserved** | Read only |
| 0 | **BCST -** Broadcast. This bit controls broadcast receive function. | 1 = Enables the $I^2C$ to accept broadcast messages at address zero.<br>0 = Disables the broadcast accept capability. |

## 14.8.4 $I^2C$ Status Register (I2CSR)

The $I^2C$ Status Register (I2CCR) reports the status for the $I^2C$ interface.

| **I2CSR** | | | | | **Addr Base+0x0C** | | | |
|---|---|---|---|---|---|---|---|---|
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | MCF | MAAS | MBB | MAL | BCSTM | SRW | MIF | RXAK |
| TYPE | r | r | r | rw | r | r | rw | r |
| RESET | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

**Table 14-7. $I^2C$ Status Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 7 | **MCF -** Data Transfer. This bit indicates when data is transferred.<br>• When one byte of data is transferred, the bit is set.<br>• It is set by the falling edge of the 9th clock of a byte transfer.<br>• MCF is cleared under two following conditions—<br>- when I2CSR is read in receive mode or<br>- when I2CDR is written in transmit mode | 0 = Byte transfer in progress<br>1 = Byte transfer is completed |
| 6 | **MAAS** - Addressed as a Slave. When the value in I2CDR matches with the calling address, this bit is set.<br>• An interrupt request is generated, if I2CCR[MIEN] is set.<br>• The processor must check the SRW bit and set I2CCR[MTX] accordingly.<br>• Writing to the I2CCR automatically clears this bit | 0 = Not addressed as a slave<br>1 = Addressed as a slave |
| 5 | **MBB -** Bus busy. This bit indicates the status of the bus. When a START condition is detected, MBB is set. If a STOP condition is detected, it is cleared. | 0 = $I^2C$ bus is idle<br>1 = $I^2C$ bus is busy |
| 4 | **MAL -** Arbitration lost. This bit is automatically set when the arbitration procedure is lost.<br>• Note that the core does not automatically retry a failed transfer attempt.<br>• This bit can only be cleared by software. | 0 = Arbitration is not lost.<br>1 = Arbitration is lost |
| 3 | **BCSTM -** Broadcast match. This bit indicates a broadcast address match.<br>**Note:** This will also be set if the $I^2C$ drives an address of all 0s and broadcast mode is enabled. | 0 = There has not been a broadcast match.<br>1 = The calling address matches with the broadcast address instead of the programmed slave address. |
| 2 | **SRW -** Slave read/write. When MAAS is set, SRW indicates the value of the R/W command bit of the calling address, which is sent from the master.<br>• This bit is valid only when both of the following conditions are true:<br>- A complete transfer occurred and no other transfers have been initiated<br>- The $I^2C$ interface is configured as a slave and has an address match.<br>• By checking this bit, the processor can select slave transmit/receive mode according to the command of the master. | 0 = Slave receive, master writing to slave<br>1 = Slave transmit, master reading from slave. |

| Bit Number | Description | Operation |
|---|---|---|
| 1 | **MIF -** Module interrupt. The MIF bit is set when an interrupt request is pending. It can be cleared only by software write.<br><br>MIF is set when one of the following events occurs:<br>• One byte of data is transferred (set at the falling edge of the 9th clock)<br>• The value in I2CADR matches with the calling address in slave-receive mode<br>• Arbitration is lost | 0 = No interrupt is pending.<br>1 = Interrupt is pending. |
| 0 | **RXAK -** Received acknowledge. This bit is the value of SDA during the reception of acknowledge bit of a bus cycle.<br>• If the received acknowledge bit (RXAK) is low, it indicates that an acknowledge signal has been received after the completion of eight bits of data transmission on the bus.<br>• If RXAK is high, it means no acknowledge signal has been detected at the 9th clock. | 0 = Acknowledge received<br>1 = No acknowledge received |

## 14.8.5 $I^2C$ Data Register (I2CDR)

The $I^2C$ Data Register (I2CDR) is used to send data to or receive data from the $I^2C$ interface.

| **I2CDR** | | | | | **Addr Base+0x10** | | | |
|---|---|---|---|---|---|---|---|---|
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | DATA | | | | | | | |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 14-8. I²C Data Register Field Description**

| Bit Number | Description | Operation |
|---|---|---|
| 7 | **DATA[7:0] -** Data Byte. This register controls bus activity.<br>• For master mode, transmission starts when an address and the R/W bit are written to the data register (the I²C interface performs as the master).<br>• A data transfer is initiated when data is written to the I2CDR.<br>• The most significant bit is sent first in both cases.<br>• In the master receive mode, reading the data register allows the read to occur, but also allows the I²C module to receive the next byte of data on the I²C interface.<br>• In slave mode, the same function is available after it is addressed | |

## 14.8.6 I²C Digital Filter Sampling Rate Register (I2CDFSRR)

The I²C Digital Filter Sampling Rate Register (I2CDFSRR) sets the sample rate for the I²C interface.

| **I2CDFSRR** | | | | | **Addr Base+0x14** | | | |
|---|---|---|---|---|---|---|---|---|
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | Reserved | | DFSR | | | | | |
| TYPE | r | r | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

**Table 14-9. I²C Digital Filter Sampling Rate Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 7-6 | **Reserved** | Read only |
| 5-0 | **DFSR[5:0]** - Digital filter sampling rate. To assist in filtering out signal noise, the sample rate is programmed.<br>• This field is used to prescale the frequency at which the digital filter takes samples from the I²C bus.<br>• The resulting sampling rate is calculated by dividing the peripheral clock by the non-zero value of DFSR.<br>• If I2CDFSRR is set to zero, the I²C bus sample rate defaults to the reset divisor 0x10. | 0x10 = Default<br><br>Sample rate = (peripheral clock) / DFSR[5:0] |

### 14.8.7 I²C Clock Enable Register (I2CCKER)

The I²C Clock Enable Register (I2CCKER) controls the peripheral clock to the I²C module.

**NOTE**

Bit CKEN must be written high before any other I²C register accesses or any I²C activity.

| I2CCKER | | | | | Addr Base+0x18 | | | |
|---|---|---|---|---|---|---|---|---|
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | Reserved | | | | | | | CKEN |
| TYPE | r | r | r | r | r | r | r | rw |
| RESET | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

**Table 14-10. I²C Clock Enable Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 7-1 | Reserved | Read only |
| 0 | CKEN - Clock Enable bit. When asserted, the clock to I2C is running continuously from the CRM. When de-asserted, the clock to I2C is gated off by the CRM unless a register write to the I2C is detected by the CRM.<br>**Note: This bit must be written high by software before any other I2C register writes, and kept high during normal I2C operation**. | 1 = Clock enabled<br>0 = Clock not enabled (default) |

## 14.9 Initialization/Application Information

This section describes some programming guidelines recommended for the I²C interface and includes Figure 14-3, a recommended flow chart for the I²C interrupt service routines.

**NOTE**

This information is provided for understanding and use of the I²C module, and it is highly recommended to use the software utility entitled the "Inter-integrated Circuit (I2C) Driver" (see Appendix B, "MC1322x Software Driver Utilities") to implement an I²C service.

Good design practice provides for illegal/malfunctioning I²C bus usage:

- The I²C controller does not guarantee its recovery from all illegal I²C bus activity.
- A malfunctioning device may hold the bus captive.
- Good programming practice is to provide a watchdog timer to help recovery from I²C bus hang-ups.

- Recovery service should also handle the case when the status bits returned after an interrupt request are not consistent with expected behavior, due to illegal $I^2C$ bus protocol behavior.

### 14.9.1 Initialization Sequence

From a default condition, the following sequence initializes the $I^2C$ unit:

- Update I2CFDR[FDR] and select the required division ratio to obtain the SCL frequency from the platform clock.
- Update I2CADR to define the slave address for this device.
- Modify I2CCR to select master/slave mode, transmit/receive mode, and interrupt-enable or disable.
- Set the I2CCR[MEN] to enable the $I^2C$ interface.

### 14.9.2 Generation of START

After initialization, the following sequence can be used to generate START:

- If the device is connected to a multi-master $I^2C$ system, test the state of I2CSR[MBB] to check whether the serial bus is free (I2CSR[MBB] = 0) before switching to master mode.
- Select master mode (set I2CCR[MSTA]) to transmit serial data and select transmit mode (set I2CCR[MTX]) for the address cycle.
- Write the slave address being called into the data register (I2CDR). The data written to I2CDR comprises the slave calling address. I2CCR[MTX] indicates the direction of transfer (transmit/receive) required from the slave.

This scenario assumes that the $I^2C$ interrupt bit (I2CSR[MIF]) is cleared. If I2CSR[MIF] = 1 at any time, the $I^2C$ interrupt handler should immediately handle the interrupt.

### 14.9.3 Post-Transfer Software Response

Transmission or reception of a byte automatically sets the data transferring bit (I2CSR[MCF]), which indicates that one byte has been transferred. The $I^2C$ interrupt bit (I2CSR[MIF]) is also set; an interrupt is generated to the processor if the interrupt function is enabled during the initialization sequence (I2CCR[MIEN] = 1). In the interrupt handler, software must do the following:

- Clear I2CSR[MIF]
- Read the contents of the $I^2C$ data register (I2CDR) in receive mode or write to I2CDR in transmit mode. Note that this causes I2CSR[MCF] to be cleared.

When an interrupt occurs at the end of the address cycle, the master remains in transmit mode. If master receive mode is required, I2CCR[MTX] should be toggled at this stage.

If the interrupt function is disabled, software can service the I2CDR in the main program by monitoring I2CSR[MIF]. In this case, I2CSR[MIF] should be polled rather than I2CSR[MCF] because MCF behaves differently when arbitration is lost. Note that interrupt or other bus conditions may be detected before the $I^2C$ signals have time to settle. Thus, when polling I2CSR[MIF] (or any other I2CSR bits), software delays may be needed (in order to give the $I^2C$ signals sufficient time to settle).

During slave-mode address cycles (I2CSR[MAAS] = 1), I2CSR[SRW] should be read to determine the direction of the subsequent transfer and I2CCR[MTX] should be programmed accordingly. For slave-mode data cycles (I2CSR[MAAS] = 0), I2CSR[SRW] is not valid and I2CCR[MTX] should be read to determine the direction of the current transfer.

### 14.9.4 Generation of STOP

A data transfer ends with a STOP condition generated by the master device. A master transmitter can generate a STOP condition after all the data has been transmitted.

If a master receiver wants to terminate a data transfer, it must inform the slave transmitter by not acknowledging the last byte of data, which is done by setting the transmit acknowledge (I2CCR[TXAK]) bit before reading the next-to-last byte of data At this time, the next-to-last byte of data has already been transferred on the $I^2C$ interface, so the last byte will not receive the data acknowledge when I2CCR[TXAK] is set. For 1-byte transfers, a dummy read should be performed by the interrupt service routine. Before the interrupt service routine reads the last byte of data, a STOP condition must first be generated.

The $I^2C$ automatically generates a STOP if I2CCR[TXAK] = 1. Therefore, I2CCR[TXAK] must be set to a 1 before allowing the $I^2C$ module to receive the last data byte on the $I^2C$ bus. Eventually, I2CCR[TXAK] will need to be cleared again for subsequent $I^2C$ transactions. This can be accomplished when setting up the I2CCR for the next transfer.

### 14.9.5 Generation of Repeated START

At the end of a data transfer, if the master still wants to communicate on the bus, it can generate another START condition followed by another slave address without first generating a STOP condition by setting I2CCR[RSTA].

## 14.10 Generation of SCL When SDA Low

In some cases it is necessary to force the $I^2C$ module to become the $I^2C$ bus master out of reset and drive the SCL signal (even though SDA may already be driven, which indicates that the bus is busy). This can occur when a system reset does not cause all $I^2C$ devices to be reset. Thus, the SDA signal can be driven low by another $I^2C$ device while the $I^2C$ module is coming out of reset and will stay low indefinitely. The following procedure can be used to force the $I^2C$ module to generate SCL so that the device driving SDA can finish its transaction:

- Disable the $I^2C$ and set the master bit by setting I2CCR to 0x20.
- Enable the $I^2C$ by setting I2CCR to 0xA0.
- Read the I2CDR.
- Return the $I^2C$ module to slave mode by setting I2CCR to 0x80.

### 14.10.1 Slave Mode Interrupt Service Routine

In the slave interrupt service routine, the module addressed as a slave should be tested to check if a calling of its own address has been received. If I2CSR[MAAS] = 1, software should set the transmit/receive mode select bit (I2CCR[MTX]) according to the R/$\overline{W}$ command bit (I2CSR[SRW]). Writing to I2CCR clears I2CSR[MAAS] automatically. The only time I2CSR[MAAS] is read as set is from the interrupt handler at the end of that address cycle where an address match occurred; interrupts resulting from subsequent data transfers will have I2CSR[MAAS] = 0. A data transfer can then be initiated by writing to I2CDR for slave transmits or dummy reading from I2CDR in slave-receive mode. The slave drives SCL low between byte transfers. SCL is released when the I2CDR is accessed in the required mode.

#### 14.10.1.1 Slave Transmitter and Received Acknowledge

In the slave transmitter routine, the received acknowledge bit (I2CSR[RXAK]) must be tested before sending the next byte of data. The master signals an end-of-data by not acknowledging the data transfer from the slave. When no acknowledge is received (I2CSR[RXAK] = 1), the slave transmitter interrupt routine must clear I2CCR[MTX] to switch the slave from transmitter to receiver mode. A dummy read of I2CDR then releases SCL so that the master can generate a STOP condition.

#### 14.10.1.2 Loss of Arbitration and Forcing of Slave Mode

When a master loses arbitration the following conditions all occur:

- I2CSR[MAL] is set
- I2CCR[MSTA] is cleared (changing the master to slave mode)
- An interrupt occurs (if enabled) at the falling edge of the 9th clock of this transfer

Thus, the slave interrupt service routine should test I2CSR[MAL] first, and the software should clear I2CSR[MAL] if it is set.

### 14.10.2 Interrupt Service Routine Flow Chart

Figure 14-3 shows an example algorithm for an $I^2C$ interrupt service routine. Deviation from the flow chart may result in unpredictable $I^2C$ bus behavior. However, in the slave receive mode (not shown in the flow chart), the interrupt service routine may need to set I2CCR[TXAK] when the next-to-last byte is to be accepted. It is recommended that a sync instruction follow each $I^2C$ register read or write to guarantee in-order instruction execution.

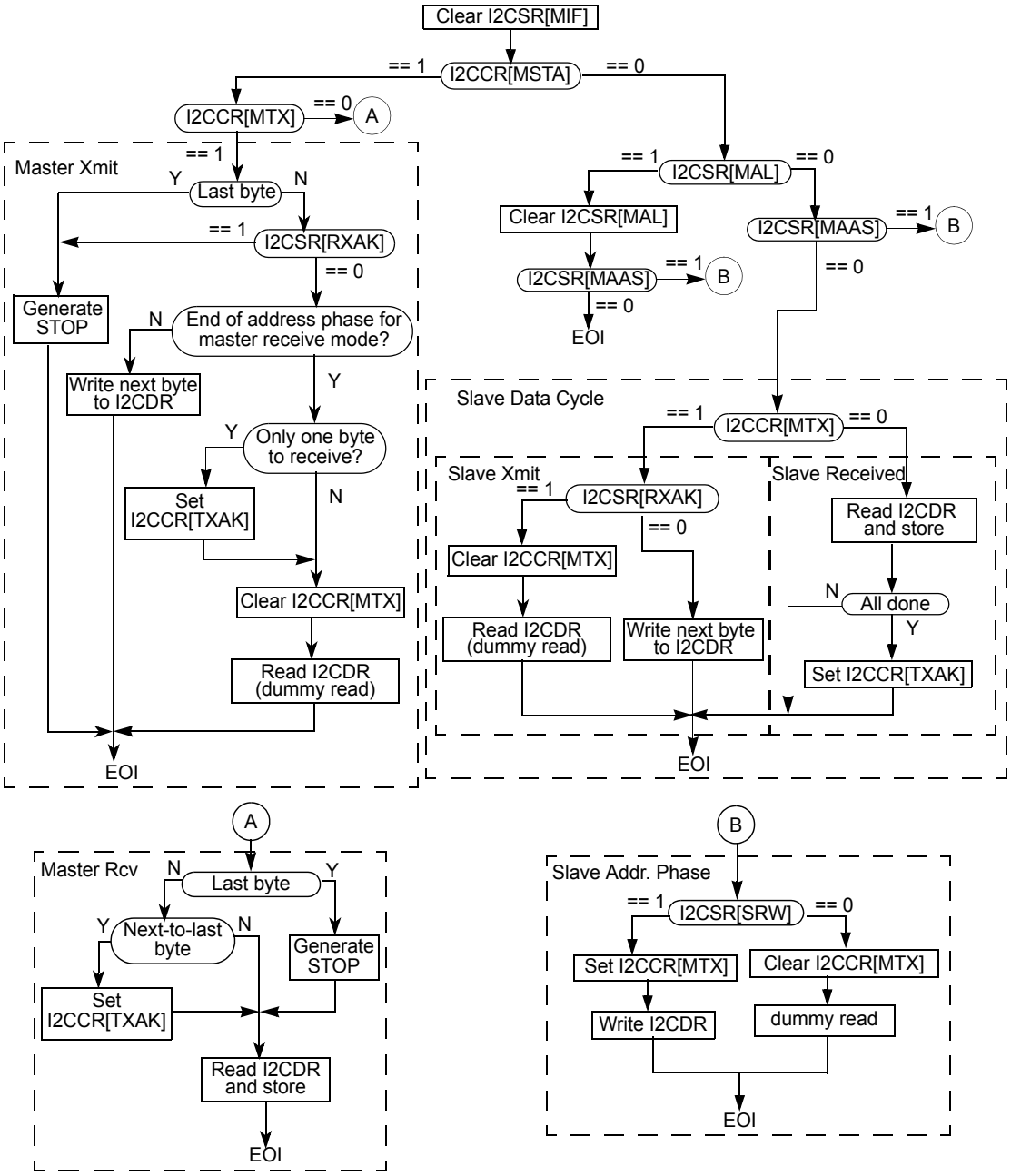



**Figure 14-3. Example I²C Interrupt Service Routine Flow Chart**

# Chapter 15
# Serial Peripheral Interface Module (SPI)

## 15.1 Overview

This chapter describes the Serial Peripheral Interface (SPI) module for external use. The MC1322x SPI is a high-speed synchronous serial data input/output port used for interfacing with serial memories, peripheral devices, or other processors. The SPI allows a serial bit stream of a programmed length (1 to 32 bits) to be shifted simultaneously into and out of the device at a programmed bit-transfer rate (called 4-wire mode). There are four pins associated with the SPI port (SPI_SCK, SPI_MOSI, SPI_MISO, and SPI_SS).

The SPI module can be programmed for master or slave operation. It also supports a 3-wire mode where for master mode, the MOSI becomes MOMI, a bidirectional data pin, and for slave mode the MISO becomes SISO, a bidirectional data pin. In 3-wire mode, data is only transferred in one direction at a time.

The SPI bit clock is derived from the peripheral reference clock (typically 24 MHz with a maximum of 26 MHz). A prescaler divides the peripheral reference clock with a programmed divide ratio from 2 to 256. Typical bit clock range will be from 12 MHz to 93.75 kHz.

**NOTE**

Although only one dedicated SPI module is present on the MC1322x, the SSI module can also be programmed as a SPI functional block. See Chapter 16, “Synchronous Serial Interface (SSI)”

## 15.2 Features

The SPI module features include:

- Master or slave mode operation
- Transfer length programmable from 1 to 32 bits
- MSB-first shifting
- Data buffer is 4 bytes (32 bits) in length (not double buffered)
- Programmable transmit bit rate (typically 12 MHz max)
- Serial clock phase and polarity options
- Full-duplex (4-wire) or bidirectional data (3-wire) operation
- SPI transaction can be polled or interrupt driven
- Maskable interrupt request
- Slave select input/output
- Low Power (SPI Master uses gated clocks. Slave clock derived completely from SPI_SCK.)
- SPI Slave does not need system clock to be active

## 15.3 SPI Module Used for FLASH Interface (SPIF)

A second separate SPI FLASH Module is used for serial interface with the internal 128 Kbyte FLASH memory. Only the SPI module Master logic is used for the SPIF module with the following differences:

- The SPIF communicates through four signals. It does not support 3-wire operation.
- The SPIF MOSI output can not be tri-stated.

The SPIF Module is described in detail in Section 4.6.1, “FLASH Access”.

## 15.4 Block Diagrams

### 15.4.1 SPI System Block Diagram

This section shows a typical simple SPI system level 4-wire block diagram.

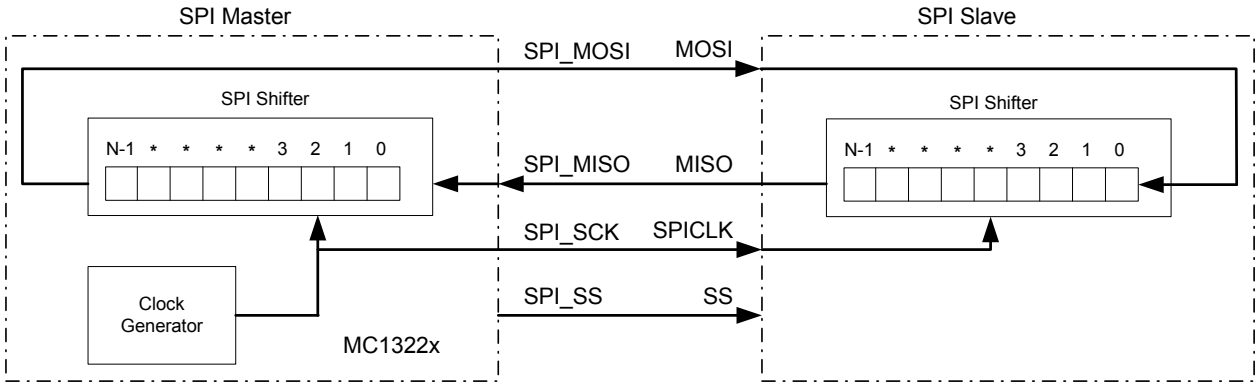



**Figure 15-1. SPI System Block Diagram**

Figure 15-1 shows the MC1322x as the SPI master and an external device as a SPI slave. The MCU (master) initiates all SPI transfers. During a transfer, the master shifts data out (on the MOSI pin) to the slave while simultaneously shifting data in (on the MISO pin) from the slave. The SPI interface supports simultaneous data exchange between master and slave, although some devices may be read or write only. The SPI_SCK signal is a clock output from the master and an input to the slave. The slave device must be enabled by a low level on the slave select input.

Most SPI devices are 8 bits in width for the SPI shifter. The MC1322x SPI module is programmable and supports up to a 32-bit width. Also, some SPI ports are capable of MSB-first or LSB-first serial shifting, however the SPI module supports only MSB-first shifting.

### 15.4.2 SPI Module Block Diagram

The SPI module is shown in Figure 15-2.

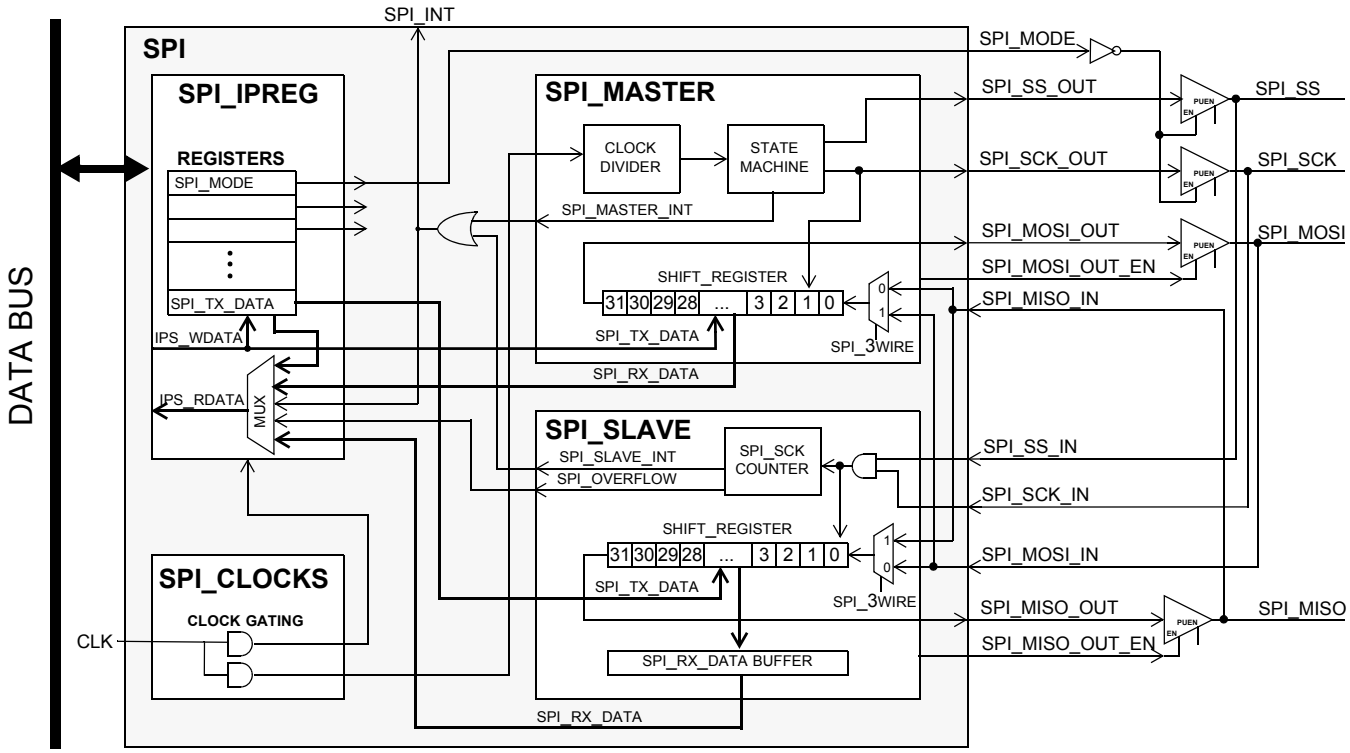



**Figure 15-2. Top Level SPI Block Diagram**

## 15.5 Functional Description

The SPI consists of control and status registers, clock generation, and data buffer registers. The master and slave blocks operate separately although they use the same clock. The following sections describe the function of the SPI.

### 15.5.1 Signal Descriptions

Table 15-1 lists SPI signal names and their descriptions.

**Table 15-1. SPI Signals**

| Signal Names | Direction | Active | Comments |
|---|---|---|---|
| SPI_SCK | Digital Input/Output | Programmable | SPI port clock. |
| SPI_MOSI | Digital Input/Output | High | SPI Port Master Out Slave In (MOSI) data signal. |

**Table 15-1. SPI Signals (continued)**

| Signal Names | Direction | Active | Comments |
|---|---|---|---|
| SPI_MISO | Digital Input/Output | High | SPI Port Master In Slave Out (MISO) data signal. |
| SPI_SS | Digital Input/Output | Low | SPI Port Slave Select (SS) signal. |

## 15.5.2 Clock Generation

Figure 15-2 shows the SPI block diagram. The SPI module generates the SPI_SCK only in master mode. The SPI bit clock is derived from the peripheral reference clock which is the same as the CPU clock (see Section 5.1.3.2, "Main System Clock Distribution"). The peripheral reference clock is the reference oscillator frequency divided by a 1-64 prescaler located in the CRM.

In the SPI module the peripheral reference clock is further divided by a second prescaler with a programmed divide ratio which is set by the field SPI_SCK_FREQ[2:0] in SPI Register SPI_SETUP. The 3-bit SPI_SCK_FREQ[2:0] is programmable from 0 to 7, where:

SPI_SCK = (peripheral reference clock) / (2 ^ (SPI_SCK_FREQ + 1))

Table 15-2 provides the SPI_SCK frequency versus SPI_SCK_FREQ[2:0] program value with an input reference frequency of 24 Mhz.

**Table 15-2. SPI_SCK Frequency versus SPI_SCK_FREQ[2:0]**

| SPI_SCK_FREQ[2:0] | Divide Ratio | SPI_SCK Frequency (reference frequency = 24 MHz) |
|---|---|---|
| 0 | 2 | 12 MHz |
| 1 | 4 | 6 MHz |
| 2 | 8 | 3 MHz |
| 3 | 16 | 1.5 MHz |
| 4 | 32 | 750 kHz |
| 5 | 64 | 375 kHz |
| 6 | 128 | 187.5 kHz |
| 7 | 256 | 93.75 kHz |

In slave mode, the SPI_SCK is determined by the SPI master (the other device).

## 15.5.3 Basic Operation

The SPI module is designed to operate either in master mode (SPI_MODE = 0) or slave mode (SPI_MODE = 1). In either mode data can be shifted in at the same time that data is being shifted out unless the device is configured as a 3-wire connection (SPI_3WIRE = 1). Programming of the SPI is done through a 32-bit IP Bus interface, which can also be accessed in half words and bytes. The following diagram shows the structure of the SPI module.

**NOTE**

Freescale recommends enabling the pad hysteresis for the SPI_SS and SPI SCK inputs in slave mode to avoid any glitches propagating into the SPI logic.

### 15.5.4 SPI Shift Clock Formats

To accommodate a wide variety of synchronous serial peripherals from different manufacturers, the SPI system has a clock (SPI_SCK) polarity bit, SPI_SCK_POL, and a clock phase control bit, SPI_SCK_PHASE, to select one of four clock formats for data transfers. SPI_SCK_POL selectively inserts an inverter in series with the clock. SPI_SCK_PHASE chooses between two different clock phase relationships between the clock and data.

The Figure 15-3 shows the clock formats when SPI_SCK_PHASE = 1. At the top of the figure, the bit times are shown for reference with bit 1 starting at the first SPI_SCK edge and the last bit ending one-half SPI_SCK cycle after the last SPI_SCK edge. The MSB first line shows the order of SPI data bits. Notice that "n" is the number of bits in a transfer (provided by SPI_DATA_LENGTH). Both variations of SPI_SCK polarity are shown, but only one of these waveforms applies for a specific transfer, depending on the value in SPI_SCK_POL. The SAMPLE IN waveform applies to the SPI_MOSI input of a slave or the SPI_MISO input of a master. The SPI_MOSI waveform applies to the SPI_MOSI output pin from a master and the SPI_MISO waveform applies to the SPI_MISO output from a slave. The SPI_SS OUT waveform applies to the slave select output from a master (provided SPI_SS_SETUP = 1). The master SPI_SS output goes to active low at least one SPI_SCK cycle before the start of the transfer and goes back high at least one SPI_SCK cycle after the end of the last bit time of the transfer (provided that SPI_SCK_COUNT is set equal to SPI_DATA_LENGTH + 1). The SPI_SS IN waveform applies to the slave select input of a slave.

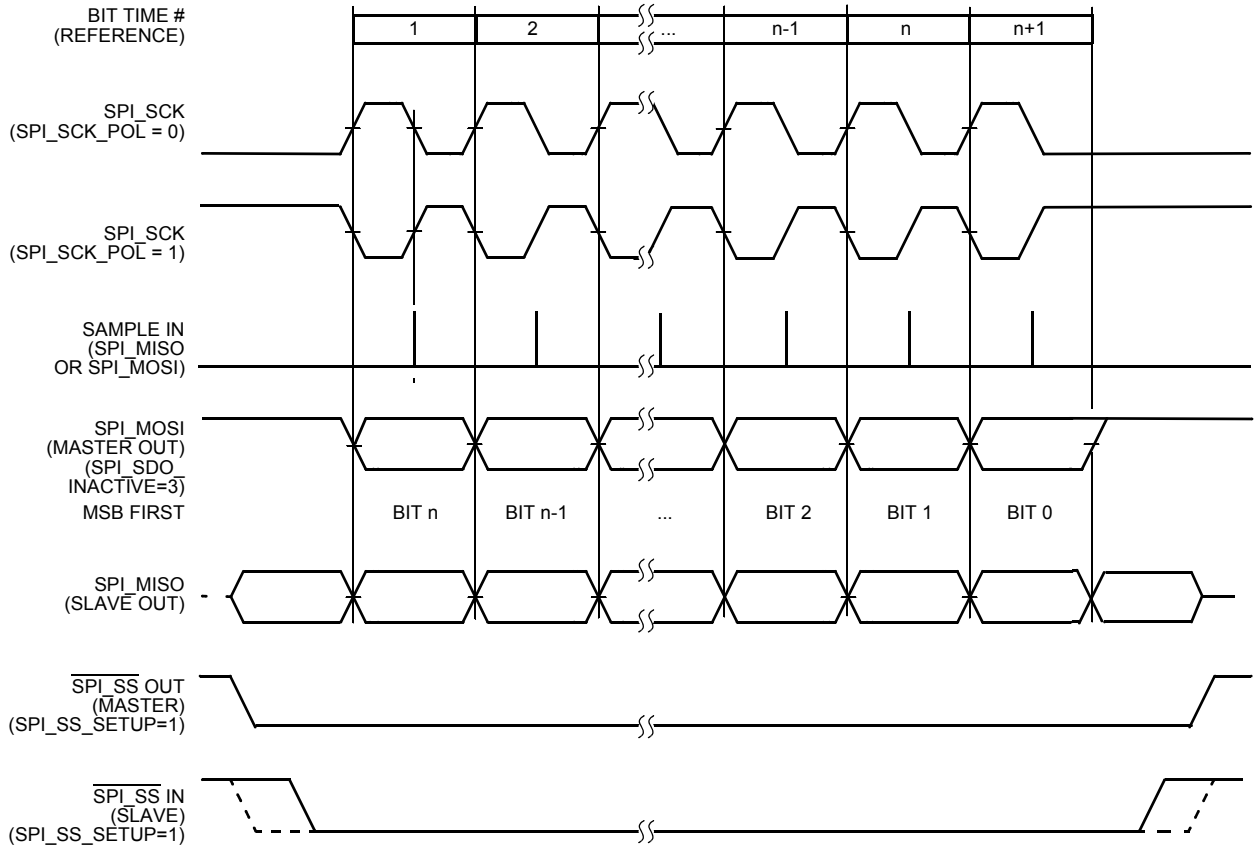



**Figure 15-3. SPI Clock Formats (SPI_SCK_PHASE = 1)**

When SPI_SCK_PHASE = 1, the slave begins to drive its SPI_MISO output when SPI_SS goes active, but the data is not defined until the first SPI_SCK edge. The first SPI_SCK edge shifts the first bit of data from the shifter onto the SPI_MOSI output of the master and the SPI_MISO output of the slave. The next SPI_SCK edge causes both the master and the slave to sample the data bit values on their SPI_MISO and SPI_MOSI inputs, respectively. At the third SPI_SCK edge, the SPI shifter shifts one bit position which shifts in the bit value that was just sampled, and shifts the second data bit value out the other end of the shifter to the SPI_MOSI and SPI_MISO outputs of the master and slave, respectively.

The Figure 15-4 shows the clock formats when SPI_SCK_PHASE = 0. At the top of the figure, the bit times are shown for reference with bit 1 starting as the slave is selected (SPI_SS IN goes low), and the last bit ends at the last SPI_SCK edge. The MSB first line shows the order of SPI data bits. Both variations of SPI_SCK polarity are shown, but only one of these waveforms applies for a specific transfer, depending on the value in SPI_SCK_POL. The SAMPLE IN waveform applies to the SPI_MOSI input of a slave or the SPI_MISO input of a master. The SPI_MOSI waveform applies to the SPI_MOSI output pin from a master and the SPI_MISO waveform applies to the SPI_MISO output from a slave. The SPI_SS OUT waveform applies to the slave select output from a master (provided SPI_SS_SETUP = 1). The master SPI_SS output goes active at the start of the first bit time of the transfer and goes inactive at least one SPI_SCK cycle after the end of the last bit time of the transfer (provide SPI_SCK_COUNT is set equal to SPI_DATA_LENGTH + 1). The SPI_SS IN waveform applies to the slave select input of a slave.

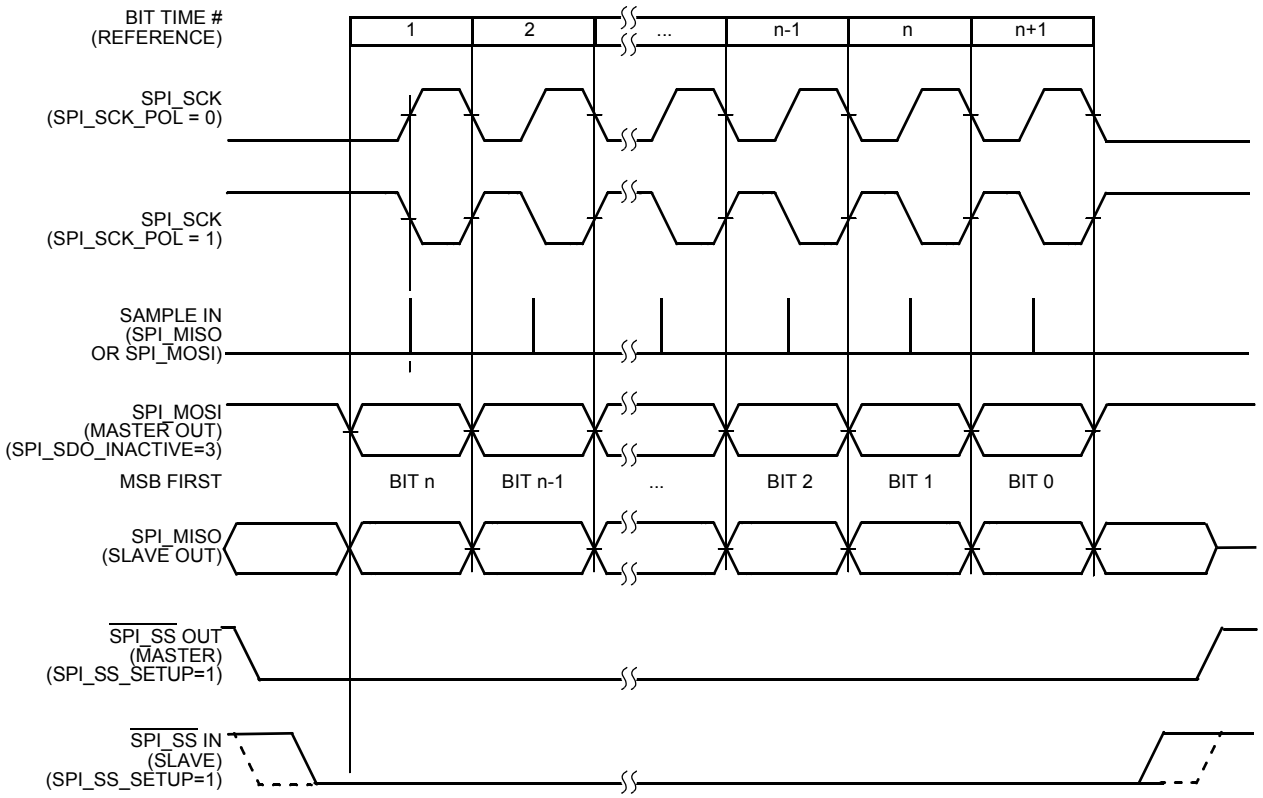



**Figure 15-4. SPI Clock Formats (SPI_SCK_PHASE = 0)**

When SPI_SCK_PHASE = 0, the slave begins to drive its SPI_MISO output with the first data bit value when SPI_SS goes active. The first SPI_SCK edge causes both the master and the slave to sample the data bit values on their SPI_MISO and SPI_MOSI inputs, respectively. At the second SPI_SCK edge, the SPI shifter shifts one bit position which shifts in the bit value that was just sampled and shifts the second data bit value out the other end of the shifter to the SPI_MOSI and SPI_MISO outputs of the master and slave, respectively.

## 15.5.5 SPI Interrupts

The SPI module has a single interrupt/status done bit, i.e., SPI_INT, Register SPI_STATUS. The SPI_INT is set when either the SPI is done sending data in master mode or is done receiving data in slave mode. There is no local mask bit for the SPI_INT. The interrupt for the module is enabled/disabled via the Interrupt Controller (ITC).

Completion of a master or slave transfer is controlled by the SPI_DATA_LENGTH[6:0] field of Register SPI_CLK_CTRL:

- In master mode, the SPI_DATA_LENGTH value controls for how many SPI_SCK periods the SPI shifts out data to SPI_MOSI.
  - The number of data shifts is equal to SPI_DATA_LENGTH.

  - The SPI_DATA_LENGTH field auto decrements, and when it reaches zero it changes SPI_MOSI to its inactive state (the inactive state is programmed in the SPI_SETUP Register) and SPI_INT is set high.
  - This register must be reprogrammed for each SPI transaction.
  - If SPI_DATA_LENGTH is greater than 32 (not allowed), then the data received from the external slave SPI device is shifted out after the 32nd shift.
- In slave mode, the SPI_DATA_LENGTH register defines the length in bits (or clocks) of the receive data.
  - The number of bits to receive is equal to SPI_DATA_LENGTH + 1.
  - The internal SPI_SCK count is automatically set to zeros when SPI_SS is deasserted or when SPI_DATA_LENGTH is reached.
  - Each time SPI_SCK count reaches SPI_DATA_LENGTH, the received data is saved in the SPI_RX_DATA buffer and the SPI_INT signal is set high.
  - If SPI_DATA_LENGTH is greater than 31(not allowed), only the last 32 bits received are saved in the SPI_RX_DATA buffers.

The SPI_INT status bit can be cleared by writing it to a “1”. It is also cleared with the start of a master SPI transfer which is caused by setting the SPI_START bit in the SPI_CLK_CTRL Register.

## 15.6 SPI Register Memory Map

The SPI module is programmed via a set of memory-mapped registers

- The base address is **0x8000_2000**.
- The register memory map for the module related to the base address is listed in Table 15-3 and shows the 32-bit registers of the SPI module.

The registers should be accessed only as 32 bits.

**Table 15-3. SPI Module Register Memory Map**

| Offset | [31:24] | [23:16] | [15:8] | [7:0] | Access Type | Access Width |
|---|---|---|---|---|---|---|
| + 0x00 | SPI_TX_DATA (SPI transmit data) | | | | R/W | 32 |
| + 0x04 | SPI_RX_DATA (SPI received data) | | | | R/W | 32 |
| + 0x08 | SPI_CLK_CTRL (SPI Status and Clock Control) | | | | R/W | 32 |
| + 0x0C | SPI_SETUP (SPI setup) | | | | R/W | 32 |
| + 0x10 | SPI_STATUS (SPI status) | | | | R/W | 32 |

# 15.7 SPI Registers and Control Bits

## 15.7.1 Transmit Data Register (SPI_TX_DATA)

Transmit Data Register (SPI_TX_DATA) is a 32-bit shift register which is used for transmitting data. During master mode operation the data are shifted left (MSB-first), where SPI_TX_DATA[31] is the first bit to come out on SPI_MOSI. During slave mode operation, SPI_TX_DATA[31] is the first bit to come out on SPI_MISO.

| SPI_TX_DATA | | | | SPI Transmit Data | | | | | | | | | Addr Base+0x00 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | SPI_TX_DATA[31:16] | | | | | | | | | | | | | | | |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | SPI_TX_DATA[15:0] | | | | | | | | | | | | | | | |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 15-4. SPI Transmit Data Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 0-31 | SPI transmit data — | Transmit shift register data. |

## 15.7.2 SPI Received Data Register (SPI_RX_DATA)

SPI Received Data Register (SPI_RX_DATA) is a 32-bit shift register which is used for receiving data. During master mode operation the data are shifted left (MSB-first), where SPI_RX_DATA[0] is the first bit to come in on SPI_MISO. During slave mode operation, SPI_TX_DATA[0] is the first bit to come in on SPI_MOSI.

| SPI_RX_DATA | | | | SPI Received Data | | | | | | | | | Addr Base+0x04 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | SPI_RX_DATA[31:16] | | | | | | | | | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | SPI_RX_DATA[15:0] | | | | | | | | | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 15-5. SPI Receive Data Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 0-31 | SPI receive data — | Receive shift register data. |

## 15.7.3 SPI Clock Control Register (SPI_CLK_CTRL)

| SPI_CLK_CTRL | Clock Control Register | | | | | | | | | | | | Addr Base+0x08 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | | | | | | | | | | | | | | | | |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | SPI_SCK_COUNT | | | | | | | | SPI_START | SPI_DATA_LENGTH | | | | | | |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw | w | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 15-6. SPI Clock Control Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 16-31 | Reserved | |
| 8-15 | **SPI_SCK_COUNT[7:0]** — In master mode this register determines how many SPI_SCK periods to generate in the following SPI transaction. | The number of SPI_SCK periods is equal to SPI_SCK_COUNT + 1. The SPI_SCK_COUNT register auto decrements during the SPI transaction and has to be set for each transaction. This register has no effect in slave mode. |
| 7 | **SPI_START** — In master mode this bit is a write-only bit used to initiate the SPI transaction. | By writing a '1' to SPI_START the SPI will exit its IDLE state which automatically turns on the gated clock in the SPI device and start the transaction. This bit has no effect in slave mode. |

**Table 15-6. SPI Clock Control Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 0-6 | **SPI_DATA_LENGTH[6:0] —** In master mode the SPI_DATA_LENGTH register controls for how many SPI_SCK periods the SPI shifts out data from the Shift Register to SPI_MOSI. The number of data shifts is equal to SPI_DATA_LENGTH. | The SPI_DATA_LENGTH register auto decrements and when it reaches 0 it changes SPI_MOSI to its inactive state. The exact behavior of the inactive state is programmed in the SPI_SETUP Register. This Register must be reprogrammed for each SPI transaction. Notice that if SPI_DATA_LENGTH is greater than 32, then the data received from the external slave SPI device is shifted out after the 32nd shift.<br>In slave mode the SPI_DATA_LENGTH register defines the length in bits of the receive data. The number of bits to receive is equal to SPI_DATA_LENGTH + 1. A internal SPI_SCK count is automatically set to zeros when SPI_SS is deasserted or when SPI_DATA_LENGTH is reached. Each time SPI_SCK count reaches SPI_DATA_LENGTH, the received data is saved in the SPI_RX_DATA buffer and the SPI_INT signal is set high. Notice that if SPI_DATA_LENGTH is greater than 31, only the last 32 bits received are saved in the SPI_RX_DATA buffer. |

## 15.7.4 SPI Setup Register (SPI_SETUP)

| SPI_SETUP | | | | SPI Setup Register | | | | | | | | | Addr Base+0x0C | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | | | | | | | | | | | | | | | SPI_3WIRE | SPI_MODE |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | | SPI_SCK_FREQ | | | | SPI_MISO_PHASE | SPI_SCK_PHASE | SPI_SCK_POL | | | SPI_SDO_INACTIVE_ST | | SPI_SS_DELAY | | SPI_SS_SETUP | |
| TYPE | r | rw | rw | rw | r | rw | rw | rw | r | r | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 15-7. SPI Setup Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 18-31 | Reserved | |
| 17 | **SPI_3WIRE** — The SPI_3WIRE bit chooses between a SPI 3-wire or a SPI 4-wire connection. | When SPI_3WIRE is set high, the SPI serial data output and input will share the same port. When SPI_3WIRE is set low, the SPI serial data output and input have independent ports, SPI_MISO and SPI_MOSI. When the SPI 3-wire connection is selected, in master mode the SPI uses the MOSI (MOMI) pin for bidirectional SPI transfers and in slave mode the SPI uses the MISO (SISO) pin for bidirectional SPI transfers. The enable or disable of the MISO or MOSI output drivers are controlled by the SPI_SDO_INACTIVE_ST bits. |
| 16 | **SPI_MODE** — This bit determines if the SPI is a master device or slave device. | When SPI_MODE is low, the SPI is in master mode. When SPI_MODE is set high, the SPI is in slave mode. |
| 15 | Reserved | |

**Table 15-7. SPI Setup Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 12-14 | **SPI_SCK_FREQ[2:0]** — In master mode the SPI_SCK_FREQ register controls the SPI_SCK frequency:<br>SPI_SCK = CLK / (2 ^ (SPI_SCK_FREQ + 1))<br>This register has no effect in slave mode. | |
| 11 | Reserved | |
| 10 | **SPI_MISO_PHASE** — In master mode the SPI_MISO_PHASE determines which edge of SPI_SCK is used to sample the SPI_MISO input signal. This allows the communication to SPI devices that latches data on one clock edge and clocks out data either on the same edge or on the opposite edge. | When SPI_MISO_PHASE is low SPI_MISO is latched with the opposite SPI_SCK edge as it clocks out SPI_MOSI. This is accomplished by pre-latching into a register on the preceding clock edge before being clocked into the Shift Register. When SPI_MISO_PHASE is high SPI_MISO is latched with the same SPI_SCK edge as it clocks out SPI_MOSI. The figure in the next section illustrates this. (Note: This figure is only a illustration and does not show the real implementation). The SPI_MISO_PHASE register has no effect in slave mode. |
| 9 | **SPI_SCK_PHASE** — The SPI_SCK_PHASE register determines the phase of SPI_SCK with respect to SPI_MISO and SPI_MOSI. | |
| 8 | **PI_SCK_POL** — The SPI_SCK_POL register is used to control the SPI_SCK polarity. | |
| 6-7 | Reserved | |
| 4-5 | **SPI_SDO_INACTIVE_ST** — In master mode SPI_SDO_INACTIVE_ST controls the state of the SPI_MOSI_OUT and SPI_MOSI_OUT_EN output signals when the SPI device is not shifting data out of the Shift Register (inactive state).<br>If SPI_DATA_LENGTH is set to zeros, the SPI Master is always in its inactive state. The SPI_3WIRE bit has no effect to this behavior in master mode. While the SPI Master is active, SPI_MOSI_OUT is connected to the serial data output and SPI_MOSI_OUT_EN is set high.<br><br>In slave mode SPI_SDO_INACTIVE_ST controls the state of the SPI_MISO_OUT and SPI_MISO_OUT_EN output signals when SPI_SS is deasserted or SPI_3WIRE has been set high. While SPI_SS is asserted and SPI_3WIRE is reset low, SPI_MISO_OUT is connected to the serial data output and SPI_MISO_OUT_EN is set high. | |

**Table 15-7. SPI Setup Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 2-3 | **SPI_SS_DELAY** — In master mode and SPI_SS set in Auto Mode (SPI_SS_SETUP = 0 or 1), SPI_SS_DELAY determines when the SPI_SS is asserted and deasserted. SPI_SS is asserted SPI_SS_DELAY + 1 SPI_SCK periods before the first shift and deasserted SPI_SS_DELAY + 1 SPI_SCK period after the last shift. This register has no effect in slave mode. | |
| 0-1 | **SPI_SS_SETUP** — In master mode the SPI_SS_SETUP determines the exact behavior of the SPI_SS_OUT signal and in slave mode it determines the behavior of the SPI_SS_IN signal.<br>In master mode if a SPI_SS is setup to be automatic, it will automatically select the SPI device prior to the first SPI_SCK edge and automatically de-select after the last SPI_SCK edge. | |

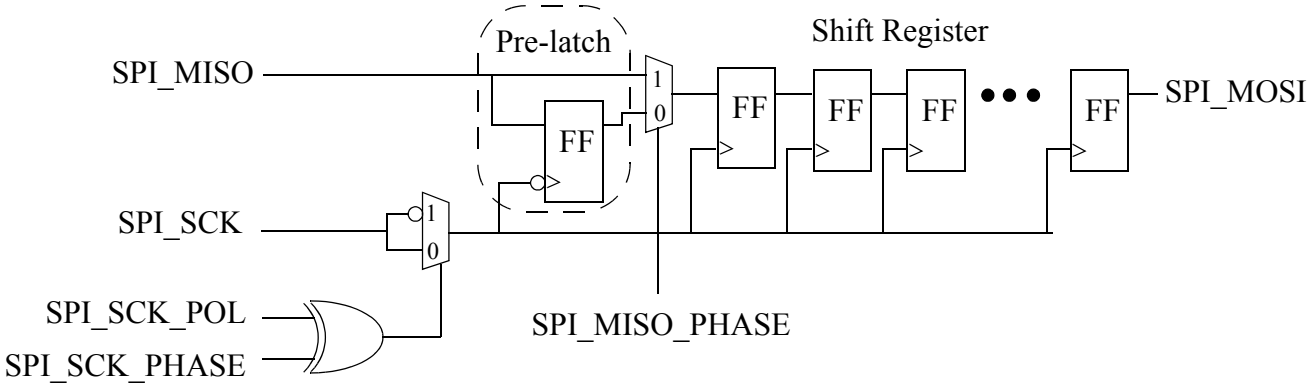



**Figure 15-5. SPI Master MISO Sampling**

**Table 15-8. SPI_MOSI Inactive State (Master Mode)**

| SPI_MODE | SPI_SDO_IN ACTIVE_ST[1:0] | SPI_MOSI_OUT | SPI_MOSI_OUT_EN | PAD |
|---|---|---|---|---|
| 0 | 00 | 0 | 0 | 'z' |
| 0 | 01 | 1 | 0 | 'z' |
| 0 | 10 | 0 | 1 | 0 |
| 0 | 11 | 1 | 1 | 1 |
| 1 | XX | 0 | 0 | 'z' |

**Table 15-9. SPI_MISO Inactive State (Slave Mode)**

| SPI_MODE | SPI_SS | SPI_3WIRE | SPI_SDO_IN ACTIVE_ST[1:0] | SPI_MISO_OUT | SPI_MISO_ OUT_EN | PAD |
|---|---|---|---|---|---|---|
| 1 | deasserted | X | 00 | 0 | 0 | 'z' |
| 1 | deasserted | X | 01 | 1 | 0 | 'z' |
| 1 | deasserted | X | 10 | 0 | 1 | 0 |
| 1 | deasserted | X | 11 | 1 | 1 | 1 |
| 1 | asserted | 0 | XX | SDO | 1 | SDO |
| 1 | asserted | 1 | 0X | SDO | 0 | 'z' |
| 1 | asserted | 1 | 1X | SDO | 1 | SDO |
| 0 | X | X | XX | 0 | 0 | 'z' |

**Table 15-10. SPI_SS Behavior**

| SPI_SS_SETUP | SPI_SS_OUT (Master Mode) | SPI_SS_IN (Slave Mode) |
|---|---|---|
| 00 | Auto/Active High | Active High |
| 01 | Auto/Active Low | Active Low |
| 10 | Low | Active High |
| 11 | High | Active Low |

## 15.7.5 SPI Status Register (SPI_STATUS)

| SPI_STATUS | | | | SPI Status Register | | | | | | | | | Addr Base+0x10 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | | | | | | | | | | | | | | | | |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | | | | | | | | SPI_FIRST_DATA | | | | SPI_OVERFLOW | | | | SPI_INT |
| TYPE | rw | rw | rw | rw | rw | rw | rw | r | w | rw | rw | r,w1c | rw | rw | rw | r,w1c |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 15-11. SPI Status Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 9-31 | Reserved | |
| 8 | **SPI_FIRST_DATA** — This bit is a read-only status bit generated by the slave SPI device. | This bit is set high when the SPI has received data for the first transfer after SPI_SS went active. For all remaining transfers, while SPI_SS is kept active, SPI_FIRST_DATA is reset to zero. This bit is not used in master mode. |
| 5-7 | Reserved | |
| 4 | **SPI_OVERFLOW** — This bit is a read-only and write-one-to-clear SPI overflow status bit generated by the slave SPI device. | This bit is set when the SPI has received data in slave mode and the SPI interrupt from the previous received data has not been cleared. The SPI_OVERFLOW bit can be cleared by writing a “1” to it. This bit is not used in master mode. |
| 1-3 | Reserved | |
| 0 | **SPI_INT** — This bit (also referred as SPI done) is a read-only and write-one-to-clear interrupt bit generated by the master or slave SPI. | This bit is set high either when the SPI is done sending data in master mode or the SPI is done receiving data in slave mode. This bit can be cleared by writing a “1” to it. This bit is also cleared with the start of a master SPI transfer which is caused by setting the SPI_START bit in the SPI_CLK_CTRL register. |

## 15.8 Timing Information

The SPI timing information is shown in the figure below. The SPI Master logic is clocked with the MCU clock which highest frequency is 26 Mhz. Thus the highest SPI Master SPI_SCK frequency is 13 Mhz (1/2 MCU clock). The SPI Slave logic is clocked by the incoming SPI_SCK and asynchronously reset by the deassertion of SPI_SS. The timing diagram is only applicable to the SPI I/O when they are in their input mode. Notice that the setup and hold times is with respect to the capturing SPI_SCK edge which could be the rising or falling edge and not just the rising edge as shown in this diagram. See the two SPI Clock Formats diagrams for more information on the capturing SPI_SCK edge. Also keep in mind that the setup and hold time of SPI_SS is with respect to its assertion which could be either active low or active high.

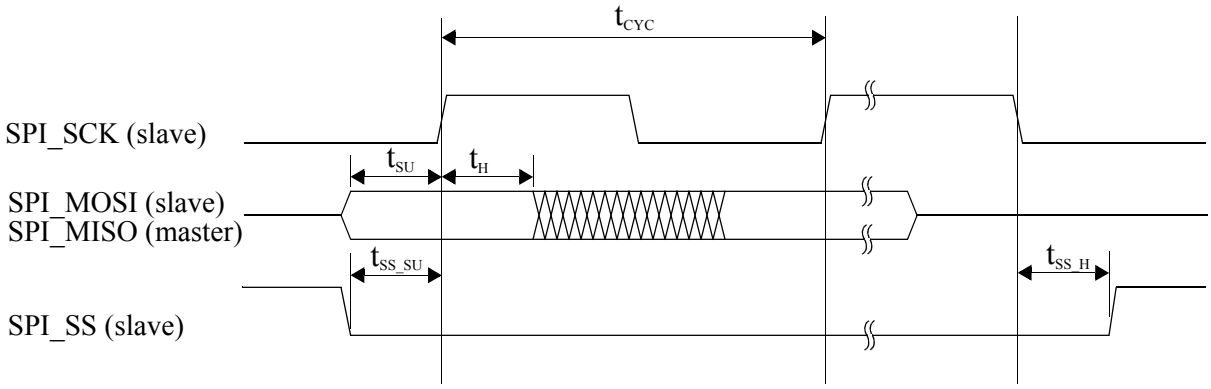



**Figure 15-6. SPI Timing Diagram**

**Table 15-12. SPI Timing Limits**

| PARAMETER | MNEMONIC | SPECIFICATION | | | UNITS |
|---|---|---|---|---|---|
| | | MIN | TYP | MAX | |
| Slave SPI_SCK Period | $t_{CYC}$ | | 38 | | nsec |
| Slave SPI_MOSI & Master SPI_MISO1 Setup Time | $t_{SU}$ | | 10 | | nsec |
| Slave SPI_MOSI & Master SPI_MISO1 Hold Time | $t_H$ | | 10 | | nsec |
| Slave SPI_SS Setup Time | $t_{SS\_SU}$ | | 10 | | nsec |
| Slave SPI_SS Hold Time | $t_{SS\_H}$ | | 10 | | nsec |

# Chapter 16
# Synchronous Serial Interface (SSI)

## 16.1 Overview

The SSI is a full-duplex, serial port that allows the MC1322x to communicate with a variety of serial devices. These serial devices can be standard CODer-DECoder (CODECs), Digital Signal Processors (DSPs), microprocessors, peripherals, and popular industry audio CODECs that implement the inter-IC sound bus standard ($I^2S$).

SSI is typically used to transfer samples in a periodic manner. The SSI consists of independent transmitter and receiver sections with independent clock generation and frame synchronization.

**NOTE**

- Although this chapter provides reference information on the SSI module, the user/programmer is directed toward the software utility entitled the "Synchronous Serial Interface (SSI) Driver", see Section Appendix B, "MC1322x Software Driver Utilities"
- A common use of the SSI is to provide a second SPI port for the MC1322x (in addition to the standard SPI port, Section Chapter 15, "Serial Peripheral Interface Module (SPI)")

### 16.1.1 Features

The SSI includes the following features:

- Synchronous transmit and receive sections with shared internal/external clocks and frame syncs, operating in Master or Slave mode.
- Normal mode operation using frame sync
- Network mode operation allowing multiple devices to share the port with as many as thirty-two time slots
- Gated Clock mode operation requiring no frame sync
- SSI clock source is Peripheral Clock (typically 24 MHz); maximum SSI transfer rate is 6.0 MHz
- Separate Transmit and Receive FIFOs. Each of which is 8x24 bits
- Programmable data interface modes including $I^2S$, LSB, MSB aligned
- Programmable word length (8, 10, 12, 16, 18, 20, 22 or 24 bits)
- Program options for frame sync and clock generation
- Programmable $I^2S$ modes (Master, Slave)
- Programmable internal clock divider

- Time Slot Mask Registers for reduced CPU overhead (for Tx and Rx both)
- SSI power-down feature

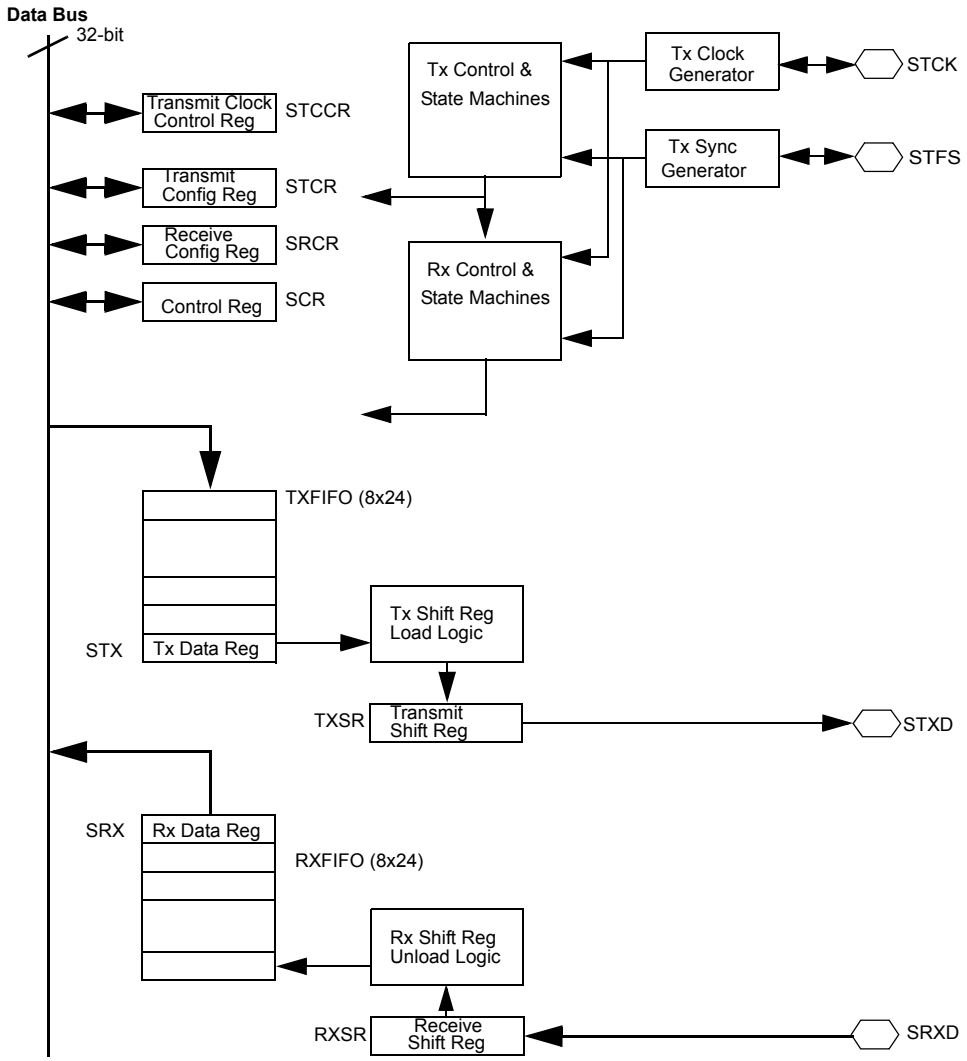



**Figure 16-1. SSI Block Diagram**

## 16.2 Block Diagram

See Figure 16-1 shows the SSI block diagram. The SSI module is a high speed synchronous serial port capable of full duplex operation. The SSI is operated by a block of memory-mapped control, status, and data registers. Independent transmitter and receiver sections with clock generation and frame synchronization can be used while in master or slave mode. Serial transmit and receive data are independently buffered by TX and RX FIFOS, each as an 8x24 configuration.

The SSI is typically used to transfer samples in a repetitive, periodic manner (framed data), although single transfer operation is possible, as would be used in a SPI emulation.

## 16.3 External Signal Description

The SSI uses four external signal ports and Table 16-1 lists their properties. As with other module signals, the SSI signals share functionality with GPIO.

**NOTE**

The use of the pins for SSI must be enabled in the GPIO module through the Function Select fields.

The following sections describe the signals.

**Table 16-1. SSI External Signal Properties**

| External Pin Name | SSI Module Signal | I/O Type | Function |
|---|---|---|---|
| GPIO1 / SSI_RX | SRXD | Input | Serial Receive Data |
| GPIO3 / SSI_BITCK | STCK | Input/Output | Serial Transmit / Receive Clock (Bit Clock) |
| GPIO2 / SSI_FSYN | STFS | Input/Output | Serial Transmit / Receive Frame Sync |
| GPIO0 / SSI_TX | STXD | Output | Serial Transmit Data |

### 16.3.1 SRXD — Serial Receive Data

The SRXD port is the serial data input into the Receive Data Shift Register.

### 16.3.2 STCK — Serial Transmit Clock (Bit Clock)

The STCK port can be either an receive input clock or the transmit output clock

- The clock signal is used by the transmitter and the receiver and can be either continuous or gated.
- During Gated Clock mode, data on the STCK port is valid only during the transmission of data, otherwise it is pulled to the inactive state.

### 16.3.3 STFS — Serial Transmit Frame Sync

The STFS port can be used as either an input or an output.

- The frame sync is used by the transmitter receiver to synchronize the transfer of data.
- The frame sync signal can be one bit or one word in length and can occur one bit before the transfer of data or right at the transfer of data.
- In Gated Clock mode, frame sync signals are not used. If STFS is configured as input, the external device should drive STFS during rising edge of STCK or SRCK.

### 16.3.4 STXD — Serial Transmit Data

The STXD port is the serial data output from the Serial Transmit Shift Register. The STXD port is an output port when data is being transmitted and is disabled between data word transmissions and on the trailing edge of the bit clock after the last bit of a word is transmitted.

### 16.3.5 Synchronous SSI Signal Configurations

Figure 16-2 shows the main SSI configurations. These ports support all transmit and receive functions with continuous or gated clock as shown.

**NOTE**

Gated clock implementations do not require the use of the frame sync port STFS.

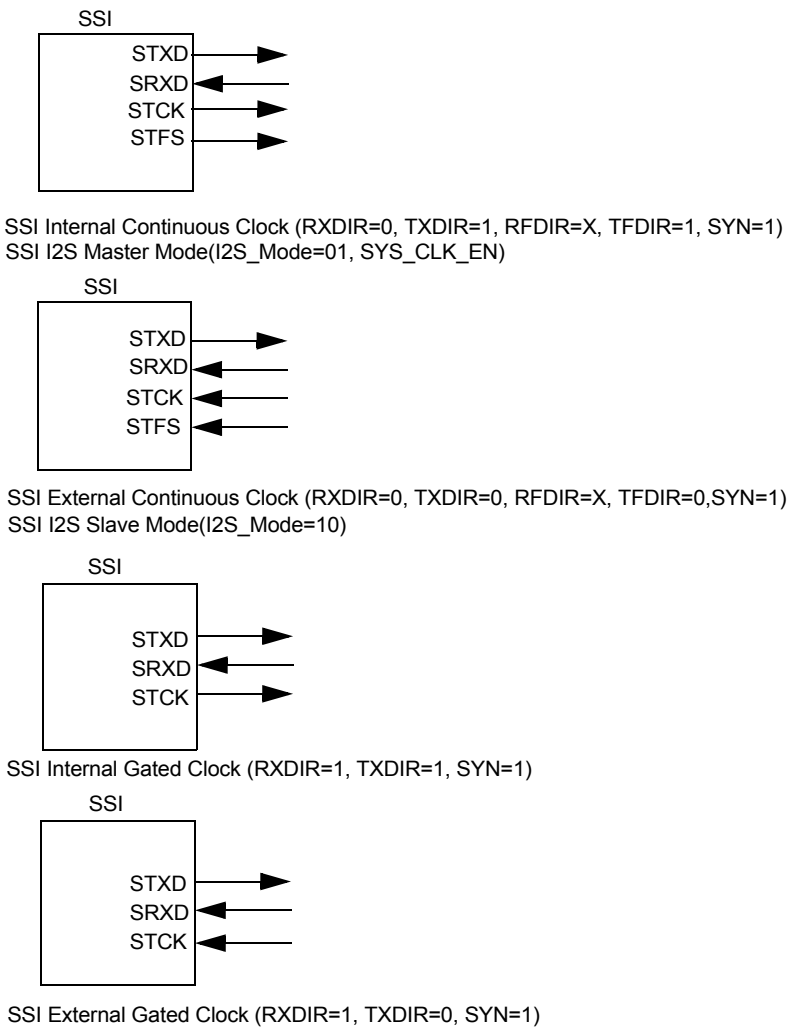



**Figure 16-2. Synchronous SSI Configurations—Continuous and Gated Clock**

See Figure 16-3 for an example of the port signals for an 8-bit data transfer. Continuous and gated clock signals are shown, as well as the bit-length frame sync signal and the word-length frame sync signal.

**NOTE**

The shift direction can be defined as MSB first or LSB first and that there are other options on the clock and frame sync.

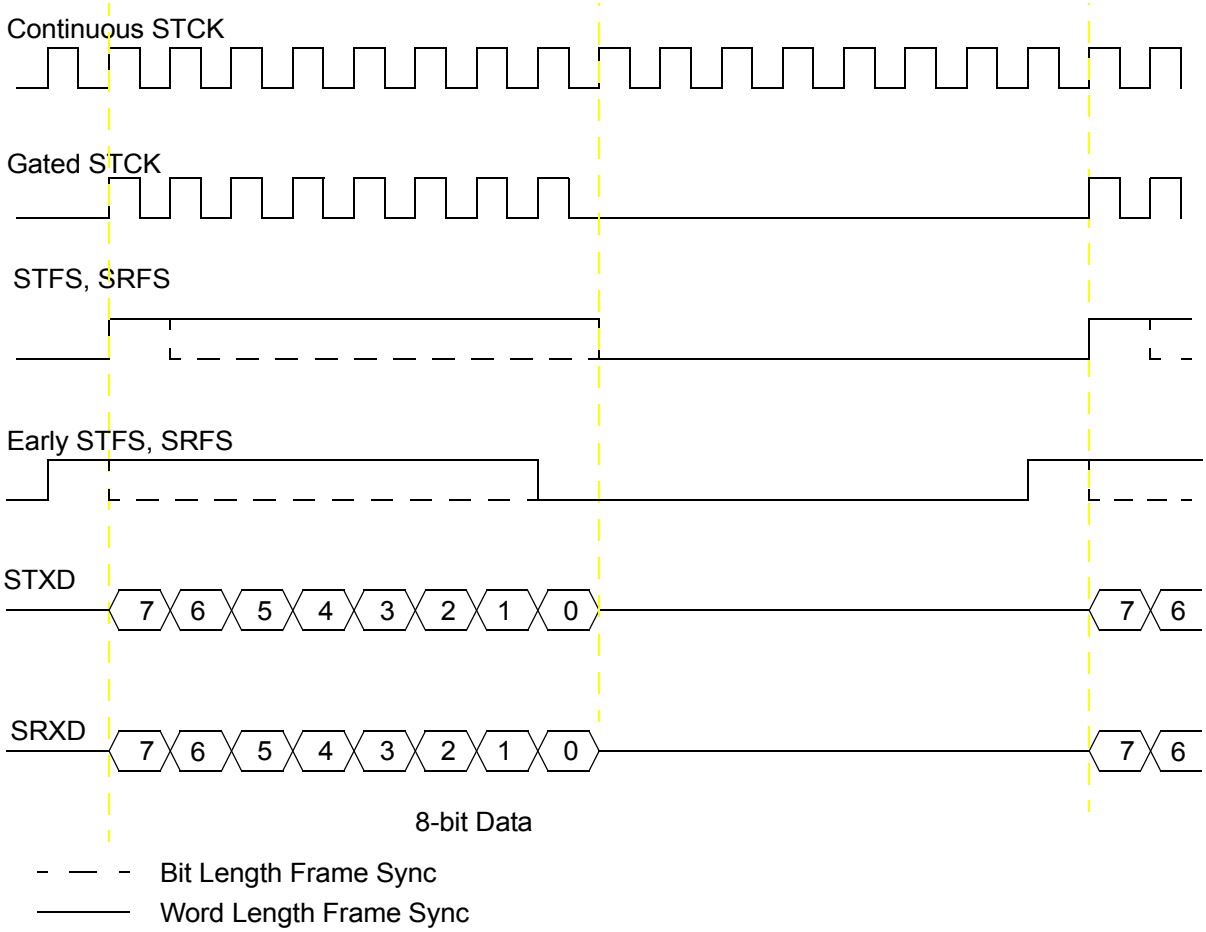



**Figure 16-3. Serial Clock and Frame Sync Timing**

Table 16-2 lists clock pin configurations.

**Table 16-2. Clock Pin Configurations**

| SYN | RXDIR | TXDIR | RFDIR | TFDIR | STCK | STFS |
|---|---|---|---|---|---|---|
| Synchronous Mode | | | | | | |
| 1 | 0 | 0 | x | 0 | CK in | FS in |
| 1 | 0 | 0 | x | 1 | CK in | FS out |
| 1 | 0 | 1 | x | 0 | CK out | FS in |
| 1 | 0 | 1 | x | 1 | CK out | FS out |
| 1 | 1 | 0 | x | x | Gated in | - |
| 1 | 1 | 1 | x | x | Gated out | - |

## 16.4 Functional Description

The following sections describe operation of the SSI module.

### 16.4.1 SSI Clock Generation

The SSI either provides or uses the following clocks:

- Bit clock (STCK) — Used to clock the serial data in and out of the SSI port. This clock is either generated internally (from Peripheral Clock) or taken from external clock source (through the STCK clock port).
- Word clock — Used to count the number of data bits per word (8, 10, 12, 16, 18, 20, 22 or 24 bits). This clock is generated internally from the bit clock.
- Frame clock / Frame Sync (STFS) — Used to count the number of words in a frame. This signal can be generated internally from the bit clock, or taken from an external source (from the Tx frame sync port STFS).
- System clock — In master mode, this is an integer multiple of frame clock. This is the Peripheral Clock. It is used in cases when SSI has to provide the clock.

**NOTE**

The SSI system clock (Peripheral Clock) must always be equal to or greater than 4 times the bit clock. As a result, with the Peripheral Clock at 24 MHz max (typically with a 24 MHz reference frequency) and the available prescale options, the maximum bit clock (serial transfer rate) is 6.0 MHz.

In Normal mode (SCR[6:5]=00), the bit clock, used to serially clock the data, is visible on the Serial Transmit Clock (STCK) port. The word clock is an internal clock used to determine when transmission of an 8, 10, 12, 16, 18, 20, 22 or 24 bit word has completed. The word clock, in turn, then clocks the frame clock, which counts the number of words in the frame. The frame clock can be viewed on the STFS frame sync port, because a frame sync is generated after the correct number of words in the frame have passed. The relationship between the clocks and the dividers is shown in Figure 16-4. The bit clock can be received from an SSI clock port or can be generated from the Peripheral Clock through a divider, as shown in Figure 16-5.

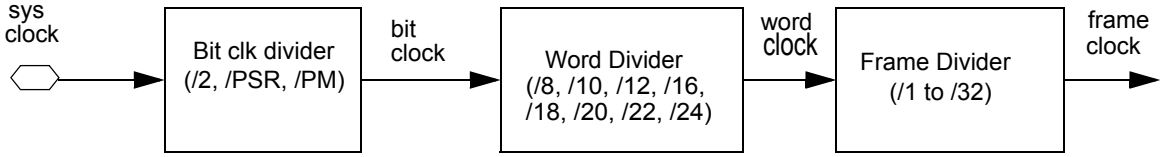



**Figure 16-4. SSI Clocking**

#### 16.4.1.1 SSI Clock and Frame Sync Generation

Data clock and frame sync signals are generated internally for master mode, or taken from the external interface in slave mode. If internally generated, the SSI clock generator is used to derive bit clock and frame sync signals from the Peripheral Clock clock. Figure 16-5 shows a block diagram of the clock generator for the transmit section.

The SSI clock generator consists of a programmable prescaler/divider chain for bit rate clock generation. A programmable frame rate divider and a word length divider are used for frame rate sync signal generation.

**NOTE**

In Gated Clock mode, the data clock is valid only when data is being transmitted. Otherwise the clock output port is pulled to the inactive state.

The serial bit clock can be internal or external, depending on the Transmit Direction (TXDIR) bit in the SSI Transmit Configuration Register (STCR).

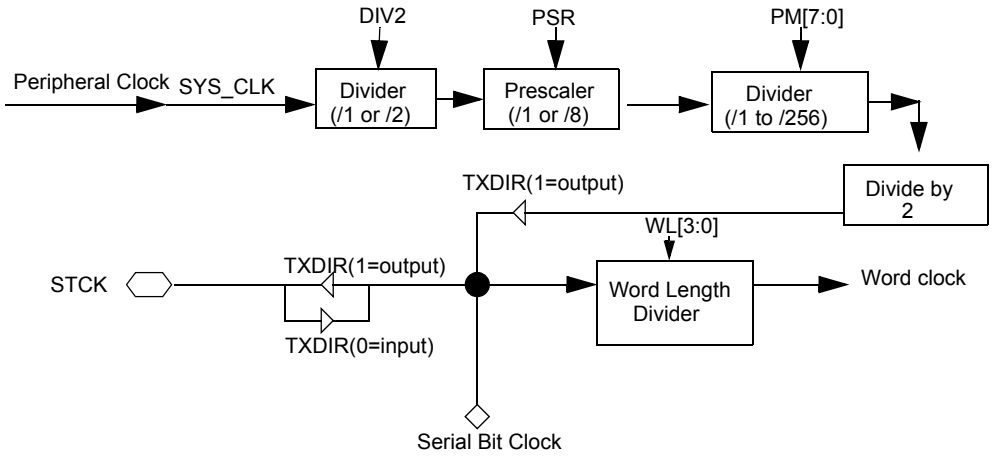



**Figure 16-5. SSI Transmit Clock Generator Block Diagram**

See Figure 16-6 shows the Frame Sync Generator block for the transmit section. When internally generated, both receive and transmit frame sync are generated from the word clock and are defined by the Frame Rate Divider (DC[4:0]) bits and the Word Length (WL[3:0]) bits of the SSI Transmit Clock Control Register (STCCR). The receive section contains an equivalent circuit for the Frame Sync Generator.

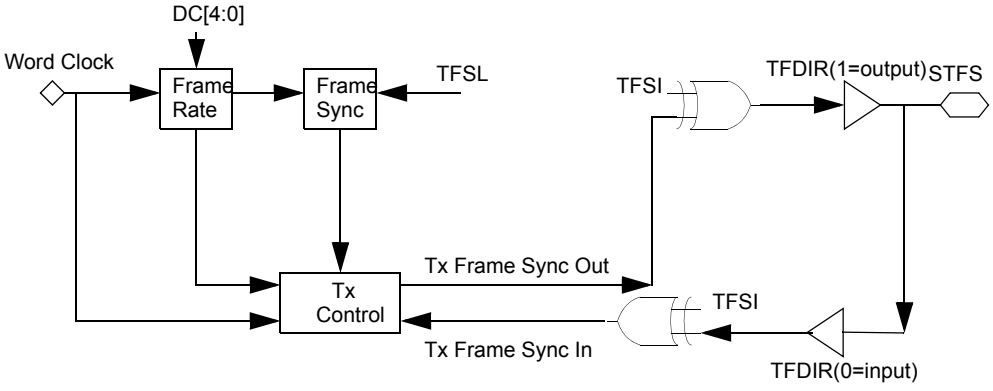



**Figure 16-6. SSI Transmit Frame Sync Generator Block Diagram**

### 16.4.1.2 Register SSI_STCCR DIV2, PSR and PM[7:0] Field Descriptions

The bit clock prescaler/divider chain consists of the following serial blocks:

1. /1 or /2 Divider (controlled by Register SSI_STCCR[DIV2])
2. /1 or /2 Prescaler (controlled by register SSI_STCCR[PSR])
3. /1 to /256 Modulus Divider (controlled by register SSI_STCCR[PM[7:0]])
4. Fixed /2 Divider

The bit clock frequency is generated from the SSI system Clock (Peripheral Clock) and can be calculated using the following equation:

$$f_{bit\_clock} = f_{peripheral\_clk} / ( (DIV2[val] + 1) \times ((7 \times PSR[val]) + 1) \times (PM[val] + 1) \times 2)$$

**NOTE**

- The SSI system clock (Peripheral Clock) must always be equal to or greater than 4 times the bit clock; this equates to a 6.0 MHz max bit clock
- The resulting divide ratio will always be an even number because of the final /2 Divider

As an example, the I2S standard requires a 2.5 Mhz bit clock with an accuracy of +/- 10%. The bit clock derived from 24 MHz will be based on an integer divide and the closest frequency is 2.4 MHz which requires an integer divide of 10. The following program values will render the 2.4 MHz bit clock:

$f_{peripheral\_clk} = 24$ MHz

DIV2[val] = 0

PSR[val] = 0

$PSM[7:0] = 4_{dec}$

$f_{I2S\_bit\_clock} = 24 \text{ MHz} / ( (0 + 1) \times ((7 \times 0) + 1) \times (4 + 1) \times 2) = 24 \text{ MHz} / (1 \times 1 \times 5 \times 2) = 2.4 \text{ MHz}$

### 16.4.1.3 Register SSI_STCCR WL[3:0] and DC[4:0] Field Descriptions

The word clock is derived from the bit clock as determined by the WL[3:0] field:

$$f_{word\_clock} = f_{bit\_clk} / ( (WL[3:0] \times 2) + 2)$$

Where for a word length example of 24:

$WL[3:0] = 11_{dec}$

$f_{word\_clock} = f_{bit\_clk} / ( (11 \times 2) + 2) = f_{bit\_clk} / 24$

In turn, the frame sync (word transfer rate) is derived from the word clock as determined by the DC[4:0] field:

$$f_{frame\_sync} = f_{word\_clk} / ( DC[4:0] + 1)$$

or related back to bit clock:

$f_{frame\_sync} = f_{bit\_clk} / [( (WL[3:0] \times 2) + 2) \times ( DC[4:0] + 1)]$

Where for an example of 16 words per frame:

$DC[4:0] = 15_{dec}$

$f_{frame\_sync} = f_{word\_clk} / ( 15 + 1) = f_{word\_clk} / 16$

## 16.4.2 Transmit Data Path

Figure 16-7 illustrates the SSI transmit data path. The transmit data is buffered (if enabled through the TX Configuration Register) by an independent 8x24 transmit FIFO (TXFIFO), i.e., 32-bit words are not supported by the SSI. The data are written to the TXFIFO by write accesses to the Transmit Data Register (SSI_STX); if the TXFIFO is not enabled, the STX register is the only buffer register. Movement of data to the transmit shift register (TSXR) is controlled by the TSXR load logic.

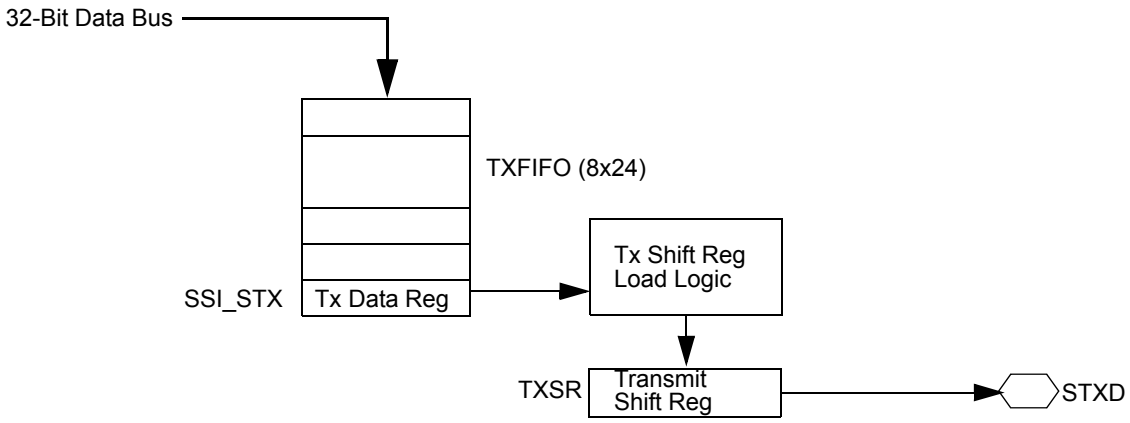



**Figure 16-7. SSI Transmit Data Path**

### 16.4.2.1 SSI Transmit FIFO

The SSI Transmit FIFO register is an 8x24-bit register set. These registers are not directly accessible, i.e., the Transmit Data Register (STX) writes data to the FIFO registers (see Section 16.6.9).

- Transmitted data is first-in-first-out.
- The TX FIFO is overflow protected.
- An interrupt request can be generated whenever the data level in the TXFIFO falls below a programmed threshold.

### 16.4.2.2 SSI Transmit Shift Register (TXSR)

The SSI Transmit Shift Register (TXSR) is a 24-bit shift register that contains the data being transmitted.

- This register is not directly accessible by the user.
- When a continuous clock is used, data is shifted out to the Serial Transmit Data (STXD) port by the selected (internal/external) bit clock when the associated (internal/external) frame sync is asserted.

- When a gated clock is used, data is shifted out to the STXD port by the selected (internal/external) gated clock.
- The Word Length control bits (WL[3:0]) in the STCCR determine the number of bits to be shifted out of the TXSR before it is considered empty and can be written to again. This word length can be 8, 10, 12, 16, 18, 20, 22 or 24 bits.
- The data to be transmitted occupies the most significant portion of the shift register if TXBIT0 is '0', otherwise it occupies the least significant portion. The unused portion of the register is ignored.
- Data is always shifted out of this register with the Most Significant Bit (MSB) first when the SHFD bit of the STCR is cleared. If this bit is set, the Least Significant Bit (LSB) is shifted out first.

Figure 16-8 through Figure 16-11 show the transmitter loading and shifting operation. The figures show the working for some WL values, the same can be extended for other values.

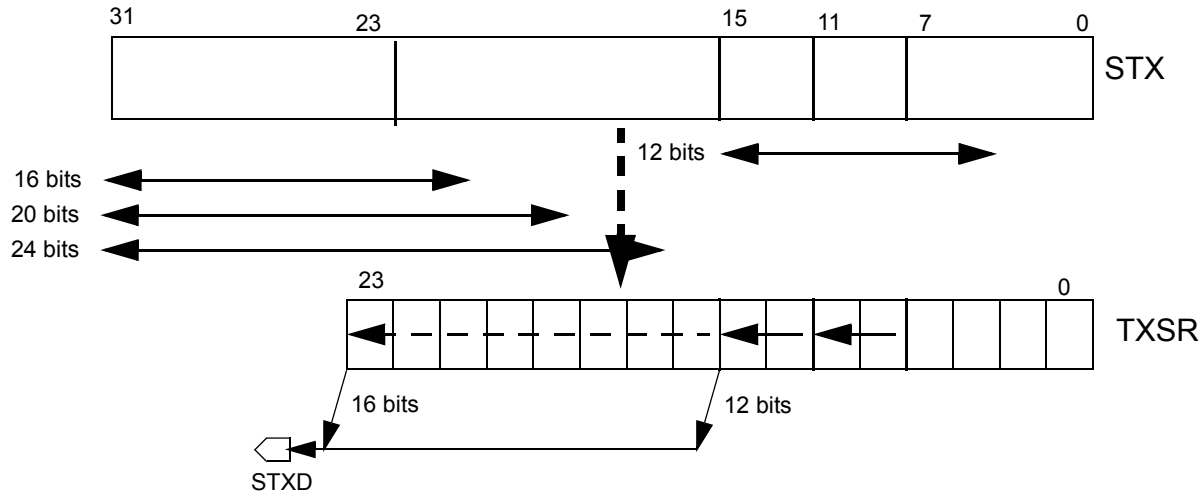



**Figure 16-8. Transmit Data Path (TXBIT0=0, TSHFD=0) (MSB Alignment)**

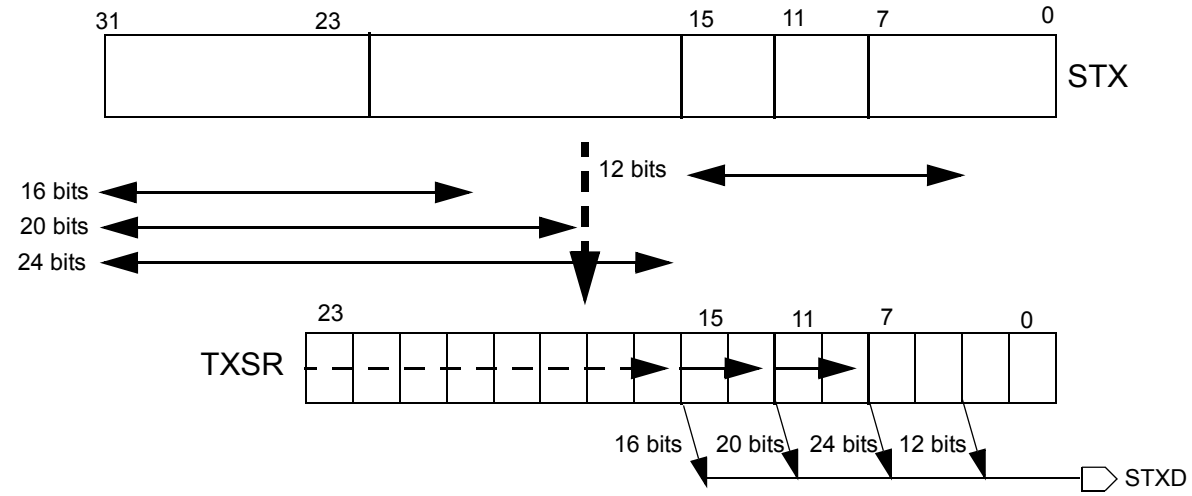



**Figure 16-9. Transmit Data Path (TXBIT0=0, TSHFD=1) (MSB Alignment)**

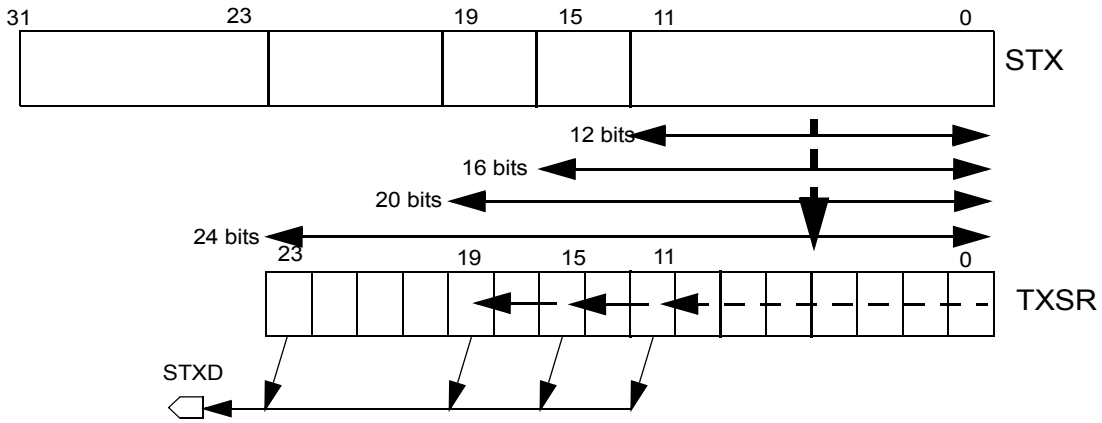



**Figure 16-10. Transmit Data Path (TXBIT0=1, TSHFD=0) (LSB Alignment)**

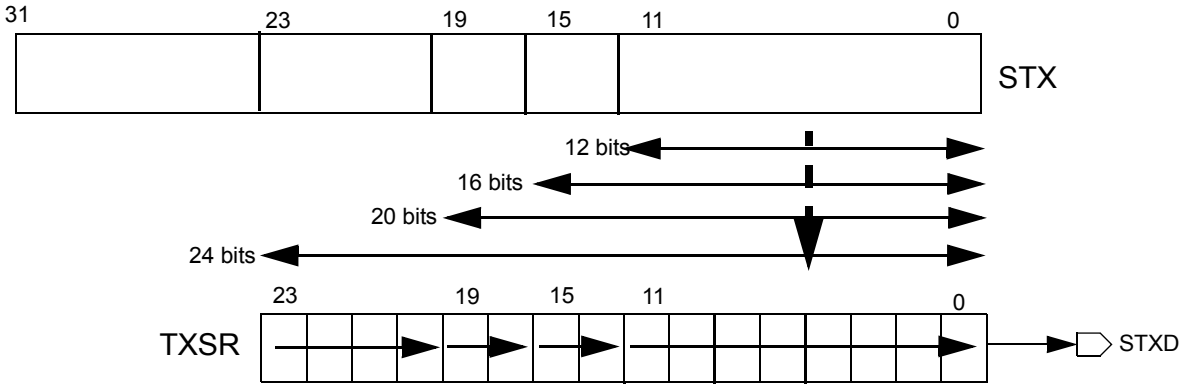



**Figure 16-11. Transmit Data Path (TXBIT0=1, TSHFD=1) (LSB Alignment)**

## 16.4.3 Receive Data Path

Figure 16-12 illustrates the SSI receive data path. The receive data is buffered (if enabled through the RX Configuration Register) by an independent 8x24 receive FIFO (RXFIFO), i.e., 32-bit words are not supported by the SSI. The data are unloaded from the RXFIFO by read accesses to the Receive Data Register (SSI_SRX); if the RXFIFO is not enabled, the SRX register is the only buffer register. Movement of data from the receive shift register (RSXR) is controlled by the RSXR unload logic.

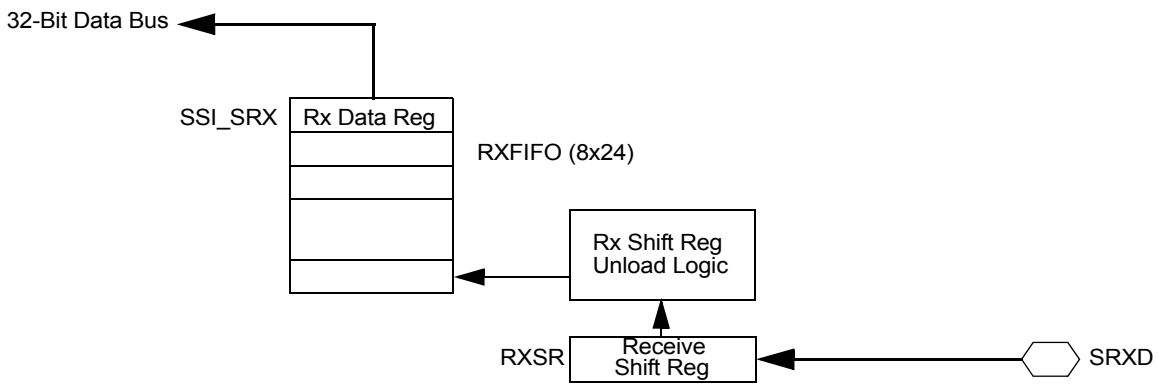



**Figure 16-12. SSI Receive Data Path**

### 16.4.3.1 SSI Receive FIFO Register

The SSI Receive FIFO Register is a 8x24-bit register set. These registers are not directly accessible, i.e., the Receive Shift Register (RXSR) accepts data from the RXFIFO (see Section 16.6.2).

- Received data is first-in-first-out.
- The RX FIFO is overflow protected.
- An interrupt request can be generated whenever when the data level in the RXFIFO rises to a programmed threshold.

### 16.4.3.2 SSI Receive Shift Register (RXSR)

The SSI Receive Shift Register (RXSR) is a 24-bit, shift register that receives incoming data from the serial receive data SRXD port.

- This register is not directly accessible by the end user.
- When a continuous clock is used, data is shifted in by the selected (internal/external) bit clock when the associated (internal/external) frame sync is asserted.
- When a gated clock is used, data is shifted in by the selected (internal/external) gated clock.
- Data is assumed to be received MSB first if the SHFD bit of the SRCR is cleared. If this bit is set, the data is received LSB first.
- Data is transferred to the RXFIFO after 8, 10, 12, 16, 18, 20, 22 or 24 bits have been shifted in depending on the WL[3:0] control bits. For receiving less than 24 bits of data, LSB bits are appended with zero.

Figure 16-13 through Figure 16-16 show the receiver loading and shifting operation. The figures show the working for some WL values, the same can be extended for other values.

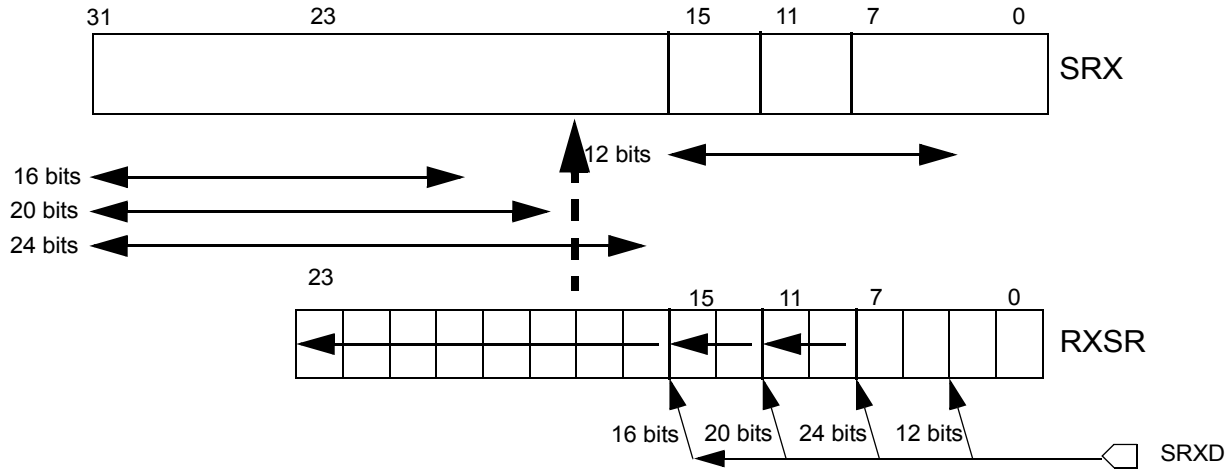



**Figure 16-13. Receive Data Path (RXBIT0=0, RSHFD=0) (MSB Alignment)**

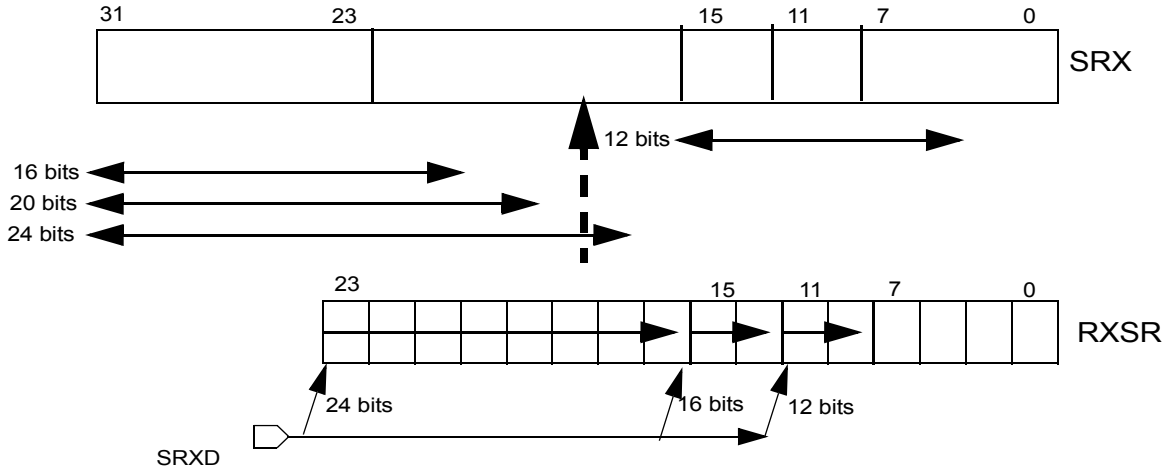



**Figure 16-14. Receive Data Path (RXBIT0=0, RSHFD=1) (MSB Alignment)**

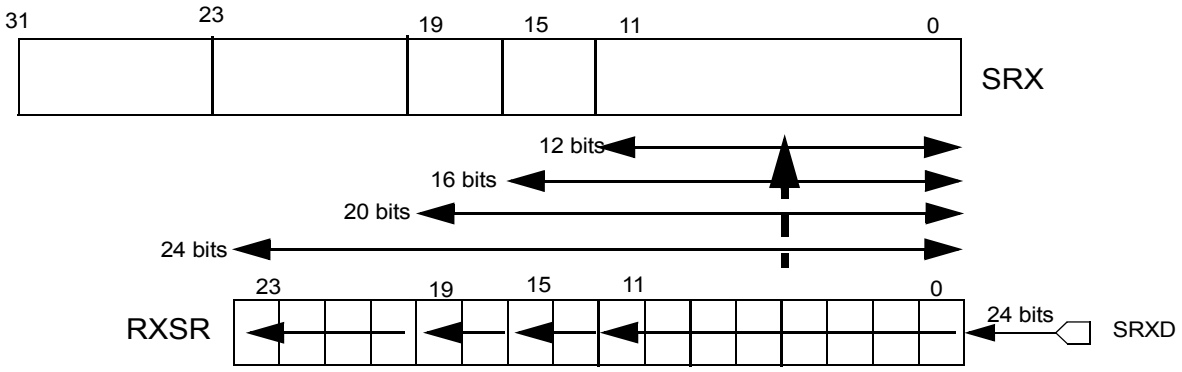



**Figure 16-15. Receive Data Path (RXBIT0=1, RSHFD=0) (LSB Alignment)**

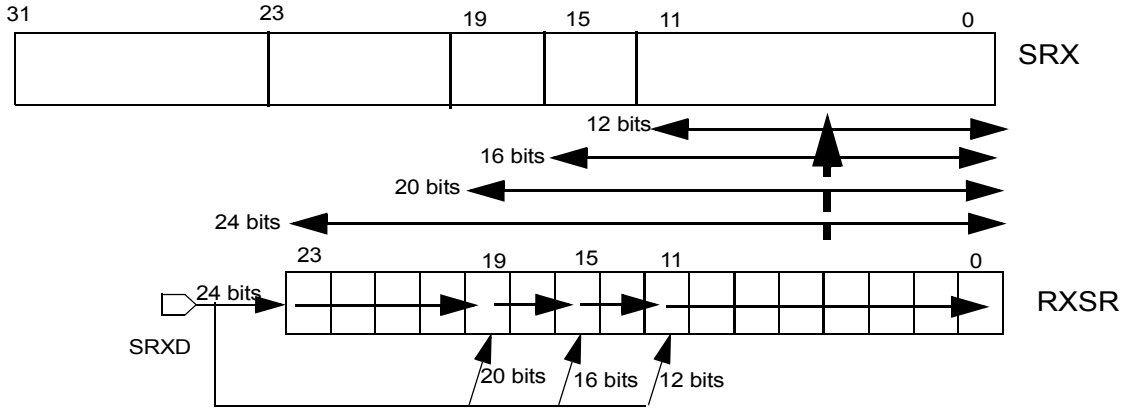



**Figure 16-16. Receive Data Path (RXBIT0=1, RSHFD=1) (LSB Alignment)**

### 16.4.4 Modes of Operation

SSI has the following basic operating modes (synchronous only).

- Normal mode
  - — Synchronous protocol only
- Network mode
  - — Synchronous protocol only
- Gated Clock mode
  - — Synchronous protocol only

These modes can be programmed by several bits in the SSI control registers. See Table 16-3 for the list of SSI operating modes and some of the typical applications in which they can be used:

**Table 16-3. SSI Operating Modes**

| TX, RX Sections | Serial Clock | Mode | Typical Application |
|---|---|---|---|
| Synchronous | Continuous | Normal | Multiple synchronous CODECs |
| Synchronous | Continuous | Network | TDM CODEC or DSP network |
| Synchronous | Gated | Normal | SPI-type devices; DSP to MCU |

The transmit and receive sections of the SSI operate in synchronous only. In Synchronous mode, the transmitter and the receiver use a common clock and frame synchronization signal. The RXBIT0 and RSHFD bits in SRCR affect shifting-in of received data in synchronous mode. Continuous or Gated Clock mode can be selected. In Continuous mode, the clock runs continuously. In Gated Clock mode, the clock is only functioning during transmission.

Normal or Network mode can also be selected. In Normal mode, the SSI functions with one data word per frame. In Network mode, any number from two to thirty-two data words of I/O per frame can be used. Network mode is typically used in star time division multiplex networks with other processors or CODECs, allowing interface to time division multiplexed networks without additional logic. Use of the gated clock is not allowed in Network mode. These distinctions result in the basic operating modes that allow the SSI to communicate with a wide variety of devices.

The SSI supports both Normal and Network modes, and these can be selected independently for the transmitter and receiver. Typically these protocols are used in a periodic manner, where data is transferred at regular intervals, such as at the sampling rate of an external CODEC. Both modes use the concept of a frame. The beginning of the frame is marked with a frame sync when programmed with continuous clock. The frame sync occurs at a periodic interval. The length of the frame is determined by the DC[4:0] bits in the STCCR register. The number of words transferred per frame depends on the mode of the SSI.

In Normal mode, one data word is transferred per frame. In Network mode, the frame is divided into anywhere between two and thirty-two time slots, where in each time slot one data word can optionally be transferred.

Apart from the above basic modes of operation, SSI also supports I2S mode. In (non-I2S) slave modes (external frame sync), the programmed word length setting of the SSI should be equal to the word length

setting of the master. In I2S slave mode, the programmed word length setting of the SSI can be lesser than or equal to the word length setting of the I2S master (external CODEC).

In slave modes, the programmed frame length setting (DC bits) of the SSI can be lesser than or equal to the frame length setting of the master (external CODEC).

The following sections provide detailed descriptions of the above modes.

#### 16.4.4.1 Normal Mode

Normal mode is the simplest mode of the SSI. It is used to transfer data in one time slot per frame. A time slot is a unit of data and the WL[3:0] bits define the number of bits in a time slot. In Continuous Clock mode, a frame sync occurs at the beginning of each frame. The length of the frame is determined by the following factors:

- The period of the Serial Bit Clock (DIV2, PSR, PM[7:0] bits for internal clock or the frequency of the external clock on the STCK port)
- The number of bits per time slot (WL[3:0] bits)
- The number of time slots per frame (DC[4:0] bits)

If Normal mode is configured with more than one time slot per frame, data is transferred only in the first time slot. No data is transferred in subsequent time slots. In Normal mode, DC[4:0] values corresponding to more than a single time slot in a frame, only result in lengthening the frame. Data transfer only takes place during the first time slot of the frame.

##### 16.4.4.1.1 Normal Mode Transmit

The conditions for data transmission from the SSI in Normal mode are:

- SSI enabled (SSIEN = 1)
- Enable FIFO and configure Transmit and Receive Watermark if FIFO is used.
- Write data to Transmit Data Register (STX)
- Transmitter enabled (TE = 1)
- Frame sync active (for continuous clock case)
- Bit clock begins (for gated clock case)

When the above conditions occur in Normal mode, the next data word is transferred into the Transmit Shift Register (TXSR) from the Transmit Data Register (STX), or from the Transmit FIFO Register, if the transmit FIFO is enabled. The new data word is transmitted immediately.

If the transmit FIFO is not enabled and the transmit data register empty (TDE) bit is set, transmit interrupt occurs if the transmit interrupt enable (TIE) and TDE_EN bits are set.

The Transmit FIFO Empty (TFE) bit is set if the Transmit FIFO reaches the selected threshold. If the transmit FIFO is enabled and the Transmit FIFO Empty (TFE) bit is set, transmit interrupt occurs if the transmit interrupt enable (TIE) and TFE_EN bits are set. If the transmit FIFO is enabled and filled with data, 8 data words can be transferred before more data must be written to the STX register.

The STXD port is disabled except during the data transmission period. For a continuous clock, the optional frame sync output and clock outputs are not disabled, even if both receiver and transmitter are disabled.

#### 16.4.4.1.2 Normal Mode Receive

The conditions for data reception from the SSI are:

- SSI enabled (SSIEN = 1)
- Receiver enabled (RE = 1)
- Frame sync active (for continuous clock case)
- Bit clock begins (for gated clock case)

With the above conditions in Normal mode with a continuous clock, each time the frame sync signal is generated (or detected) a data word is clocked in. With the above conditions and a gated clock, each time the clock begins, a data word is clocked in.

If the receive FIFO is not enabled, the received data word is transferred from the Receive Shift Register (RXSR) to the Receive Data Register (SRX), the Receive Data Ready (RDR) flag is set. Receive Interrupt occurs if RIE and RDR_EN bits are set.

If the receive FIFO is enabled, the received data word is transferred to the Receive FIFO. The Receive FIFO Full (RFF) flag is set if the Receive Data Register (SRX) is full and Receive FIFO reaches the selected threshold. Receive Interrupt occurs if Receive Interrupt Enable (RIE) and RFF_EN bits are set.

Software has to read the data from the Receive Data Register (SRX) before a new data word is transferred from the Receive Shift Register (RXSR), otherwise the Receive Overrun Error (ROE) bit is set. If receive FIFO is enabled, the Receive Overrun Error (ROE) bit is set when the Receive FIFO data level reaches the selected threshold and a new data word is ready to be transferred to the Receive FIFO.

See Figure 16-17 for illustration of transmitter and receiver timing for an 8-bit word in the first time slot in Normal mode, continuous clock with a late word length frame sync. The Tx Data register is loaded with the data to be transmitted. On arrival of the clock, this data is transferred to the Transmit Shift Register which gets transmitted on arrival of the frame-sync on the STXD output. Simultaneously, the Receive Shift Register shifts in the received data available on the SRXD input and at the end of the time slot, this data is transferred to the Rx Data Register.

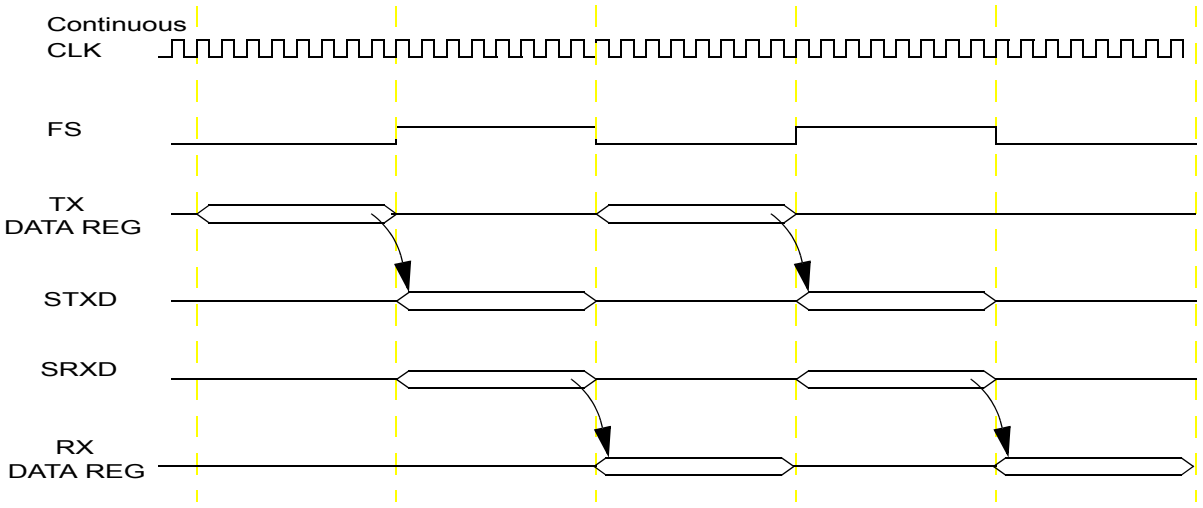



**Figure 16-17. Normal Mode Timing - Continuous Clock**

Figure 16-18 shows a similar case for internal (SSI generates clock) gated clock mode and Figure 16-19 shows a case for external (SSI receives clock) gated clock mode.

**NOTE**

A pull-down resistor is required in the gated clock case because the clock port is disabled between transmissions.

The Tx Data register is loaded with the data to be transmitted. On arrival of the clock, this data is transferred to the Transmit Shift Register which gets transmitted on arrival of the frame-sync on the STXD output. Simultaneously, the Receive Shift Register shifts in the received data available on the SRXD input and at the end of the time slot, this data is transferred to the Rx Data Register. In case of Internal Gated clock mode, the Tx Data line and clock output port are put in the high-impedance state at the end of transmission of the last bit (at the completion of the complete clock cycle), whereas, in External Gated clock mode, the Tx Data line is tri-stated at the last inactive edge of the incoming bit clock (during the last bit in a data word).

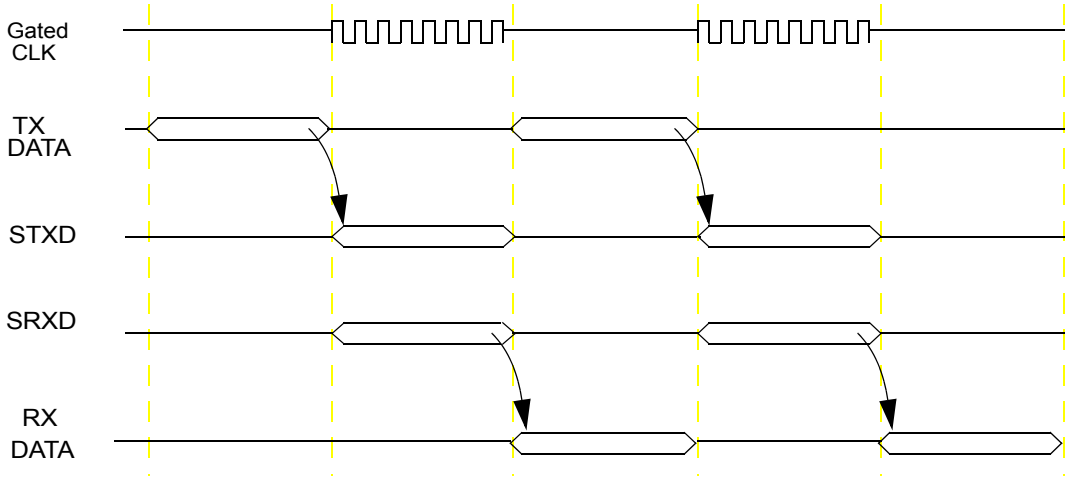



**Figure 16-18. Normal Mode Timing - Internal Gated Clock**

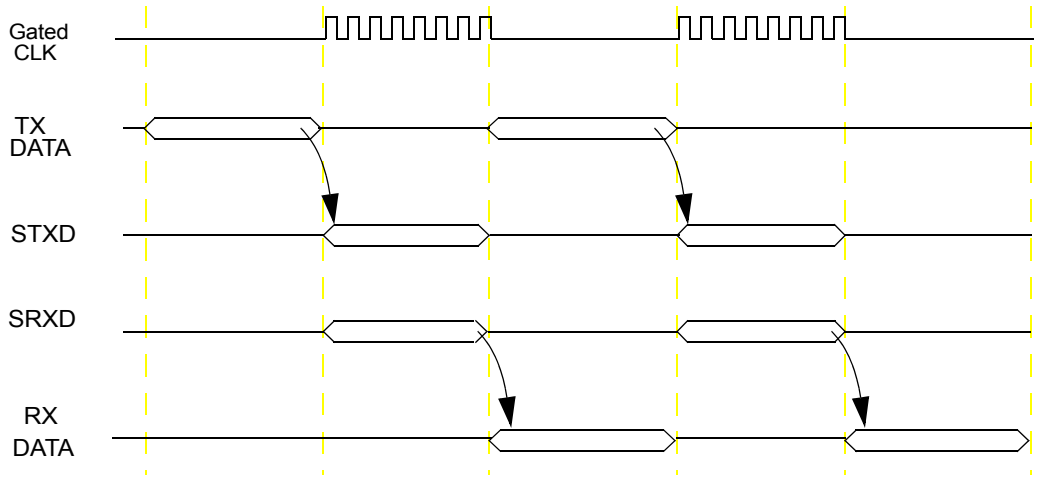



**Figure 16-19. Normal Mode Timing - External Gated Clock**

### 16.4.4.2 Network Mode

Network mode is used for creating a Time Division Multiplexed (TDM) network, such as a TDM CODEC network or a network of DSPs. In Continuous Clock mode, a frame sync occurs at the beginning of each frame. In this mode, the frame is divided into more than one time slot. During each time slot, one data word can be transferred. Each time slot is then assigned to an appropriate CODEC or DSP on the network. The DSP can be a master device that controls its own private network, or a slave device that is connected to an existing TDM network and occupies a few time slots.

The frame sync signal indicates the beginning of a new data frame. Each data frame is divided into time slots and transmission and/or reception of one data word can occur in each time slot (rather than in just the frame sync time slot as in Normal mode). The frame rate dividers, controlled by the DC[4:0] bits, select two to thirty-two time slots per frame. The length of the frame is determined by the following factors:

- The period of the serial bit clock (PSR, PM[7:0] bits for internal clock, or the frequency of the external clock on the STCK port)
- The number of bits per sample (WL[3:0] bits)
- The number of time slots per frame (DC[4:0] bits)

In Network mode, data can be transmitted in any time slot. The distinction of the Network mode is that each time slot is identified with respect to the frame sync (data word time). This time slot identification allows the option of transmitting data during the time slot by writing to the STX registers or ignoring the time slot as determined by STMSK register bits. The receiver is treated in the same manner and received data is only transferred to the receive data register/fifo if the corresponding time slot is enabled (through SRMSK).

By utilizing the STMSK and SRMSK registers, software only has to service the SSI during valid time slots. This eliminates any overhead associated with unused time slots. Refer to Section 16.6.10 and Section 16.6.11 for more information on STMSK and SRMSK.

#### 16.4.4.2.1 Network Mode Transmit

The transmit portion of SSI is enabled when the SSIEN and the TE bits in the SCR are both set. However, for continuous clock, when the TE bit is set, the transmitter is enabled only after detection of a new frame sync (transmission starts from the next frame boundary).

Normal start-up sequence for transmission is to perform the following:

1. Write the data to be transmitted to the STX register. This clears the TDE flag.
2. Set the TE bit to enable the transmitter on the next word boundary (for continuous clock case).
3. Enable transmit interrupts.

Alternatively, the programmer may decide not to transmit in a time slot by configuring the STMSK. The TDE flag is not cleared, but the STXD port remains disabled during the time slot. When the frame sync is detected or generated (continuous clock), the first enabled data word is transferred from the STX register to the TXSR and is shifted out (transmitted). When the STX register is empty, the TDE bit is set, which causes a transmitter interrupt (in case the FIFO is disabled) to be sent if the TIE bit is set. Software can poll the TDE bit or use interrupts to reload the STX register with new data for the next time slot. Failing to reload the STX register before the TXSR is finished shifting (empty) causes a transmitter underrun and the TUE error bit is set. In case the FIFO is enabled, the TFE flag is set in accordance with the watermark setting and this flag causes the transmitter interrupt to occur.

The operation of clearing the TE bit disables the transmitter after completion of transmission of the current frame. Setting the TE bit enables transmission from the next frame. During that time the STXD port is disabled. The TE bit should be cleared after the TDE bit is set to ensure that all pending data is transmitted.

To summarize, the Network mode transmitter generates interrupts every enabled time slot and requires the core program to respond to each enabled time slot. These responses from the core are one of the following:

- Write data in data register to enable transmission in the next time slot.
- Configure the time slot register to disable transmission in the next time slot (unless time slot is already masked by STMSK register bit).
- Do nothing—transmit underrun occurs at the beginning of the next time slot and the previous data is re-transmitted.

#### 16.4.4.2.2 Network Mode Receive

The receiver portion of the SSI is enabled when both the SSIEN and the RE bits in the SCR are set. However, the receive enable only takes place during that time slot if RE is enabled before the second to last bit of the word. If the RE bit is cleared, the receiver is disabled at the end of the current frame. The SSI is capable of finding the start of the next frame automatically. When the word is completely received, it is transferred to the SRX register, which sets the RDR bit (Receive Data Ready). Setting the RDR bit causes a receive interrupt to occur if the receiver interrupt is enabled (the RIE bit is set). The second data word (second time slot in the frame), begins shifting in immediately after the transfer of the first data word to the SRX register. The core program has to read the data from the Receive Data Register (which clears RDR) before the second data word is completely received (ready to transfer to RX data register) or a receive overrun error occurs (the ROE bit is set).

An interrupt can occur after the reception of each enabled data word or the programmer can poll the RDR flag. The core program response can be one of the following:

- Read RX and use the data.
- Read RX and ignore the data.
- Do nothing—the receiver overrun exception occurs at the end of the current time slot.

For a continuous clock, the optional frame sync output and clock output signals are not affected, even if the transmitter or receiver is disabled. TE and RE do not disable the bit clock or the frame sync generation. To disable the bit clock and the frame sync generation, the SSIEN bit in the SCR should be cleared.

The transmitter and receiver timing for an 8-bit word with continuous clock, FIFO disabled, three words per frame sync in Network mode is shown in Figure 16-20 (“Network Mode Timing - Continuous Clock”).

**NOTE**

The transmitter repeats the value 0x5E because of an underrun condition

For the transmit section, the STMSK value is updated in the last time slot of frame 1, to mask the first two time slots (0x3). This value takes effect from the next time slot and consequently, the next frame transmits data in the third time slot only.

For the receive section, data received on the SRXD pin gets transferred to the Rx Data register at the end of each time slot. If the FIFO is disabled, the RDR flag gets set and causes a receiver interrupt if RE, RIE and RDR_EN bits are set. If the FIFO is enabled, then the RFF flag is used for interrupt generation (this flag is set in accordance with the watermark settings). Here all time slots are enabled. The receive data ready flag is set after reception of the first data (0x55). Since the flag is not cleared (Rx Data Register is not read by core), the Receive Overrun Error (ROE) flag is set on reception of the next data (0x5E). ROE flag is cleared on writing ‘1’ to the corresponding interrupt status bit in SSI Status Register.

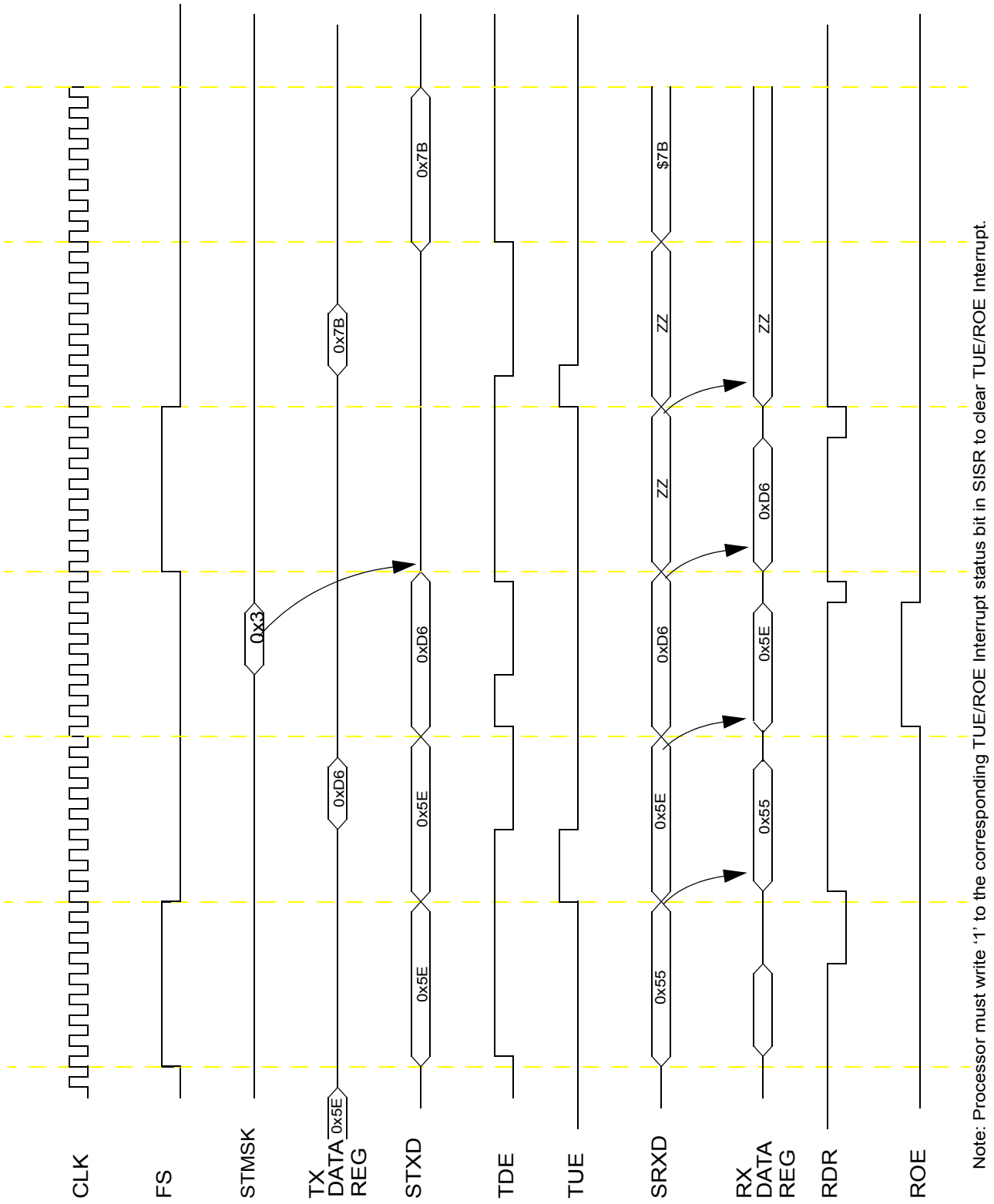



**Figure 16-20. Network Mode Timing - Continuous Clock**

### 16.4.4.3 Gated Clock Mode

Gated Clock mode is often used to hook up to SPI-type interfaces on Microcontroller Units (MCUs) or external peripheral chips. In Gated Clock mode, the presence of the clock indicates that valid data is on the STXD or SRXD ports. For this reason, no frame sync is needed in this mode. Once transmission of data has completed, the clock is pulled to the inactive state. Gated clocks are allowed for both the transmit and receive sections with either internal or external clock in Normal mode. Gated clocks are not allowed in Network mode. See Table 16-2 (“Clock Pin Configurations”) for SSI configuration for gated-mode operation.

The clock runs when the TE bit and/or the RE bit are appropriately enabled. For the case of internally generated clock, all internal bit clocks, word clocks, and frame clocks continue to operate. When a valid time slot occurs (such as the first time slot in Normal mode), the internal bit clock is enabled onto the appropriate clock port. This allows data to be transferred out in periodic intervals in Gated Clock mode. With an external clock, the SSI waits for a clock signal to be received. Once the clock begins, valid data is shifted in. Care should be taken to clear all DC bits (0x00000) when SSI is used in Gated mode.

For Gated clock operated in external clock mode, a proper clock signalling must be applied to the SSI STCK in order for it to function properly. If the SSI uses rising edge transition to clock data (TSCKP=0) and the falling edge transition to latch data (RSCKP=0), the clock must be in an active low state when idle. If the SSI uses falling edge transition to clock data (TSCKP=1) and the rising edge transition to latch data (RSCKP=1), the clock must be in a active high state when idle. Figure 16-21, through Figure 16-24 illustrate the different edge clocking/latching.

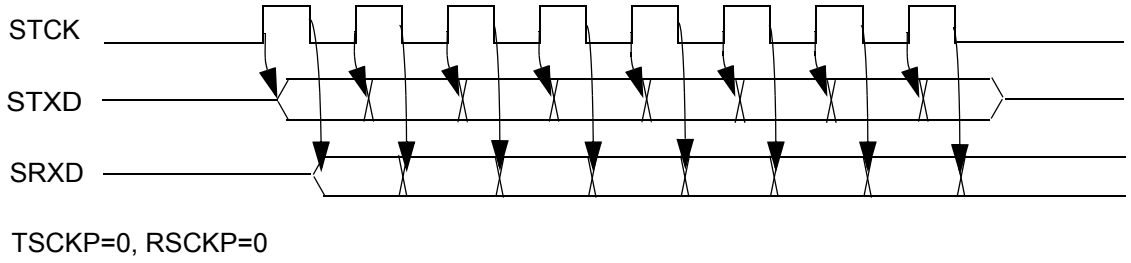



**Figure 16-21. Internal Gated Mode Timing - Rising Edge Clocking / Falling Edge Latching**

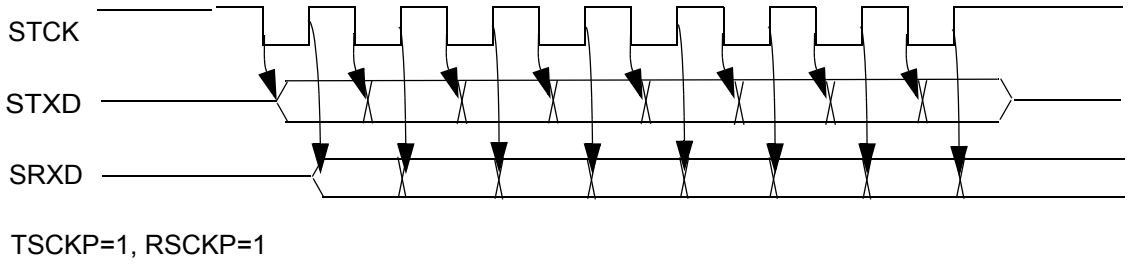



**Figure 16-22. Internal Gated Mode Timing - Falling Edge Clocking / Rising Edge Latching**

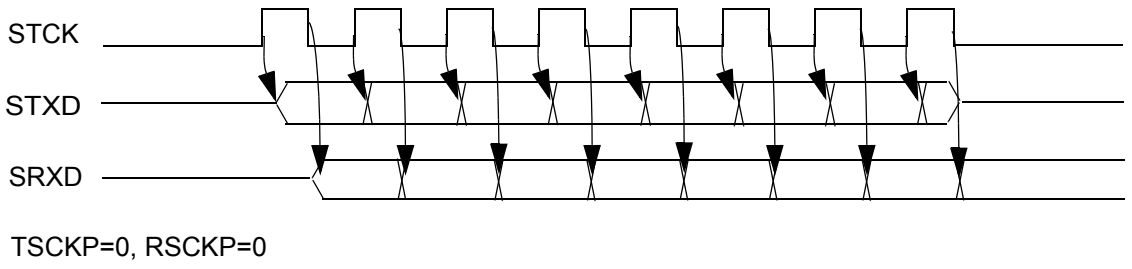



**Figure 16-23. External Gated Mode Timing - Rising Edge Clocking / Falling Edge Latching**

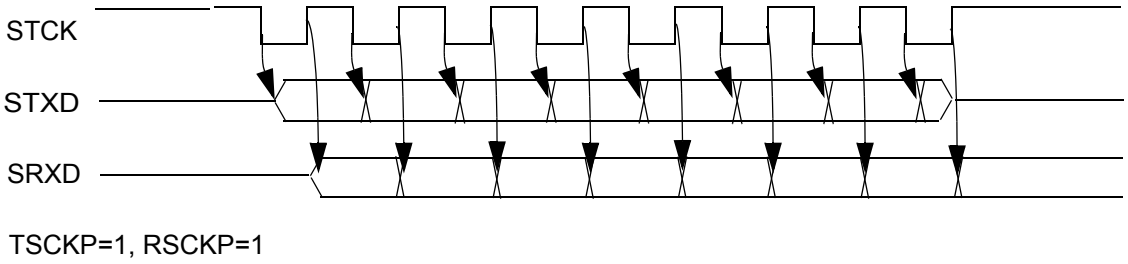



**Figure 16-24. External Gated Mode Timing - Falling Edge clocking / Rising Edge Latching**

**NOTE**

- The bit clock ports must be kept free of timing glitches. If a single glitch occurs, all ensuing transfers will be out of synchronization.
- In case of External Gated Mode, even though the Tx Data line is put in the high-impedance state at the last non-active edge of the bit clock, the round trip delay should be sufficient to take care of hold time requirements at the external receiver.

### 16.4.4.4 I2S Mode

The SSI is compliant to $I^2S$ bus specification from Philips Semiconductors (February 1986, Revised June 5, 1996). See Figure 16-25 for an illustration of the basic I2S protocol timing.

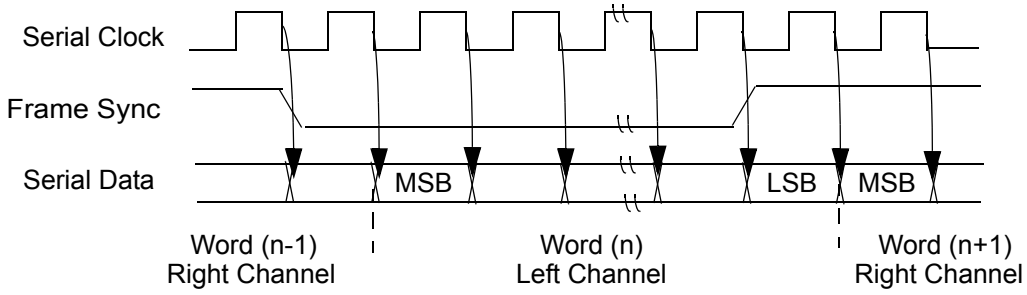



**Figure 16-25. $I^2S$ Mode Timing - Serial Clock, Frame Sync and Serial Data**

**NOTE**

The $I^2S$ bit clock (serial clock) is defined as 2.5 MHz +/- 10% in the specification. With the Peripheral Clock at 24 MHz, the closest available SSI $I^2S$ bit rate is 2.4 MHz

Select $I^2S$ mode using the options listed in Table 16-4.

**Table 16-4. I2S Mode Selection**

| $I^2S$_MODE[1] | I2S_MODE[0] | Mode Type |
|---|---|---|
| 0 | 0 | Normal mode |
| 0 | 1 | I2S master mode |
| 1 | 0 | I2S slave mode |
| 1 | 1 | Normal mode |

In normal mode operation, no register bits are forced to any particular state internally and the user can program the SSI to work in any operating condition.

When $I^2S$ modes are entered (I2S master (01) or I2S slave (10)), the following settings are recommended:

- Tx shift direction: MSB transmitted first (STCR[4]=0)
- Rx shift direction: MSB received first (SRCR[4]=0)
- Tx data clocked at falling edge of the clock (STCR[3]=1)
- Rx data latched at rising edge of the clock (SRCR[3]=1)
- Tx frame sync active low (STCR[2]=1)
- Rx frame sync active low (SRCR[2]=1)
- Tx frame sync initiated one bit before data is transmitted (STCR[0]=1)
- Rx frame sync initiated one bit before data is received (SRCR[0]=1)

In $I^2S$ master mode(SCR[6:5]=01), the following additional settings are recommended:

- TXDIR bit (STCR[5]) set to 1 to select internal generated bit clock
- TFDIR bit (STCR[6]) set to 1 to select internal generated frame sync

In $I^2S$ master mode(SCR[6:5]=01), the following settings are internally overridden by the hardware:

- Network mode is selected (SCR[3]=1)
- Tx frame sync length set to one-word-long-frame (STCR[1]=0)
- Rx frame sync length set to one-word-long-frame (SRCR[1]=0)
- Tx shifting w.r.t. bit 0 of TXSR (STCR[9]=1)
- Rx shifting w.r.t. bit 0 of RXSR (SRCR[9]=1)

The user needs to set the following control bits to configure the bit clock and frame sync:

- PM (STCCR[7:0])
- DC (STCCR[12:8])
- WL (STCCR[16:13]

- PSR (STCCR[17])
- DIV2 (STCCR[18])

The word length is fixed to 32 in $I^2S$ Master mode and the WL bits determine the number of bits that will contain valid data (out of the 32 transmitted/received bits in each channel). The fixing of word duration as 32 simplifies the relation between oversampling clock (Peripheral Clock) and Frame Sync (Peripheral Clock becomes an integer multiple of Frame Sync).

In $I^2S$ slave mode(SCR[6:5]=10), the following additional settings are recommended:

- TXDIR bit(STCR[5]) set to 0 to select external generated bit clock
- TFDIR bit(STCR[6]) set to 0 to select external generated frame sync

In $I^2S$ slave mode(SCR[6:5]=10), the following settings are done internally overridden by the hardware :

- Normal mode is selected (SCR[3]=0)
- Tx frame sync length set to one-bit-long-frame (STCR[1]=1)
- Rx frame sync length set to one-bit-long-frame (SRCR[1]=1)
- Tx shifting w.r.t. bit 0 of TXSR (STCR[9]=1)
- Rx shifting w.r.t. bit 0 of RXSR (SRCR[9]=1)

The user needs to set the following control bits to configure the data transmission:

- WL (STCCR[16:13])
- DC (STCCR[12:8])

The word length is variable in $I^2S$ slave mode and the WL bits determine the number of bits that will contain valid data. The actual word length is determined by the external CODEC. The external $I^2S$ Master still sends frame sync according to the $I^2S$ protocol (early, word wide and active low), the SSI internally operates so that each frame sync transition is the start of a new frame (the WL bits determine the number of bits to be transmitted/received). After one data word has been transferred, the SSI waits for the next frame sync transition to start operation in the next time slot. Transmit (STMSK) and receive (SRMSK) mask bits should not be used in $I^2S$ Slave mode of operation.

### 16.4.4.5 External Frame and Clock Operation

When applying external frame sync and clock signals to SSI, there should be at least 4-bit clock cycles between the enabling of the transmit or receive section and the rising edge of the corresponding frame sync signal. The transition of STFS or SRFS should be synchronized with the rising edge of external clock signal, STCK or SRCK.

### 16.4.4.6 Data Alignment Formats Supported

The SSI supports three data formats in order to provide flexibility with handling data. These formats dictate how data is written to (and read from) the data registers. Therefore, data can appear in different places in STX and SRX based on the data format and the number of bits per word. Independent data formats are supported for both the transmitter and receiver (that is, the transmitter and receiver can use different data formats).

The supported data formats are:

- MSB alignment
- LSB alignment
  - — Zero-extended (receive data only)
  - — Sign-extended (receive data only)

With MSB alignment, the most significant byte is bits 31 through 24 of the data register if the word length is larger than or equal to 16 bits. If the word length is less than 16 bits and MSB alignment is chosen, the most significant byte is bits 15 through 8. With LSB alignment, the least significant byte is bits 7 through 0. Data alignment is controlled by the TXBIT0 bit in the STCR and the RXBIT0 bit in the SRCR. See Table 16-5 for the bit assignment for all the data formats supported by the SSI.

**Table 16-5. Data Alignment**

| Format | Bit Number | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 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 |
| 8-bit LSB Aligned | | | | | | | | | | | | | | | | | | | | | | | | | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 8-bit MSB Aligned | | | | | | | | | | | | | | | | | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | | | |
| 10-bit LSB Aligned | | | | | | | | | | | | | | | | | | | | | | | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 10-bit MSB Aligned | | | | | | | | | | | | | | | | | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | |
| 12-bit LSB Aligned | | | | | | | | | | | | | | | | | | | | | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 12-bit MSB Aligned | | | | | | | | | | | | | | | | | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | |
| 16-bit LSB Aligned | | | | | | | | | | | | | | | | | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 16-bit MSB Aligned | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | | | | | | | | | | | |
| 18-bit LSB Aligned | | | | | | | | | | | | | | | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 18-bit MSB Aligned | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | | | | | | | | | |
| 20-bit LSB Aligned | | | | | | | | | | | | | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 20-bit MSB Aligned | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | | | | | | | |
| 22-bit LSB Aligned | | | | | | | | | | | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 22-bit MSB Aligned | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | | | | | |

**Table 16-5. Data Alignment (continued)**

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 24-bit LSB Aligned | | | | | | | | | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 24-bit MSB Aligned | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | | | |

In addition, receive data can either be zero-extended or sign-extended if LSB alignment is selected. With zero-extension, all bits above the most significant bit are 0's. This format is useful when data is stored in a pure integer format. With sign-extension, all bits above the most significant bit are equal to the most significant bit. This format is useful when data is stored in a fixed-point integer format (which implies fractional values). Receive data extension is controlled by the RXEXT bit in the SRCR. Transmit data used with LSB alignment has no concept of sign/zero-extension. Unused bits above the most significant bit are simply ignored.

When configured in I2S mode, the SSI forces the selection of LSB alignment. However, RXEXT still permits a choice between zero-extension and sign-extension.

Refer to Section 16.6.6 and Section 16.6.7 for more details on the relevant bits in the STCR and SRCR registers.

## 16.4.5 Internal Frame and Clock Shutdown

During transmit/receive operation, disabling TE/RE will ensure that data transmission/reception stops after current frame ends following which TFRC/RFRC Status bits will get set to indicate the Frame Completion State. If TFR_CLK_DIS/RFR_CLK_DIS bit is set in the current or any of the previous frames, SSI will stop driving the STFS/SRFS and STCK/SRCK signals after the current frame ends.

If TFR_CLK_DIS/RFR_CLK_DIS bit is not set, SSI will continue generating STFS/SRFS and STCK/SRCK signals (in case direction is from SSI), which then can be disabled by writing '1' to TFS_CLK_DIS/RFR_CLK_DIS bit. SSI will then stop driving these signals after end of frame is reached following which TFRC/RFRC status bits will get set to indicate the Frame Completion State.

Figure 16-26 is an illustration of transmission case where TXDIR and TFDIR are both set to '1'. In this case TE is disabled with TFS_CLK_DIS bit set in current or any of the previous frames.

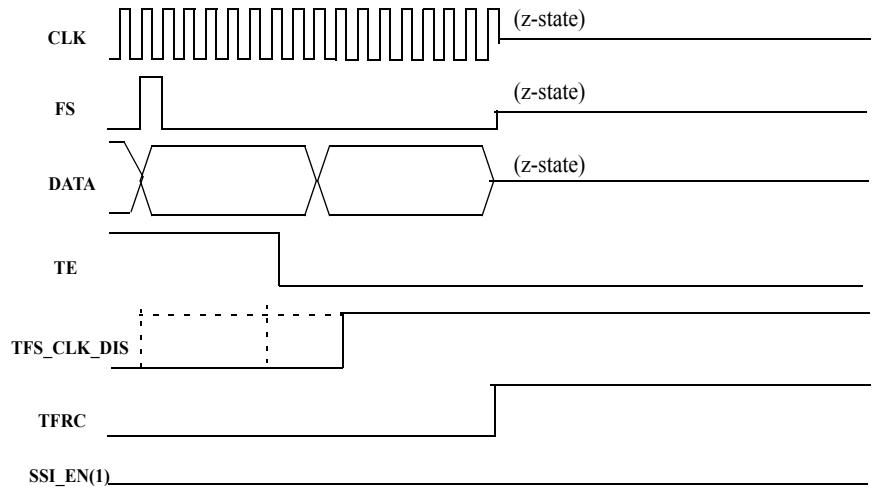



**Figure 16-26. TFS_CLK_DIS assertion in current or previous frame as TE disable**

Figure 16-27 is an illustration of transmission case where TXDIR and TFDIR are both set to '1'. In this case TFS_CLK_DIS bit is set after few frames of disabling TE. TFRC (Transmit Frame Complete) is set at frame boundary after TE is cleared. Once software services this interrupt and sets TFR_CLK_DIS bit later, TFRC bit is again set at next frame boundary.

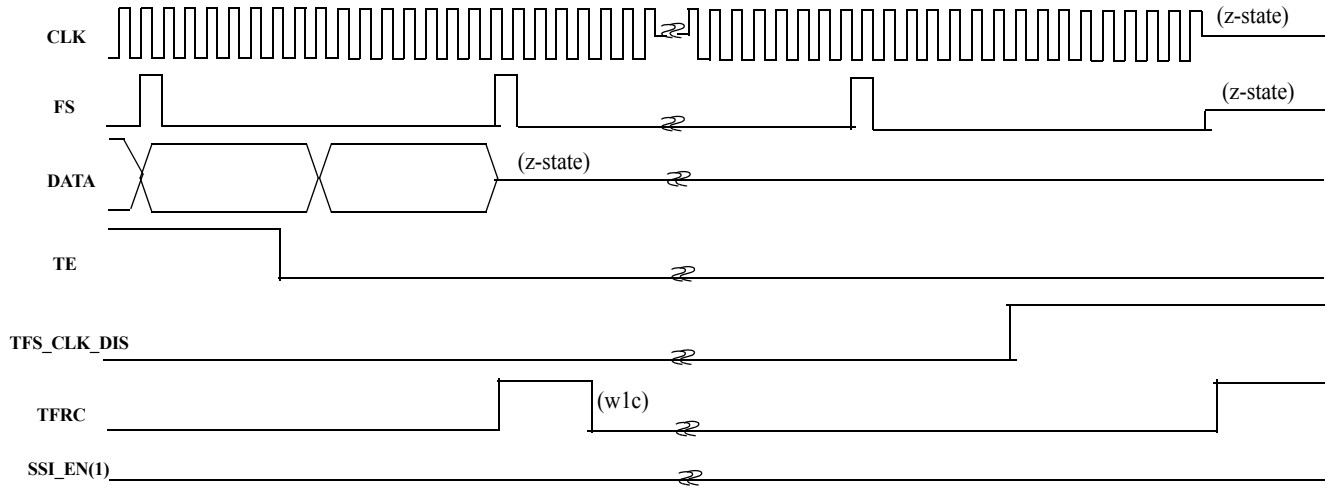



**Figure 16-27. TFS_CLK_DIS assertion in subsequent frame after disabling TE**

## 16.4.6 SSI Interrupts

The SSI module has a single interrupt request signal (IRQ) to the Interrupt Controller (ITC). The IRQ can be generated from 12 individual sources (see Table  for a summary of interrupt sources) that are OR'ed together.

- All interrupt request status bits are contained in the Interrupt Status Register (SSI_SISR)
- All interrupt request mask (enable) bit are contained in the Interrupt Enable Register (SSI_SIER)
- The SSI module must be enabled in the Control Register for any interrupt requests to be enabled

- For receiver-related interrupt requests
  - The receiver must be enabled in the Control Register
  - The Receive Interrupt Enable Bit (RIE) in the Interrupt Enable Register must be set
  - The individual receiver interrupt status enable bit must be set
  - For an RXFIFO-related request, the RXFIFO must be enabled in the Receive Configuration Register
- For transmitter-related interrupt requests
  - The transmitter must be enabled in the Control Register
  - The Transmit Interrupt Enable Bit (TIE) in the Interrupt Enable Register must be set
  - The individual transmit status interrupt enable bit must be set
  - For an TXFIFO-related request, the TXFIFO must be enabled in the Transmit Configuration Register

**NOTE**

- All the flags in the Status Register are updated after the first bit of the next SSI word has completed transmission or reception.
- See Section 16.6.4 and Section 16.6.5 for descriptions of individual status bits and interrupt enable bits.

**Table 16-6. SSI Interrupt Sources**

| Item | Status Bit | Mask Bit[1] | Source Description | Interrupt Clear Mechanism |
|---|---|---|---|---|
| 1 | RFRC | RFRC_EN | Receive Frame Complete | Writing a “1” to the RFRC bit clears the status. |
| 2 | TFRC | TFRC_EN | Transmit Frame Complete | Writing a “1” to the TFRC bit clears the status. |
| 3 | RDR | RDR_EN | Receive Data Ready | Emptying the RXFIFO by reading data from the SRX register clears the RDR status |
| 4 | TDE | TDE_EN | Transmit Data Register Empty | Writing data to the STX register clears the TDE status |
| 5 | ROE | ROE_EN | Receiver Overrun Error | Writing a “1” to the ROE bit clears the status. |
| 6 | TUE | TUE_EN | Transmitter Underrun Error | Writing a “1” to the TUE bit clears the status. |
| 7 | TFS | TFS_EN | Transmit Frame Sync | TFS status is set by first slot of frame. Status is cleared when next slot is received |
| 8 | RFS | RFS_EN | Receive Frame Sync | RFS status is set by first slot of frame. Status is cleared when next slot is received |
| 9 | TLS | TLS_EN | Transmit Last Slot | Writing a “1” to the TLS bit clears the status. |
| 10 | RLS | RLS_EN | Receive Last Slot | Writing a “1” to the RLS bit clears the status. |
| 11 | RFF | RFF_EN | Receive FIFO Full | When RXFIFO data level falls below programmed threshold, RFF status is cleared |
| 12 | TFE | TFE_EN | Transmit FIFO Empty | When TXFIFO data level meets programmed threshold, TFE status is cleared. |

[1] These are the individual interrupt request mask (enable) bits only

#### 16.4.6.1 Receive Data Status and Interrupt Requests

The receiver can generate an interrupt request based on available received data (RIE Bit must be set):

- RXFIFO not enabled - receive status includes:
  - RDR status is set when an RX word transfer is complete (the SRX register contains the data). An interrupt request is generated if the RDR_EN enable is set. The RDR status is cleared by reading the SRX register. The data must be removed before the next word transfer is complete.
  - ROE (receiver overrun) status will get set if the receive shift register attempts to load a received word into the SRX register BEFORE the register has been unloaded, i.e., it would overwrite previously received data. An IRQ can be enabled. If an overrun error occurs, the ROE status must be cleared via a register write.
- RXFIFO enabled - In this case, an additional FIFO status is available.
  - RDR status is set as long as the RXFIFO is not empty. This status can be used to read RXFIFO data until the FIFO is empty. An interrupt request can be enabled.
  - RFF status is set any time the RXFIFO contents are at or above the RXFIFO programmed watermark level. An RFF interrupt request can be used to wait until the FIFO is sufficiently full to initiate a data block transfer from the RXFIFO. The status is cleared once the RXFIFO contents fall below the watermark.
  - ROE (receiver overrun) status will get set if the receive shift register attempts to load a received word into the RXFIFO while the FIFO is full, i.e., it would overwrite previously received data. In IRQ can be enabled. In an overrun error occurs, the ROE status must be cleared via a register write.

#### 16.4.6.2 Transmit Data Status and Interrupt Requests

The transmitter can generate an interrupt request based on available transmit data status (TIE Bit must be set):

- TXFIFO not enabled - transmit status includes:
  - TDE (empty) status is set when a TX word in the STX register has been transferred into the TX shift register, meaning the STX register is ready for fresh data to be transmitted. An interrupt request is generated if the TDE_EN enable is set. The TDE status is cleared by writing new data to the STX register. The data must be written before the present word transfer is complete.
  - TUE (transmitter underrun) status will get set if the transmit shift register attempts to load a word from the STX register and the data is stale, i.e., the STX register has not received new data. An IRQ can be enabled. In an overrun error occurs, the TUE status must be cleared via a register write.
- TXFIFO enabled - In this case, an additional FIFO status is available.
  - TDE (empty) status is not set as long as the TXFIFO is not empty. An interrupt request is generated if the TDE_EN enable is set. The TDE status is cleared by writing new data to the STX register. The data must be written before the present word transfer is complete.
  - TFE (TXFIFO capacity available) status is set any time the TXFIFO contents are at or below the TXFIFO programmed watermark level. A TFE interrupt request can be used to wait until

the FIFO is sufficiently empty to initiate a data block transfer to the TXFIFO. The status is clear once the TXFIFO contents rise above the watermark

— TUE (transmitter underrun) status will get set if the transmit shift register attempts to load a word from the TXFIFO and no fresh data is available. An IRQ can be enabled. In an underrun error occurs, the TUE status must be cleared via a register write.

## 16.5 SSI Register Memory Map

The SSI module is programmed via a set of memory-mapped registers and their respective offset addresses are listed in Table 16-7.

- The SSI base address is 0x8000_1000
- The SSI only supports 32-bit transfers with all SSI registers.

**NOTE**

Addresses that are not described in the memory map are not implemented and should not be accessed. A transfer error will be generated if any of these addressed are accessed. Transfer bus errors are generated upon response to the following:

- Write transfer to a read-only register.
- Access to a reserved location
- Access to a register space beyond the last populated register of the SSI in memory page 0x8000_1000.

**Table 16-7. SSI Register Address Memory Map**

| Address | Register | Access Type | Access Width |
|---|---|---|---|
| Base + 0x00 | SSI Transmit Data Register (SSI_STX) | R/W | 32-bit only |
| Base + 0x04 | Reserved | | |
| Base + 0x08 | SSI Receive Data Register (SSI_SRX) | R | 32-bit only |
| Base + 0x0C | Reserved | | |
| Base + 0x10 | SSI Control Register (SSI_SCR) | R/W | 32-bit only |
| Base + 0x14 | SSI Interrupt Status Register (SSI_SISR) | R | 32-bit only |
| Base + 0x18 | SSI Interrupt Enable Register (SSI_SIER) | R/W | 32-bit only |
| Base + 0x1C | SSI Transmit Configuration Register (SSI_STCR) | R/W | 32-bit only |
| Base + 0x20 | SSI Receive Configuration Register (SSI_SRCR) | R/W | 32-bit only |
| Base + 0x24 | SSI Transmit Clock Control Register (SSI_STCCR) | R/W | 32-bit only |
| Base + 0x28 | Reserved | | |
| Base + 0x2C | SSI FIFO Control/Status Register (SSI_SFCSR) | R/W | 32-bit only |
| Base + 0x30 to 0x44 | Reserved | | 32-bit only |
| Base + 0x48 | SSI Transmit Time Slot Mask Register (SSI_STMSK) | R/W | 32-bit only |
| Base + 0x4C | SSI Receive Time Slot Mask Register (SSI_SRMSK) | R/W | 32-bit only |
| Base + 0x50 to Base + 0x58 | Reserved | | |

## 16.6 SSI Registers

The following sections provide descriptions of the SSI registers.

### 16.6.1 SSI Transmit Data Register (SSI_STX)

The SSI Transmit Data Register (SSI_STX) is written to provide transmit data to the TX FIFO. The FIFO is 8 words (24-bit word) deep, and SSI_STX is mapped to the TXFIFO. Multiple writes to the register loads its FIFO.

- As data are transmitted, data from the FIFO are transferred to the Transmit Shift Register (TXSR).
- Multiple writes to the STX register will not result in the previous data being over-written by the subsequent data. Protection from over-writing is present irrespective of whether the transmitter is enabled or not.
  - — Example 1: If the user writes Data1...Data9 to STX, Data9 will not overwrite Data1. Data1...Data8 are stored in the FIFO while Data9 is discarded.
  - — Example 2: If TXFIFO is not in use and the user writes Data1, Data2 to STX, Data2 will not overwrite Data1 and will be discarded.

**NOTE**

Enable SSI (SSIEN=1) before writing to SSI Transmit Data Register.

| SSI_STX | | | | | | | | | | | | | **Addr Base+0x00** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| STX[31:16] | Reserved | | | | | | | | STX23:16] | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | STX[15:0] | | | | | | | | | | | | | | | |
| TYPE | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 16-8. SSI_STX Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-24 | **Reserved** | Read only |
| 23-0 | **STX[23:0]** - SSI Transmit Data. Writing data to this 24-bit field loads a 24-bit word to the TXFIFO | Reading a register returns the last data written. |

## 16.6.2 SSI Receive Data Register (SSI_SRX)

The read-only SSI Receive Data Registers (SSI_SRX) is read to access the SSI receive data from the RX FIFO. The FIFO is 8 words (24-bit word) deep, and SSI_SRX is mapped to the RXFIFO. A register read removes the first received data from the RXFIFO.

| SSI_SRX | | | | | | | | | | | | | **Addr Base+0x08** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | Reserved | | | | | | | | SRX[23:16] | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | SRX[15:0] | | | | | | | | | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 16-9. SSI_SRX Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31-24 | **Reserved** | Read only |
| 23-0 | **SRX[23:0]** - SSI Receive Data. Reading data from this 23-bit field provides a 23-bit word from the RXFIFO | Read only |

## 16.6.3 SSI Control Register (SSI_SCR)

The SSI Control Register (SSI_SCR) is a 10-bit register used to control and setup the SSI block.

| SSI_SCR | | | | | | | | | | | | | Addr Base+0x10 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | Reserved | | | | | | | | | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | Reserved | | | | RFR_CLK_DIS | TFR_CLK_DIS | CLK_IST | Reserved | | I2S MODE[1:0] | | Rsvd | NET | RE | TE | SSIEN |
| TYPE | r | r | r | r | r/w | r/w | r/w | r | r | r/w | r/w | r | r/w | r/w | r/w | r/w |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 16-10. SSI_SCR Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31–12 | Reserved | Read only |
| 11 | **RFR_CLK_DIS** - Receive Frame Clock Disable. This bit provides the option to keep the Frame-sync and Clock enabled after the current receive frame, in which receiver is disabled by clearing RE bit.<br><br>**Note:** Writing to this bit has effect only when RE is disabled. | 1 = Stop Frame-sync/Clock generation at next frame boundary. This will be effective also in the case where the receiver is already disabled in current or previous frames.<br>0 = Continue Frame-sync/Clock generation after the current frame during which RE is cleared (default). This may be required when Frame-sync and Clocks are required from SSI , even when no data is to be received (default). |
| 10 | **TFR_CLK_DIS** - Transmit Frame Clock Disable. This bit provides the option to keep the Frame-sync and Clock enabled after the current transmit frame, in which transmitter is disabled by clearing TE bit.<br><br>**Note:** Writing to this bit has effect only when TE is disabled. | 1 = Stop Frame-sync/Clock generation at the next frame boundary. This will be effective also in the case where the transmitter is already disabled in current or previous frames.<br>0 = Continue Frame-sync/Clock generation after current frame during which TE is cleared (default). This may be required when Frame-sync and Clocks are required from SSI , even when no data is to be received (default) |
| 9 | **CLK_IST** - Clock Idle State. This bit controls the idle state of the transmit clock port during SSI internal gated mode. | 1 = Clock idle state is ‘1’.<br>0 = Clock idle state is ‘0’ (default) |
| 8-7 | Reserved | Read only |
| 6–5 | **I2S MODE[1:0]** - $I^2S$ Mode Select. These bits control SSI normal mode vs. $I^2S$ Master or $I^2S$ Slave mode.<br>• See Section 16.4.4.4 for a detailed description of $I^2S$ Mode of operation.<br>• See Table 16-4 (“Mode Selection”) | 00 = Normal mode (default)<br>01 = $I^2S$ master mode<br>10 = $I^2S$ slave mode<br>11 = Normal mode |
| 4 | Reserved | Read only |

**Table 16-10. SSI_SCR Register Bit Descriptions (continued)**

| Bit Number | Description | Operation |
|---|---|---|
| 3 | **NET** - Network Mode. This bit controls whether SSI is in network mode or not. | 1 = Network mode selected.<br>0 = Network mode disabled (default). |
| 2 | **RE** - Receive Enable. This control bit enables the receive section of the SSI.<br>• When this bit is enabled, data reception starts with the arrival of the next frame sync.<br>• If data is being received when this bit is cleared, data reception continues until the end of the current frame and then stops.<br>• If this bit is set again before the second to last bit of the last time slot in the current frame, then reception continues without interruption. | 1 = Receive section enabled.<br>0 = Receive section disabled (disabled) |
| 1 | **TE** - Transmit Enable. This control bit enables the transmit section of the SSI. It enables the transfer of the contents of the STX registers to the TXSR and also enables the internal transmit clock.<br>• The transmit section is enabled when this bit is set and a frame boundary is detected.<br>• When this bit is cleared, the transmitter continues to send data until the end of the current frame and then stops.<br>• Data can be written to the STX registers with the TE bit cleared (the corresponding TDE bit will be cleared).<br>• If the TE bit is cleared and then set again before the second to last bit of the last time slot in the current frame, data transmission continues without interruption.<br>• The normal transmit enable sequence is to write data to the STX register(s) and then set the TE bit.<br>• The normal disable sequence is to clear the TE and TIE bits after the TDE bit is set.<br>• In gated clock mode, clearing the TE bit results in the clock stopping after the data currently in TXSR has shifted out. When the TE bit is set, the clock starts immediately (for internal gated clock mode). | 1 = Transmit section enabled.<br>0 = Transmit section disabled (default) |
| 0 | **SSIEN** — SSI Enable. This bit is used to enable/disable the SSI module.<br>• When disabled, all SSI status bits are preset to the same state produced by the power-on reset, all control bits are unaffected, the contents of Tx and Rx FIFOs are cleared.<br>• When SSI is disabled, all internal clocks are disabled (except register access clock). | 1 = SSI is enabled.<br>0 = SSI is disabled (default) |

## 16.6.4 SSI Interrupt Status Register (SSI_SISR)

The SSI Interrupt Status Register (SSI_SISR) is used to monitor the SSI status. The status bits provided are used to generate interrupt requests and must be enabled properly to do so.

- See Section 16.4.6, "SSI Interrupts" for a description of interrupt request operation
- The Interrupt Enable Register (SSI_SIER) contains related interrupt enable (mask) bits
- Status flags are valid when SSI is enabled.
- All the flags in the SSI_SISR are updated after the first bit of the next SSI word has completed transmission or reception.
- Certain status bits are cleared by writing 1 to the corresponding interrupt status bit in SSI_SISR.

| SSI_SISR | | | | | | | | | | | | | Addr Base+0x14 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | Reserved | | | | | | | RFRC | TRFC | Reserved | | | | | | |
| TYPE | r | r | r | r | r | r | r | rw | rw | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | Rsvd | RDR | Rsvd | TDE | Rsvd | ROE | Rsvd | TUE | TFS | RFS | TLS | RLS | Rsvd | RFF | Rsvd | TFE |
| TYPE | r | r | r | r | r/w | r/w | r/w | r/w | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

**Table 16-11. SSI_SISR Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31– 25 | Reserved | Read only |
| 24 | **RFRC** - Receive Frame Complete. This flag is set at the end of the frame during which Receiver is disabled.<br>• If Receive Frame & Clock are not disabled in the same frame, this flag is also set at the end of the frame in which Receive Frame & Clock are disabled.<br>• See description of RFR_CLK_DIS ( add cross reference) bit for more details on how to disable Receiver Frame & Clock or keep them enabled after receiver is disabled.<br>• Write a 1 to this bit position to clear the status. | 1 = End of frame reached after disabling RE or disabling RFR_CLK_DIS, when receiver is already disabled.<br>0 = End of Frame not reached (default) |
| 23 | **TFRC** - Transmit Frame Complete. This flag is set at the end of the frame during which Transmitter is disabled.<br>• If Transmit Frame & Clock are not disabled in the same frame, this flag is also set at the end of the frame in which Transmit Frame & Clock are disabled.<br>• See description of TFR_CLK_DIS Bit, SSI_SCR Register, for more details on how to disable Receiver Frame & Clock or keep them enabled after receiver is disabled.<br>• Write a 1 to this bit position to clear the status. | 1 = End of frame reached after disabling TE or disabling TFR_CLK_DIS, when transmitter is already disabled.<br>0 = End of Frame not reached (default) |

**Table 16-11. SSI_SISR Register Bit Descriptions (continued)**

| Bit Number | Description | Operation |
|---|---|---|
| 22 - 15 | **Reserved** | Read only |
| 14 | **RDR** - Receive Data Ready. This flag bit is set when SRX or RXFIFO is loaded with a new value.<br>• If the RXFIFO is disabled, RDR is set when the SRX is loaded from the shift register. The bit is cleared when the SRX is read.<br>• If the RXFIFO is enabled, RDR is set anytime data is available in the RXFIFO. The RDR is cleared only when the FIFO is empty. | 1 = Receive data is available<br>0 = No receive data is available |
| 13 | **Reserved** | Read only |
| 12 | **TDE** - Transmit Data Register Empty. This flag is set whenever no transmit data is available in the STX or TXFIFO.<br>• If the TXFIFO is disabled, TDE is set when the STX register contents are transferred to TXSR. The bit is cleared when fresh TX is written to STX<br>• If the TXFIFO is enabled, TDE is set whenever no TX data is available in the TXFIFO. TDE is cleared when fresh TX data is written to STX (TXFIFO). | 1 = No transmit data is available<br>0 = Transmit data is available for TX |
| 11 | **Reserved** | Read only |
| 10 | **ROE** - Receiver Overrun Error. This flag is set when the RXSR is filled and ready to transfer to the SRX register or the RXFIFO (when enabled) and no space is available.<br>• If the RXFIFO is disabled, ROE is set when the receive shift register attempts to write the SRX and the SRX data has not been read by the CPU.<br>• If the RXFIFO is enabled, ROE is set when the receive shift register attempts to write the RXFIFO and no space is available.<br>• Write a 1 to this bit position to clear the status. | 1 = Receiver overrun exception has occurred<br>0 = No receiver overrun exception |
| 9 | **Reserved** | Read only |
| 8 | **TUE** - Transmitter Underrun Error. This flag is set when the TXSR is empty (no data to be transmitted), the TDE flag is set and a transmit time slot occurs.<br>• When a transmit underrun error occurs, the previous data is retransmitted.<br>• In Network mode, each time slot requires data transmission (unless masked through STMSK register), when the transmitter is enabled (TE is set).<br>• Write a 1 to this bit position to clear the status. | 1 = Transmitter underrun exception has occurred<br>0 = No transmitter underrun exception |
| 7 | **TFS** - Transmit Frame Sync. This flag indicates the occurrence of transmit frame sync.<br>• In Network mode, the TFS bit is set during transmission of the first time slot of the frame and is then cleared when starting transmission of the next time slot.<br>• Data written to the STX register during the time slot when the TFS flag is set, is sent during the second time slot (in Network mode) or in the next first time slot (in Normal mode).<br>• In Normal mode, this bit is always high. | 1 = Transmit frame sync occurred during transmission of last word written to STX register.<br>0 = No Occurrence of Transmit frame sync. |

**Table 16-11. SSI_SISR Register Bit Descriptions (continued)**

| Bit Number | Description | Operation |
|---|---|---|
| 6 | **RFS** - Receive Frame Sync. This flag indicates the occurrence of receive frame sync.<br>• In Network mode, the RFS bit is set when the first slot of the frame is being received. It is cleared when the next slot begins to be received.<br>• In Normal mode, this bit is always high. | 1 = Receive frame sync occurred during reception of next word in SRX register.<br>0 = No Occurrence of Receive frame sync |
| 5 | **TLS** -Transmit Last Time Slot. This flag indicates the last time slot in a frame.<br>• When set, it indicates that the current time slot is the last time slot of the frame.<br>• TLS is set at the start of the last transmit time slot<br>• TLS is cleared when the SSI_SISR is read while this bit set. | 1 = Current time slot is the last transmit time slot of frame.<br>0 = Current time slot is not last time slot of frame. |
| 4 | **RLS** - Receive Last Time Slot. This flag indicates the last time slot in a frame.<br>• When set, it indicates that the current time slot is the last receive time slot of the frame.<br>• RLS is set at the end of the last time slot<br>• RLS is cleared when the SSI_SISR is read while this bit set. | 1 = Current time slot is the last receive time slot of frame.<br>0 = Current time slot is not last time slot of frame. |
| 3 | **Reserved** | Read only |
| 2 | **RFF** - Receive FIFO Full. This flag is set when RXFIFO is enabled and the data level in RXFIFO reaches the programmed RXFIFO WaterMark threshold (RFWM) .<br>• An interrupt request can only be enabled when RXFIFO is enabled.<br>• RFF is automatically cleared when the data level in RXFIFO falls below the threshold.<br>• When RXFIFO is full, all further received data is ignored; the FIFO contents are not overwritten. | 1 = RXFIFO is full.<br>0 = Space available in RXFIFO. |
| 1 | **Reserved** | Read only |
| 0 | **TFE** - Transmit FIFO Empty. This flag is set when TXFIFO is enabled and the data level in TXFIFO falls below the programmed Tx FIFO WaterMark threshold (TFWM) .<br>• An interrupt request can only be enabled when TXFIFO is enabled<br>• The TFE bit is automatically cleared when the data level in TXFIFO equals or exceeds the threshold. | 1 = Transmit FIFO is space available.<br>0 = Transmit FIFO has no space available. |

## 16.6.5 SSI Interrupt Enable Register (SSI_SIER)

The SSI Interrupt Enable Register (SIER) is used to enable SSI interrupt requests.

- See Section 16.4.6, "SSI Interrupts" for a description of interrupt request operation
- The Interrupt Status Register (SSI_SISR) contains the related status bits
- The SSI Control Register (SSI_SCR) must first be set to enable interrupts.

| SSI_SIER | | | | | | | | | | | | | Addr Base+0x18 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | Reserved | | | | | | | RFRC_EN | TFRC_EN | Rsvd | RIE | Rsvd | TIE | Reserved | | |
| TYPE | r | r | r | r | r | r | r | rw | rw | r | r/w | r | r/w | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | Rsvd | RDR_EN | Rsvd | TDE_EN | Rsvd | ROE_EN | Rsvd | TUE_EN | TFS_EN | RFS_EN | TLS_EN | RLS_EN | Rsvd | RFF_EN | Rsvd | TFE_EN |
| TYPE | r | r/w | r | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r | r/w | r | r/w |
| RESET | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

**Table 16-12. SSI_SIER Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31– 25 | Reserved | Read only |
| 24 | **RFRC_EN** - Receive Frame Complete Interrupt Enable | 1 = Receive Frame Complete IRQ enabled<br>0 = Receive Frame Complete IRQ disabled (default) |
| 23 | **TFRC_EN** - Transmit Frame Complete Interrupt Enable. | 1 = Transmit Frame Complete IRQ enabled<br>0 = Transmit Frame Complete IRQ disabled (default) |
| 22 | **Reserved** | Read only |
| 21 | **RIE** - Receive Interrupt Enable. This control bit enables receiver related interrupt requests. | 1 = Receive related IRQs enabled<br>0 = Receive related IRQs disabled (default) |
| 20 | **Reserved** | Read only |
| 19 | **TIE** - Transmit Interrupt Enable. This control bit enables transmitter related interrupt requests. | 1 = Transmit related IRQs enabled<br>0 = Transmit related IRQs disabled (default) |
| 18-15 | **Reserved** | Read only |
| 14 | **RDR_EN** - Receive Data Ready Interrupt Enable. | 1 = Receive Data Ready IRQ enabled<br>0 = Receive Data Ready IRQ disabled (default) |
| 13 | **Reserved** | Read only |
| 12 | **TDE_EN** - Transmit Data Register Empty Interrupt Enable | 1 = Transmit Data Register Empty IRQ enabled (default)<br>0 = Transmit Data Register Empty IRQ disabled |

**Table 16-12. SSI_SIER Register Bit Descriptions (continued)**

| Bit Number | Description | Operation |
|---|---|---|
| 11 | **Reserved** | Read only |
| 10 | **ROE_EN** - Receiver Overrun Error Interrupt Enable | 1 = Receiver Overrun Error IRQ enabled<br>0 = Receiver Overrun Error IRQ disabled (default) |
| 9 | **Reserved** | Read only |
| 8 | **TUE_EN** -Transmitter Underrun Error Interrupt Enable | 1 = Transmitter Underrun Error IRQ enabled<br>0 = Transmitter Underrun Error IRQ disabled (default) |
| 7 | **TFS_EN** - Transmit Frame Sync Interrupt Enable | 1 = Transmit Frame Sync IRQ enabled<br>0 = Transmit Frame Sync IRQ disabled (default) |
| 6 | **RFS_EN** - Receive Frame Sync Interrupt Enable | 1 = Receive Frame Sync IRQ enabled<br>0 = Receive Frame Sync IRQ disabled (default) |
| 5 | **TLS_EN** - Transmit Last Slot Interrupt Enable | 1 = Transmit Last Slot IRQ enabled<br>0 = Transmit Last Slot IRQ disabled (default) |
| 4 | **RLS_EN** - Receive Last Slot Interrupt Enable | 1 = Receive Last Slot IRQ enabled<br>0 = Receive Last Slot IRQ disabled (default) |
| 3 | **Reserved** | Read only |
| 2 | **RFF_EN** - Receive Fifo Full Interrupt Enable | 1 = Receive Fifo Full IRQ enabled<br>0 = Receive Fifo Full IRQ disabled (default) |
| 1 | **Reserved** | Read only |
| 0 | **TFE_EN** - Transmit Fifo Empty Interrupt Enable | 1 = Transmit Fifo Empty IRQ enabled (default)<br>0 = Transmit Fifo Empty IRQ disabled |

## 16.6.6 SSI Transmit Configuration Register (SSI_STCR)

The SSI Transmit Configuration Register (STCR) is a read/write control register used to control the transmit operation of the SSI. STCR controls the direction of the bit clock and frame sync ports, STCK and STFS.

- The power-on reset clears all SSI_STCR bits.
- SSI reset does not affect the SSI_STCR bits.

| SSI_STCR | | | | | | | | | | | | | Addr Base+0x1C | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | Reserved | | | | | | | | | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | Reserved | | | | | | TX-BIT0 | Rsvd | TFEN | TF-DIR | TX-DIR | TSHFD | TSCKP | TFSI | TFSL | TEFS |
| TYPE | r | r | r | r | r | r | r/w | r | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 16-13. SSI_STCR Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31– 10 | **Reserved** | Read only |
| 9 | **TXBIT0** - Transmit Bit 0. This control bit allows SSI to transmit the data word from bit position 0 or 15/31 in the transmit shift register.<br>• The shifting data direction can be MSB or LSB first, controlled by the TSHFD bit.<br>• See Section 16.4.2.2, “SSI Transmit Shift Register (TXSR)” | 1 = Shifting with respect to Bit 0 of transmit shift register (LSB aligned) (default)<br>0 = Shifting with respect to Bit 31 (if word length = 16, 18, 20, 22 or 24) or bit 15 (if word length = 8, 10 or 12) of transmit shift register (MSB aligned). |
| 8 | **Reserved** | Read only |
| 7 | **TFEN** - Transmit FIFO Enable. This bit enables the transmit FIFO .<br>• When enabled, the TXFIFO allows eight samples to be transmitted by the SSI per channel (a 9th sample can be shifting out) before TDE0 bit is set.<br>• When the FIFO is disabled, an interrupt is generated when a single sample is transferred to the transmit shift register (provided the interrupt is enabled). | 1 - Transmit FIFO enabled.<br>0 - Transmit FIFO disabled (default) |
| 6 | **TFDIR** - Transmit Frame Direction. This bit controls the direction and source of the transmit frame sync signal.<br>• An Internally generated frame sync signal is sent out through the STFS port<br>• External frame sync is taken from the same port. | 1 = Frame Sync generated internally.<br>0 = Frame Sync is external (default) |

**Table 16-13. SSI_STCR Register Bit Descriptions (continued)**

| Bit Number | Description | Operation |
|---|---|---|
| 5 | **TXDIR** - Transmit Clock Direction. This bit controls the direction and source of the clock signal used to clock the TXSR.<br>• An internally generated clock is output through the STCK port<br>• External clock is taken from the same port. | 1 = Transmit Clock generated internally.<br>0 = Transmit Clock is external (default) |
| 4 | **TSHFD** - Transmit Shift Direction. This bit controls whether the MSB or LSB will be transmitted first in a sample.<br>• LSB is Bit 0.<br>• Affected by **TXBIT0**. See Section 16.4.2.2, "SSI Transmit Shift Register (TXSR)" | 1 = Data transmitted LSB first.<br>0 = Data transmitted MSB first (default) |
| 3 | **TSCKP** - Transmit Clock Polarity. This bit controls which bit clock edge is used to clock data out of the transmit section. | 1 = Data clocked out on falling edge of bit clock.<br>0 = Data clocked out on rising edge of bit clock (default) |
| 2 | **TFSI** - Transmit Frame Sync Invert. This bit controls the active state of the frame sync I/O signal for the transmit section of SSI. | 1 = Transmit frame sync is active low.<br>0 = Transmit frame sync is active high. |
| 1 | **TFSL** - Transmit Frame Sync Length. This bit controls the length of the frame sync signal to be generated or recognized for the transmit section. The length of a word-long frame sync is same as the length of the data word selected by WL[3:0]. | 1 = Transmit frame sync is one-clock-bit long.<br>0 = Transmit frame sync is one-word long (default) |
| 0 | **TEFS** - Transmit Early Frame Sync. This bit controls when the frame sync is initiated for the transmit section.<br>• The frame sync signal is de-asserted after one bit-for-bit length frame sync and after one word-for-word length frame sync.<br>• In the case of synchronous operation, the frame sync can also be initiated on receiving the first bit of data. | 1 = Transmit frame sync is initiated one bit before the data is transmitted.<br>0 = Transmit frame sync initiated as the first bit of data is transmitted (default) |

## 16.6.7 SSI Receive Configuration Register (SSI_SRCR)

The SSI Receive Configuration Register (SSI_SRCR) is a read/write control register used to direct the receive operation of the SSI. SRCR controls the direction of the bit clock and frame sync ports, SRCK and SRFS.

- Power-on reset clears all SRCR bits.
- SSI reset does not affect the SRCR bits.

| SSI_SRCR | | | | | | | | | | | | | Addr Base+0x20 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | Reserved | | | | | | | | | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | Reserved | | | | | RX-EXT | RX-BIT0 | Rsvd | RFEN | RX-DIR | RX-DIR | RSHFD | RSCKP | RFSI | RFSL | REFS |
| TYPE | r | r | r | r | r | r/w | r/w | r | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 16-14. SSI_SRCR Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31– 11 | **Reserved** | Read only |
| 10 | **RXEXT** - Receive Data Extension. This control bit allows the SSI to store the received data word in sign extended form. This bit affects data storage only in the case received data is LSB aligned (SRCR[9]=1) | 1 = Sign extension turned on.<br>0 = Sign extension turned off (default |
| 9 | **RXBIT0** - Receive Bit 0. This control bit allows SSI to receive the data word from bit position 0 or 15/31 in the transmit shift register.<br>• The shifting data direction can be MSB or LSB first, controlled by the RSHFD bit.<br>• See Section 16.4.3.2, “SSI Receive Shift Register (RXSR)” | 1 = Shifting with respect to Bit 0 of receive shift register (LSB aligned) (default)<br>0 = Shifting with respect to Bit 31 (if word length = 16, 18, 20, 22 or 24) or bit 15 (if word length = 8, 10 or 12) of receive shift register (MSB aligned). |
| 8 | **Reserved** | Read only |
| 7 | **RFEN** - Receive FIFO Enable. This bit enables the receive FIFO .<br>• When enabled, the FIFO allows 8 samples to be received by the SSI (per channel) (a 9th sample can be shifting in) before RDR0 bit is set.<br>• When the FIFO is disabled, an interrupt is generated when a single sample is received by the SSI (provided the interrupt is enabled). | 1 - Receive FIFO enabled.<br>0 - Receive FIFO disabled (default) |

**Table 16-14. SSI_SRCR Register Bit Descriptions (continued)**

| Bit Number | Description | Operation |
|---|---|---|
| 6 | **RFDIR** - Receive Frame Direction. This bit controls the direction and source of the transmit frame sync signal.<br>• An internally generated frame sync signal is sent out through the SRFS port<br>• External frame sync is taken from the same port. | 1 = Frame Sync generated internally.<br>0 = Frame Sync is external (default) |
| 5 | **RXDIR** - Receive Clock Direction. This bit controls the direction and source of the clock signal used to clock the RXSR.<br>• Internally generated clock is output through the SRCK port.<br>• External clock is taken from this port. | 1 = Receive Clock generated internally.<br>0 = Receive Clock is external (default) |
| 4 | **RSHFD** - Receive Shift Direction. This bit controls whether the MSB or LSB will be received first in a sample.<br>• LSB is Bit 0.<br>• Affected by **RXBIT0**. See Section 16.4.3.2, “SSI Receive Shift Register (RXSR)” | 1 = Data received LSB first.<br>0 = Data received MSB first (default) |
| 3 | **RSCKP** - Receive Clock Polarity. This bit controls which bit clock edge is used to latch in data for the receive section. | 1 = Data latched on rising edge of bit clock.<br>0 = Data latched on falling edge of bit clock. (default) |
| 2 | **RFSI** - Receive Frame Sync Invert. This bit controls the active state of the frame sync I/O signal for the receive section of SSI. | 1 = Receive frame sync is active low.<br>0 = Receive frame sync is active high. |
| 1 | **RFSL** - Receive Frame Sync Length. This bit controls the length of the frame sync signal to be generated or recognized for the receive section. The length of a word-long frame sync is same as the length of the data word selected by WL[3:0]. | 1 = Receive frame sync is one-clock-bit long.<br>0 = Receive frame sync is one-word long (default) |
| 0 | **REFS** - Receive Early Frame Sync. This bit controls when the frame sync is initiated for the receive section. The frame sync is disabled after one bit-for-bit length frame sync and after one word-for-word length frame sync | 1 = Receive frame sync initiated as the first bit of data is received.<br>0 = Receive frame sync is initiated one bit before the data is received (default) |

## 16.6.8 SSI Transmit and Receive Clock Control Register (STCCR)

The SSI Transmit and Receive (SSI_STCCR) register is a read/write register used to direct the operation of the SSI clock control. This register controls the SSI clock generator, bit and frame sync rates, word length, and number of words per frame for the serial data.

**NOTE**

Because the SSI only operates in synchronous mode, the SSI_STCCR register controls both the receive and transmit sections.

- The Peripheral Clock is the system clock input to the SSI (typically 24 MHz). Control of the Peripheral Clock is determined by the Clock and Reset Module (CRM) is the source for the SSI clock (Peripheral Clock).(see Section 5.9.1, “System Control (SYS_CNTL)”)
- Power-on reset clears all STCCR bits.
- SSI reset does not affect the STCCR bits

| SSI_STCCR | | | | | | | | | | | | | Addr Base+0x24 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | Reserved | | | | | | | | | | | | | DIV2 | PSR | WL[3} |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r/w | r/w | r/w |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | WL[2:0] | | | DC4[4:0] | | | | | PM[7:0] | | | | | | | |
| TYPE | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**NOTE**

- Reference Section 16.4.1, “SSI Clock Generation”
- Setting DIV2 = 0, PSR = 0, and PM = 0 is an invalid configuration and will cause the SSI to function incorrectly.

**Table 16-15. SSI_STCCR Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31– 19 | **Reserved** | Read only |
| 18 | **DIV2** - Divide By 2. This bit controls a divide-by-two divider in series with the rest of the prescalers.<br>0 - Divider bypassed.<br>1 - Divider used to divide clock by 2. | 1 = Initial divide-by-2 prescaler used (default)<br>0 = Divider bypassed. |
| 17 | **PSR** - Prescaler Range. This bit controls a fixed divide-by-eight prescaler in series with the variable prescaler. It extends the range of the prescaler for those cases where a slower bit clock is required. | 1 = Divide-by-8 Prescaler used<br>0 = Prescaler bypassed. |

**Table 16-15. SSI_STCCR Register Bit Descriptions (continued)**

| Bit Number | Description | Operation |
|---|---|---|
| 16–13 | WL[3:0] Word Length Control. This field is used to control the length of the data words being transferred by the SSI.<br>• This field controls the Word Length Divider in the Clock Generator.<br>• The field also controls the frame sync pulse length when the FSL bit is cleared.<br>• In I2S Master mode, the SSI works with a fixed word length of 32, and the WL bits are used to control the amount of valid data in those 32 bits.<br>• Word lengths of 8, 10, 12, 16, 18, 20, 22 or 24 bits are supported. Refer to Table | WL[3:0] values - See Table . |
| 12–8 | DC[4:0] - Frame Rate Divider Control. This field is used to control the divide ratio for the programmable frame rate dividers and sets the number of words in a frame.<br>• The divide ratio works on the word clock.<br>• In Normal mode, this ratio determines the word transfer rate.<br>• In Network mode, this ratio sets the number of words per frame. | DC[4:0] values -<br>• The divide ratio ranges from 1 to 32 in Normal mode and from 2 to 32 in Network mode.<br>• In Normal mode, a divide ratio of 1 (DC=00000) provides continuous periodic data word transfer. A bit-length frame sync must be used in this case.<br>• Values ranging from “00000” to “11111” can be used to control the number of words in a frame. |
| 7–0<br>PM7–PM0 | PM[7:0] - Prescaler Modulus Select. This field controls the prescale divider in the clock generator. This prescaler is used only in Internal Clock mode to divide the internal clock (ccm_ssi_clk). The bit clock output is available at the clock port.<br>A divide ratio from 1 to 256 (PM[7:0] = 0x00 to 0xFF) can be selected. Refer to Section 16.4.2.2, “SSI Transmit Shift Register (TXSR)” for details regarding settings. | PM[7:0] values -<br>• A divide ratio from 1 to 256 (PM[7:0] = 0x00 to 0xFF) can be selected<br>• See Section 16.4.1.2, “Register SSI_STCCR DIV2, PSR and PM[7:0] Field Descriptions” |

**Table 16-16. SSI Data Lengths (WL[3:0])**

| WL[3:0] (hex) | Number of Bits/Word | Supported Function |
|---|---|---|
| 0 - 2 | NA | No |
| 3 | 8 | Yes |
| 4 | 10 | Yes |
| 5 | 12 | Yes |
| 6 | NA | No |
| 7 | 16 | Yes |
| 8 | 18 | Yes |
| 9 | 20 | Yes |

**Table 16-16. SSI Data Lengths (WL[3:0]) (continued)**

| WL[3:0] (hex) | Number of Bits/Word | Supported Function |
|---|---|---|
| A | 22 | Yes |
| B | 24 | Yes |
| C - F | NA | No |

## 16.6.9 SSI FIFO Control/Status Register (SSI_SFCSR)

The SSI FIFO Control/Status Register provides the data level status of the RXFIFO and TXFIFO, as well as, sets the “watermark’ levels of both FIFOs used for generating status and interrupt requests.

| SSI_SFCSR | | | | | | | | | | | | | Addr Base+0x2C | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | Reserved | | | | | | | | | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | RFCNT[3:0] | | | | TFCNT[3:0] | | | | RFWM[3:0] | | | | TFWM[3:0] | | | |
| TYPE | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |

**Table 16-17. SSI_SFCSR Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31– 16 | **Reserved** | Read only |
| 15-12 | **RFCNT0[3:0]** - Receive FIFO Counter. This field indicates the number of data words in the receive FIFO. | RFCNT0[3:0] value -<br>• Value equals the number of words in RXFIFO<br>• Maximum value equals 0x10 (FIFO has maximum capacity of 8 words) |
| 11-8 | **TFCNT[3:0]** - Transmit FIFO Counter. This field indicates the number of data words in the transmit FIFO. | TFCNT0[3:0] value -<br>• Value equals the number of words in TXFIFO<br>• Maximum value equals 0x10 (FIFO has maximum capacity of 8 words) |

**Table 16-17. SSI_SFCSR Register Bit Descriptions (continued)**

| Bit Number | Description | Operation |
|---|---|---|
| 7-4 | **RFWM[3:0]** - Receive FIFO Full WaterMark. This field control set the threshold at which the RFF flag will be set. The RFF flag is set whenever the data level in RXFIFO reaches the selected threshold. | RFWM[3:0] value -<br>0x00 - Reserved<br>0x01 - RFF set when 1 or more words in RXFIFO<br>0x02 - RFF set when 2 or more words in RXFIFO<br>0x03 - RFF set when 3 or more words in RXFIFO<br>0x04 - RFF set when 4 or more words in RXFIFO<br>0x05 - RFF set when 5 or more words in RXFIFO<br>0x06 - RFF set when 6 or more words in RXFIFO<br>0x07 - RFF set when 6 or more words in RXFIFO<br>0x08 - RFF set when 8 words in RXFIFO |
| 3-0 | **TFWM[3:0]** - Transmit FIFO Empty WaterMark. This field control sets the threshold at which the TFE flag will be set. The TFE flag is set whenever the data level in TXFIFO falls below the selected threshold. | TFWM[3:0] value -<br>0x00 - Reserved<br>0x01 - TFE set when 1 or more empty slots in the TXFIFO<br>0x02 - TFE set when 2 or more empty slots in the TXFIFO<br>0x03 - TFE set when 3 or more empty slots in the TXFIFO<br>0x04 - TFE set when 4 or more empty slots in the TXFIFO<br>0x05 - TFE set when 5 or more empty slots in the TXFIFO<br>0x06 - TFE set when 6 or more empty slots in the TXFIFO<br>0x07 - TFE set when 7 or more empty slots in the TXFIFO<br>0x08 - TFE set when 8 empty slots in the TXFIFO |

## 16.6.10 SSI Transmit Time Slot Mask Register (SSI_STMSK)

The SSI Transmit Time Slot Register (SSI_STMSK) contains the.mask for setting the time slots in a transmission.

| SSI_STMSK | | | | | | | | | | | | | Addr Base+0x48 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | STMSK[31:16] | | | | | | | | | | | | | | | |
| TYPE | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | STMSK[15:0] | | | | | | | | | | | | | | | |
| TYPE | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 16-18. SSI_STMSK Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31– 0 | **STMSK[31:0] -** Transmit Mask. These bits indicate which slot has been masked in the current frame. Writing to this register controls the time slots in which the SSI transmits data. Each bit corresponds to the respective time slot in the frame.<br>• If a change is made to the register contents, the transmission pattern is updated from the next time slot.<br>• Transmit mask bits should not be used in I2S Slave mode of operation. | 1 = Time Slot masked (no data transmitted in this time slot).<br>0 = Valid Time Slot (default) |

### 16.6.11 SSI Receive Time Slot Mask Register (SRMSK)

The SSI Receive Time Slot Register (SSI_SRMSK) contains the.mask for setting the time slots in a reception.

| SSI_SRMSK | | | | | | | | | | | | | Addr Base+0x4C | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | BIT 16 |
| | SRMSK[31:16] | | | | | | | | | | | | | | | |
| TYPE | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | | | | |
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | SRMSK[15:0] | | | | | | | | | | | | | | | |
| TYPE | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w | r/w |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 16-19. SSI_STMSK Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 31– 0 | **STMSK[31:0] -** Receive Mask. These bits indicate which slot has been masked in the current frame. Writing to this register controls the time slots in which the SSI receives data. Each bit corresponds to the respective time slot in the frame.<br>• If a change is made to the register contents, the reception pattern is updated from the next time slot.<br>• Receive mask bits should not be used in I2S Slave mode of operation. | 1 = Time Slot masked (no data received in this time slot).<br>0 = Valid Time Slot (default) |

## 16.7 Initialization/Application Information

The SSI is affected by the following types of reset:

- Power-on Reset—The Power-on reset is generated by asserting the hardware. The power-on reset clears the SSIEN bit in SCR, which disables the SSI.
- SSI Reset—The SSI reset is generated when the SSIEN bit in the SCR is cleared. The SSI status bits are preset to the same state produced by the Power-on reset. The SSI control bits are unaffected. The control bits in the SCR are also unaffected. The SSI reset is useful for selective reset of the SSI without changing the present SSI control bits and without affecting the other peripherals.

The correct sequence to initialize the SSI is as follows:

1. Issue an SSI reset (SCR[SSIEN]=0).
2. Set all control bits for configuring the SSI).
3. Enable appropriate interrupts requests through SIER.

4. Set the SCR[SSIEN] bit (=1) to enable the SSI.
5. Set SCR[TE/RE] bits.
6. To ensure proper operation of the SSI, use the SSI reset before changing any of the SSI Control" bits listed in Table 16-20).

**NOTE**

These control bits should not be changed when SSI is enabled.

**Table 16-20. SSI Control Bits Requiring SSI to be Disabled Before Change**

| Control Register | Bit |
|---|---|
| SSI_SCR | [9]=CLK_IST<br>[6:5]=I2S_MODE<br>[3]=NET |
| SSI_SRCR<br>SSI_STCR | [9]=RXBIT0<br>[9]=TXBIT0<br>[7]=RFEN<br>[7]=TFEN<br>[6]=RFDIR<br>[6]=TFDIR<br>[5]=RXDIR<br>[5]=TXDIR<br>[4]=RSHFD<br>[4]=TSHFD<br>[3]=RSCKP<br>[3]=TSCKP<br>[2]=RFSI<br>[2]=TFSI<br>[1]=RFSL<br>[1]=TFSL<br>[0]=REFS<br>[0]=TEFS |
| SSI_STCCR | [16-13]=WL[3:0] |

# Chapter 17
# Analog to Digital Converter Module (ADC)

## 17.1 Overview

The ADC module manages the interface to external sensors, scans multiple channels, and controls the warm-up of the analog portions of the analog to digital system. The ADC module can be set to generate interrupt requests based on programmed compare values that can be set for up to 8 channels. The primary ADC (ADC_1) can be programmed to convert/monitor 9 channels (8 external analog channel pins plus internal battery reference voltage); the secondary ADC (ADC_2) is limited to the 8 analog channel pins.

**NOTE**

Although this chapter provides reference information on the ADC module, the user/programmer is directed toward the software utility entitled the "Analog to Digital Converter (ADC) Driver", see Appendix B, "MC1322x Software Driver Utilities".

## 17.2 Features

The ADC module has the following features:

- 12-bit resolution. Effective number of bits 8-9
- Valid usable input voltage range: [Vref_high-0.2V] to [Vref_low+0.2V]
- Maximum input range: VBATT to VSS
- Minimum sample time 20us
- Peripheral Clock (set by CRM) provides the timebase for the ADC module clocks
- Two independent channels, each with a 32-bit timer
- ADC_1 has 9 channels: 8 external analog inputs plus battery reference voltage
- ADC_2 has 8 channels: 8 external analog inputs
- Active channels for each ADC are programmable
- Eight active monitors plus battery reference monitors can generate a IRQ
- An 8-deep FIFO for recording data
- IRQs can be generated by the channel compare values, FIFO status, and sequencers

## 17.3 Block diagram

The ADC Block Diagram is shown in Figure 17-1.

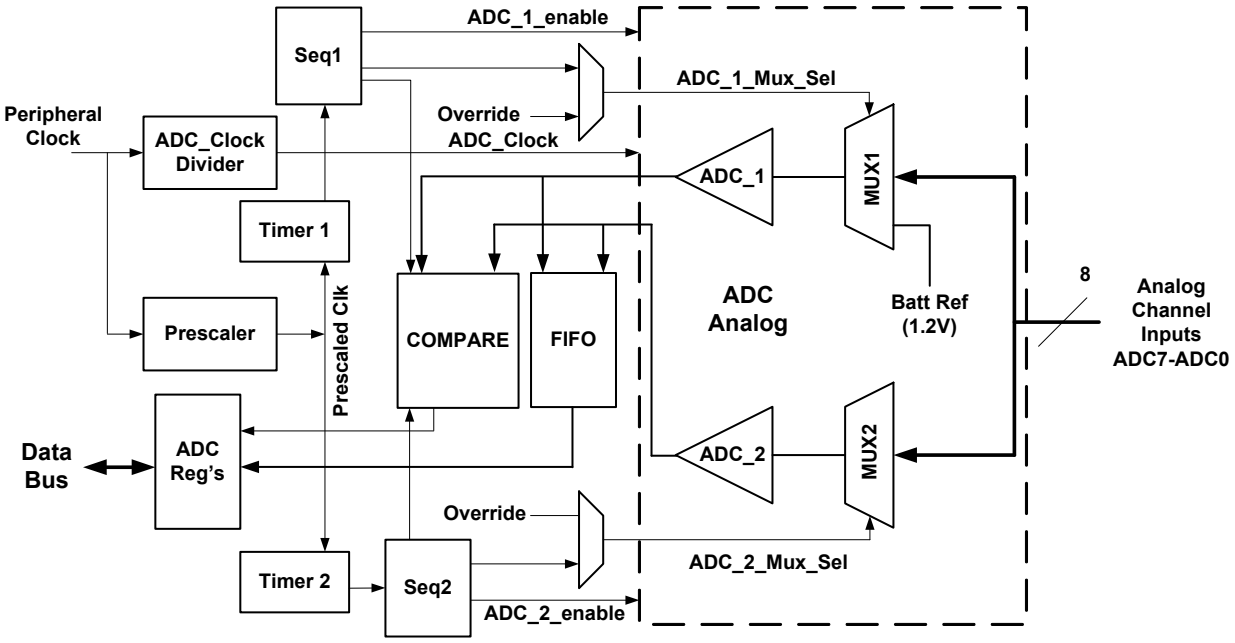



**Figure 17-1. ADC Block Diagram**

## 17.4 External Signal Description

The ADC module supports eight external analog input channel channels and separate reference voltage pins for ADC_1 and ADC_2. There are not dedicated power supply pins for the ADC. Table 17-1 lists the external signals.

**Table 17-1. ADC External Signal Descriptions**

| Signal Name | Description |
|---|---|
| ADC0-ADC7 | Analog channel input pins (also GPIO) |
| ADC1_VREFH | High reference voltage for ADC_1 |
| ADC1_VREFL | Low reference voltage for ADC_1 |
| ADC2_VREFH | High reference voltage for ADC_2[1] |
| ADC1_VREFL | Low reference voltage for ADC_2[1] |

1 Defaults to GPIO input from reset and is used to test for FLASH erase.

**NOTE**

The GPIO module chapter Section 11.5, "Special Conditions and Signal Usage" discusses active reset, default conditions, low power operation, and shared functionality of GPIO pins including the analog signals. Review this section including Table 11-6.

### 17.4.1 Analog Channel Input Pins (ADC0-ADC7)

There are a total of 8 pins that can be used as ADC analog input channels. Each channel is connected to an analog multiplexer for ADC_1 and an analog multiplexer for ADC_2. Each channel is accessible from either ADC. These signals can also function as GPIO, and are controlled by the GPIO module (see Chapter 11, "General Purpose I/O Module").

**NOTE**

- Each required channel must be enabled for analog mode via its GPIO_FUNC_SEL field
- The digital pad for a channel is tristated (completely off) when in analog mode and the ADC mode is enabled.
- The digital mode bus keeper function must be disabled when analog mode is enabled
- ADC7 is shared with Return Clock (RTCK) for the JTAG debug port. It is suggested to either use other ADC input channels where possible, or in a target application allow ADC7_RTCK to be disconnected from the debug port via a jumper or 0-ohm resistor to allow debug while using the ADC7 analog channel.

### 17.4.2 ADC_1 Reference Pins (ADC1_VREFH, ADC1_VREFL)

These external pins can be selected as the reference voltage potential for ADC_1, or alternatively, the ADC_1 reference voltage can be internally selected to VBATT and VSS.

- The ADC_1 reference voltage is selected by "AD1_Vrefhl_En" Bit of the ADC_CONTROL Register.
- If the external ADC1_VREFH and ADC1_VREFL pins are selected as the voltage reference, they must also be configured by the GPIO module
  - Analog mode must be selected via their GPIO_FUNC_SEL fields.
  - Bus keeper must be disabled
- These signals default to their analog function

### 17.4.3 ADC_2 Reference Pins (ADC2_VREFH, ADC2_VREFL)

These external pins can be selected as the reference voltage potential for ADC_2, or alternatively, the ADC_2 reference voltage can be internally selected to VBATT and VSS.

- The ADC_2 reference voltage is selected by "AD2_Vrefhl_En" Bit of the ADC_CONTROL Register.
- If the external ADC2_VREFH and ADC2_VREFL pins are selected as the voltage reference, they must also be configured by the GPIO module
  - Analog mode must be selected via their GPIO_FUNC_SEL fields.
  - Bus keeper must be disabled

- These signals are special in that they default to inputs used to test for FLASH erase at BOOT when exiting reset. They should be treated accordingly.

## 17.5 Functional Description

The block diagram of the ADC module is shown in Figure 17-1. The module consists of 8 analog input channels (from external inputs), two analog multiplexers (one for each ADC), A-to-D converters ADC_1 and ADC_2, sample FIFO, 8 general comparators, two control sequencers (one for each ADC), timing control counters, and the control register block.

Each ADC can be programmed to sample any input channel (ADC_1 can also sample a battery reference voltage). There are two basic modes of operation:

- Manual or override mode - this is used to enable a single channel sample and is directly under software control.
- Automated mode - each ADC has a sequencer that can be programmed as to which channels to sample and can generate interrupt requests based on a sample being above or below preset comparator values. The sample rate is based on the sequencer (simple convert rate) or the timer sample rate:
  - Convert rate - the sample time is determined by the ADC_CONVERT_TIME register. In this scenario, all channels are looked-at in sequence, but only the selected active channels are sampled/captured. The captured sample data value is entered into the FIFO with an associated channel number. The next channel is evaluated based on convert time.
  - Timer rate - each sequencer has an associated 32-bit timer. In this scenario, the timer between sampling active channels is based on the programmed timer delay.

In any mode of operation, the ADCs will continue to run as long as enabled. In manual mode, an enabled ADC will continuously update the sample value for the selected channel.

Interrupt requests can be generated from different types of events:

- FIFO filled to a pre-determined level
- Sequencer 1 having completed a cycle
- Sequencer 2 having completed a cycle
- Comparator event - captured sample exceeds limit set in comparator

### 17.5.1 Clocks and Timing

The primary clock source to the ADC module is the peripheral clock as set by the prescaler in the SYS_CNTL Register (XTAL_CLKDIV[5:0] field) in the CRM (see Section 5.9.1, “System Control (SYS_CNTL)”). Common usage is a default 24MHz reference oscillator with prescaler divide-by-1, or the peripheral clock is also 24 MHz. All the ADC clocks are derived from the peripheral clock. Figure 17-2 shows the ADC module clock generation blocks.

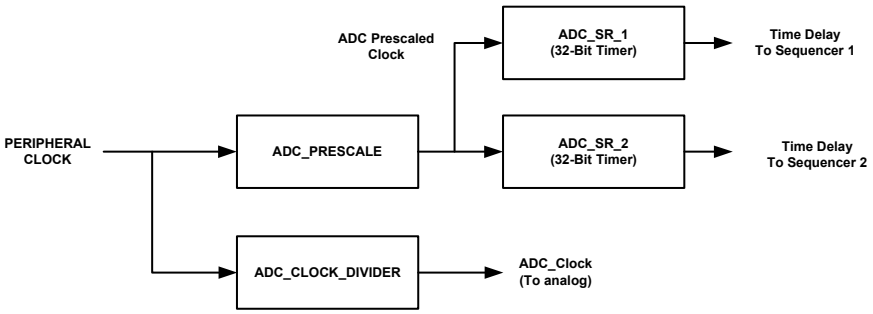



**Figure 17-2. ADC Clocks**

### 17.5.1.1 Analog ADC_Clock

The ADC analog circuitry requires an ADC_Clock which gets generated from the peripheral clock by the ADC_CLOCK_DIVIDER counter.

- ADC_Clock should be targeted for <= 300kHz
- A conversion requires 6 clocks of ADC_Clock (minimum of 20µs with a 300kHz frequency)
- ADC_CLOCK_DIVIDER Register - contains a 16-bit values that sets the divide value of the ADC_Clock divider:

ADC_Clock = Peripheral Clock / ADC_CLOCK_DIVIDER

As an example for 24MHz peripheral clock:

300kHz = 24Mhz / 80

Set ADC_CLOCK_DIVIDER = 0x0050 (80dec)

#### 17.5.1.1.1 ADC Prescale Clock

The prescale clock is used generate the time delays for cycling the ADC sequencers and for generating the on-time and convert-time.

- The prescale clock can be targeted for about 1MHz
- The ADC_PRESCALE Register - contains an 8-bit value that sets the prescale number N for the prescale clock where N = value + 1.

Prescale Clock = Peripheral Clock / (ADC_PRESCALE + 1)

As an example for 24MHz peripheral clock:

1MHz = 24Mhz / 24

Set ADC_PRESCALE = 0x17 (23dec)

#### 17.5.1.1.2 On Time and Convert Time

The analog circuitry requires:

1. A “warm-up” or enable time of 10µs minimum.
2. Six ADC_Clock edges to do a sampled data conversion (when enabled).

The digital circuitry uses two programmed time delays called ON time and CONVERT time to control required the analog timing. These two delays are based on the Prescale Clock and are programmable:

- ON time - this delay is the time between when an ADCx_enable signal is given to an ADC and when the conversion time starts. As stated it must be 10µs or longer and is programmed via ADC Turn-On Time Register (ADC_ON_TIME) (see 17.7.16/17-25).
- CONVERT time - this is the time allowed for a sample-data conversion to occur. As stated it should be six ADC_Clocks or longer (20µs minimum) and is programmed via ADC Convert Time Register (ADC_CONVERT_TIME) (see 17.7.17/17-26).

**NOTE**

The CONVERT time can be kept to a minimum when using the Timer mode of sequencer operation or can be extended to provide a desired sample cycle time when using the “convert rate” mode of sequencer operation.

#### 17.5.1.1.3 Sequencer Timers (ADC_SR_1 and ADC_SR_2)

The 32-bit timers as set by registers ADC_SR_1 and ADC_SR_2 generate the time delays for cycling the ADC sequencers.

- Each 32-bit timer is programmed via two 16-bit registers designated as ADC_SR_x_HIGH and ADC_SR_x_LOW. The 16-bit registers provide a concantenated 32-bit divide value.
- The clock input into the sequence timers is the prescale clock.

### 17.5.2 Analog ADC Operation

The ADC module has two single-ended, 12-bit resolution analog-to-digital converters.

- The ADC_Clock must be active for conversion
  - — ADC_Clock <= 300kHz
  - — Six ADC_Clocks are required for a conversion (20µs minimum)
- Each ADC is enabled individually - a “turn-on” time of 10µs minimum must be provided
- The reference voltages for each ADC are independent
  - — The associated external voltage pins can be selected (VREFH and VREFL)
  - — Alternatively, the internally connected VBATT voltage and ground (VSS) can be selected
- Each ADC has an independent analog channel multiplexer
  - — The selected channel is determined by either the proper sequencer or the manual channel select field
  - — The manual select field only enables a given channel for repeated conversion during override mode
  - — ADC_1 also has a fixed voltage (1.2Vdc) input called the Battery Reference input that can be used to determine the present supply voltage at the VBATT pin.

**NOTE**

As stated each ADC is enabled individually, either in the auto sequencer mode or the manual (override) mode. Once enabled, the ADC is continuously running and generating sampled data at the "six ADC_Clock" conversion rate. The sample will be generated from the analog mux channel presently selected. Implications include:

- In auto mode as samples get generated, they are loaded into the FIFO as enabled by the sequencer mode and timing
- In manual mode, the samples are reflected in the ADC Result Registers, and are read at the discretion of the software.

## 17.5.3 Operational Modes

As briefly described earlier, the ADC has two basic modes of operation, i.e., manual (override) mode and automated mode. Common to both modes:

- The ADC module must be enabled - "On" Bit set in ADC_CONTROL Register
- The Prescale Clock and ADC_Clock both must be configured
- The ON and CONVERT times must be configured

### 17.5.3.1 Manual (Override) Mode

Although the more direct mode, manual operation is not the default; it must be enabled via the ADC_MODE Register (see 17.7.21/17-30). This mode allows either or both ADCs to be sampling data continuously on a selected channel without use of the sequencers and timers, and is controlled by the ADC Manual Control Register (ADC_OVERRIDE). The application must provide its own cyclical timing for multiple samples as no interrupt request is available to signify completion a sample cycle. The application must initiate and enable the mode, and then it may read sampled values from a selected channel once or multiple times for averaging.

To use manual mode:

- Set the **Override** Bit in ADC_MODE Register
- The clocks along with On-time and Convert-time must be configured.
- Within the ADC_OVERRIDE Register
  - — Select the desired analog input channel via the **Mux2[3:0]** and/or **Mux1[3:0]** analog mux select fields
  - — Enable the desired ADC via the **ADC2_On** and/or **ADC1_On** Bits
- The **On** Bit in the ADC Module Primary Control Register (ADC_CONTROL) must be set
- Read the sampled results in the ADC_2_RESULT and/or ADC_1_RESULT Registers - the initial read must wait the sum of the ON and CONVERT times before accessing a result; after, a new sample will be generated every 6 ADC_Clocks.

**NOTE**

In manual mode the ADCs will continue to run and update the results registers with new samples as long as the ADCs are enabled (ADCx_On is set). There are no flags to indicate updated data.

### 17.5.3.2 Automated Mode (Using Sequencers, Timers, and FIFO)

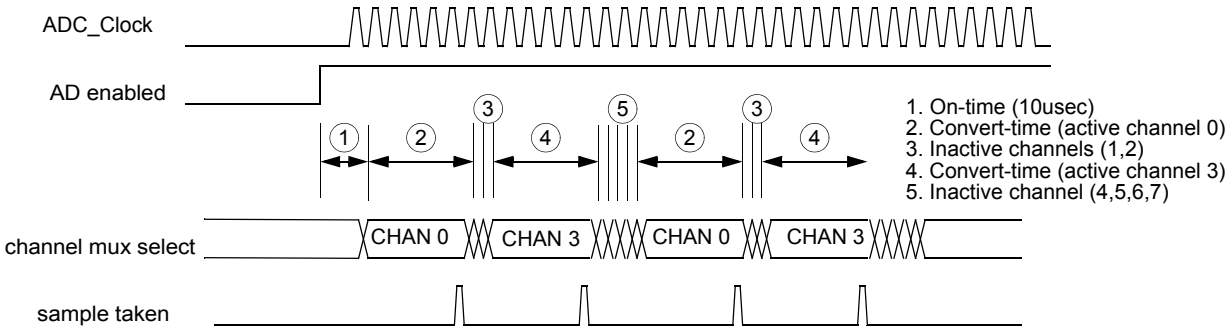
Automated operation is available by default (the **Override** Bit in ADC_MODE Register is disabled by default). This mode allows either or both ADCs to be sampling data continuously on selected channels through use of the sequencers, optionally with the timers, and results stored in the 8-sample deep FIFO. In addition to gathering samples on a timed basis, comparators are available to generate status flags and interrupt requests based on sample results.

#### 17.5.3.2.1 Sequencers (Seq 1 and Seq 2) and Timers

There is an independent sequencer for each ADC (Seq 1 and Seq 2); each managed by its Sequencer X Mode Control Register (ADC_SEQ_X). The sequencers must also be enabled via the ADC Module Primary Control Register (ADC_CONTROL).

The sequencers are state machines that cycle through all enabled analog input channels based upon a selected timebase option:

- Sequence based on convert time (see Section 17.7.17, "ADC Convert Time Register (ADC_CONVERT_TIME)"- this option is used typically for faster sample rate. The convert time is based on the Prescale Clock (example 1 MHz), is 20 µs minimum, and the active channels are sampled at this rate. The overall cycle time equates to about the number of active channels times the conversion time. Figure 17-3 shows an example of conversion rate timing.




NOTE: This timing scenario represents the case where ADC channels 0 and 3 are active and 1,2,4,5,6,7 are inactive.

**Figure 17-3. Sample Timing Based on Convert Rate**

- Sequence based on Timer 1 and Timer 2 - Seq 1 and Seq 2 each have an associated delay timer also based on the Prescale Clock. The time between active selected channel samples is based on the programmed timer delay as opposed to the conversion rate.

To use sequencer mode:

- The **Override** Bit in ADC_MODE Register must be disabled (default)

- The clocks along with On-time and Convert-time must be configured.
- Within the appropriate Sequencer Mode Control Register (ADC_SEQ_X)
  - — Select the desired analog input channels via **Batt** and **Ch7:Ch0** Bits
  - — Enable the desired mode via the **Seq_Mode** Bit
- If the timer mode is used
  - — Configure the timer delay via the appropriate ADC Sample Rate High/Low Registers (ADC_SR_X_HIGH and ADC_SR_X_LOW)
  - — Time must be enabled via the appropriate **TimerX_On** Bit in the ADC Module Primary Control Register (ADC_CONTROL)
- The **On** Bit in the ADC Module Primary Control Register (ADC_CONTROL) must be set
- Results are read from the FIFO (ADC_FIFO_READ Register)

**NOTE**

In sequencer mode the ADCs will continue to run at the programmed rate(s) and update the FIFO registers with new samples as long as the ADCs are enabled. The FIFO must be unloaded in a timely manner, in that, if the FIFO is full, a new sample will not be written (old data will not be overwritten with new data).

#### 17.5.3.2.2 FIFO

The sampled data FIFO is 8 words deep and provides a 12-bit unsigned binary magnitude and a 4-bit channel number for each sample. The channel data are written into the FIFO from both ADCs based on the sequencer timing and there is no pre-determined order. The data are written only from sequencer mode, not from manual mode. The register interface for the FIFO includes:

1. ADC FIFO Read Register (ADC_FIFO_READ) - a read access to this register unloads one word from the FIFO and decrements the FIFO level (count).
2. ADC FIFO Control Register (ADC_FIFO_CONTROL) - an interrupt request based on FIFO contents level can be enabled. A 4-bit field in this register sets the level (values 1 to 8) at which the FIFO level will trigger a “Level” status and IRQ.
3. ADC FIFO Status Register (ADC_FIFO_STATUS) - this register reports FIFO status of **Empty**, **Full**, and **Level[3:0]**. The Level status can (meeting or exceeding the preset level) trigger a FIFO IRQ.

### 17.5.4 Comparators

In addition to self-timed sampling through the sequencers, the ADC module has a block of comparators to test for channel data being above or below a programmed threshold. These comparators can be used to cause an IRQ for intervention of the CPU based upon the sampled data, and also to monitor VBATT supply voltage for battery and switching regulator management.

#### 17.5.4.1 ADC Comparators Channels [7:0]

There are a block of 8 comparators, each assigned to a dedicated analog input channel. Section 17.7.1, "ADC Compare Registers (ADC_COMP_0:ADC_COMP_7)" describes the associated control register for each comparator.

- Comparator 0 can only be used with Channel 0, and each respective comparator can only be used with its associated channel (1 to 1, 2 to 2, etc).
- The comparator will only be active if the associated channel is being sampled by Seq 1 and/or Seq 2.
- Each comparator can be programmed to trigger a status based on a sampled value being "greater than" or "less than" a 12-bit value loaded into the associated control register. The status can assert an IRQ if enabled.
- Each comparator can be individually disabled by setting its associated channel field to a non-valid channel number.

#### 17.5.4.2 Battery Voltage Monitor Comparators (ADC_BAT_COMP_OVER and ADC_BAT_COMP_UNDER Registers)

In addition to the comparators for the external analog sample channels, there are two comparators for an internal sample channel designated as the Battery Reference Voltage channel, which is a fixed 1.2V voltage nominal. These comparators provide a simple way to monitor the supply voltage (VBATT):

1. Battery Voltage Upper Trip Point Register (ADC_BAT_COMP_OVER) - a compare flag is set if the sampled data value on the fixed voltage 1.2V channel exceeds the value contained in this register.
2. Battery Voltage Lower Trip Point Register (ADC_BAT_COMP_UNDER) - a compare flag is set if the sampled data value on the fixed voltage 1.2V channel is lower than the value contained in this register.

Use of these comparators is essentially the "inverse" of normal ADC operation. The fixed 1.2V (battery reference) is sampled via ADC_1 with the ADC_1 Vrefh selected as the internal reference VBATT (AD1_Vrefhl_En = 0). With a fixed, known input voltage, the ADC_1 reading (VALUE) can be equated to 1.2V and the full scale (0xFFF) of the ADC_1 range equated to VBATT voltage. As a result, using a ratio provides the VBATT supply voltage:

0xFFF / VALUE = VBATT / 1.2V

Given the relationship in the ratio, it should be considered when using these registers:

1. The "Upper Trip Point" is really a measure of DECREASING supply voltage. As the 1.2V sample value increases, the Vrefh (VBATT) decreases.
2. The "Lower Trip Point" is really a measure of INCREASING supply voltage. As the 1.2V sample value decreases, the Vrefh (VBATT) increases.

These registers are useful for monitoring battery life, controlling battery recharge, or controlling use of the onboard Buck Regulator.

#### 17.5.4.3 Comparators Status (ADC_TRIGGERS Register)

Activity of the comparators can be monitored in the Triggered Comparator Status Register (ADC_TRIGGERS).

- If a given comparator triggers, an associated status bit in this register is set to reflect the channel/comparator. This status can be used to determine which channel(s) have triggered when servicing and interrupt request.
- If a status bit is set, the **Compare** Bit in the ADC IRQ Status Register (ADC_IRQ) is also set.
- The Compare status can be enabled to generate an interrupt request.

### 17.5.5 ADC Interrupt Requests

The ADC module has a single interrupt request signal that is connected to the interrupt controller. The ADC IRQ can be generated from four primary sources which are logically OR'ed such that any can generate the interrupt request if enabled.

Table 17-2 lists the ADC interrupt sources and their characteristics.

**Table 17-2. ADC Interrupt Sources**

| Item | Status Bit (ADC_IRQ Reg) | Mask Bit (ADC_CONTROL Reg) | Source Description | Interrupt Clear Mechanism |
|---|---|---|---|---|
| 1 | FIFO | FIFO_IRQ_Mask | FIFO contents have filled to the preset level | Writing a "1" to the FIFO bit clears the status. |
| 2 | Seq2 | Seq2_IRQ_Mask | Sequencer 2 has completed a cycle through its active selected channels | Writing a "1" to the Seq2 bit clears the status. |
| 3 | Seq1 | Seq1_IRQ_Mask | Sequencer 1 has completed a cycle through its active selected channels | Writing a "1" to the Seq1 bit clears the status. |
| 4 | Compare | Compare_IRQ_Mask | One or more comparators has triggered.<br>• Monitor the ADC_TRIGGERS Register to view triggered channels<br>• Valid only when using sequencers | • Writing a "1" to the Compare bit clears the status.<br>• Reading the ADC_TRIGGERS Register clears all bits |

## 17.6 ADC Register Memory Map

The ADC module is programmed via a set of memory-mapped registers and their respective offset addresses are listed in Table 17-3.

- The ADC base address is **0x8000_D000**
- All registers are 16 bits wide with 16-bit access only. Note register address offset is 0x2, not 0x4.

**Table 17-3. ADC Module Registers Memory Map**

| Address | Register Name | Access Type | Access Width |
|---|---|---|---|
| Base + 0x00 | ADC Compare 0 Register (ADC_COMP_0) | R/W | 16-bit only |
| Base + 0x02 | ADC Compare 1 Register (ADC_COMP_1) | R/W | 16-bit only |
| Base + 0x04 | ADC Compare 2 Register (ADC_COMP_2) | R/W | 16-bit only |
| Base + 0x06 | ADC Compare 3 Register (ADC_COMP_3) | R/W | 16-bit only |

**Table 17-3. ADC Module Registers Memory Map**

| | | | |
|---|---|---|---|
| Base + 0x08 | ADC Compare 4 Register (ADC_COMP_4) | R/W | 16-bit only |
| Base + 0x0A | ADC Compare 5 Register (ADC_COMP_5) | R/W | 16-bit only |
| Base + 0x0C | ADC Compare 6 Register (ADC_COMP_6) | R/W | 16-bit only |
| Base + 0x0E | ADC Compare 7 Register (ADC_COMP_7) | R/W | 16-bit only |
| Base + 0x10 | Battery Voltage Upper Trip Point Register (ADC_BAT_COMP_OVER) | R/W | 16-bit only |
| Base + 0x12 | Battery Voltage Lower Trip Point Register (ADC_BAT_COMP_UNDER) | R/W | 16-bit only |
| Base + 0x14 | Sequencer 1 Mode Control Register (ADC_SEQ_1) | R/W | 16-bit only |
| Base + 0x16 | Sequencer 2 Mode Control Register (ADC_SEQ_2) | R/W | 16-bit only |
| Base + 0x18 | ADC Module Primary Control Register (ADC_CONTROL) | R/W | 16-bit only |
| Base + 0x1A | Triggered Comparator Status Register (ADC_TRIGGERS) | R | 16-bit only |
| Base + 0x1C | Prescale Clock Register (ADC_PRESCALE) | R/W | 16-bit only |
| Base + 0x1E | Reserved | | 16-bit only |
| Base + 0x20 | ADC FIFO Read Register (ADC_FIFO_READ) | R | 16-bit only |
| Base + 0x22 | ADC FIFO Control Register (ADC_FIFO_CONTROL) | R/W | 16-bit only |
| Base + 0x24 | ADC FIFO Status Register (ADC_FIFO_STATUS) | R | 16-bit only |
| Base + 0x26 to 0x2E | Reserved | | |
| Base + 0x30 | ADC_1 Sample Rate High Register (ADC_SR_1_HIGH) | R/W | 16-bit only |
| Base + 0x32 | ADC_1 Sample Rate Low Register (ADC_SR_1_LOW) | R/W | 16-bit only |
| Base + 0x34 | ADC_2 Sample Rate High Register (ADC_SR_2_HIGH) | R/W | 16-bit only |
| Base + 0x36 | ADC_2 Sample Rate Low Register (ADC_SR_2_LOW) | R/W | 16-bit only |
| Base + 0x38 | ADC Turn-On Time Register (ADC_ON_TIME) | R/W | 16-bit only |
| Base + 0x3a | ADC Convert Time Register (ADC_CONVERT_TIME) | R/W | 16-bit only |
| Base + 0x3c | ADC Clock Divider Register (ADC_CLOCK_DIVIDER) | R/W | 16-bit only |
| Base + 0x3e | Reserved | | |
| Base + 0x40 | ADC Manual Control Register (ADC_OVERRIDE) | R/W | 16-bit only |
| Base + 0x42 | ADC IRQ Status Register (ADC_IRQ) | R/W | 16-bit only |
| Base + 0x44 | ADC Mode Register (ADC_MODE) | R/W | 16-bit only |
| Base + 0x46 | ADC_1 Result Register (ADC_1_RESULT) | R | 16-bit only |
| Base + 0x48 | ADC 2 Result Register (ADC_2_RESULT) | R | 16-bit only |

## 17.7 ADC Registers

The following sections provide descriptions of the ADC registers.

### 17.7.1 ADC Compare Registers (ADC_COMP_0:ADC_COMP_7)

There are eight ADC Compare Register that can be used to monitor the eight external analog input channels. See Table 17-4 for the register names, associated ADC analog channel, and address.

**Table 17-4. ADC Compare Register Designations and Addresses**

| Compare Register Name | Associated ADC Analog Channel | Address |
|---|---|---|
| ADC Compare 0 Register (ADC_COMP_0) | ADC0 | Base+0x00 |
| ADC Compare 1 Register (ADC_COMP_1) | ADC1 | Base+0x02 |
| ADC Compare 2 Register (ADC_COMP_2) | ADC2 | Base+0x04 |
| ADC Compare 3 Register (ADC_COMP_3) | ADC3 | Base+0x06 |
| ADC Compare 4 Register (ADC_COMP_4) | ADC4 | Base+0x08 |
| ADC Compare 5 Register (ADC_COMP_5) | ADC5 | Base+0x0A |
| ADC Compare 6 Register (ADC_COMP_6) | ADC6 | Base+0x0C |
| ADC Compare 7 Register (ADC_COMP_7) | ADC7 | Base+0x0E |

These registers all have a similar model and are used in the same manner:

- The comparator can only be used with its associated analog channel
- Each register contains “greater/less than” (compare type), channel number, and compare value fields that determine comparator operation
- The **Channel** field in the register defaults to 0x0 and must be programmed to the matching channel number for the comparator to be operative

| ADC_COMP_X | | | | | | | | | | | | | Addr Base+0xXX | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | GL | Channel | | | Value | | | | | | | | | | | |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 17-5. ADC Compare Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 15 | GL - Greater / Less Than. This bit determines if the comparator tests for a sampled data to be greater than or less than the Value field of the register. | 1 = If (value on channel < value), trigger a compare flag<br>0 (default) = If (value on channel > value), trigger a compare flag |
| 14-12 | Channel[2:0] - ADC Input Channel Number. This field must match the associated channel to trigger the compare function. | 0x0 = ADC0 (Channel 0); default<br>. . .<br>0x7 = ADC7 (Channel 7) |
| 11-0 | **Value[11:0] -** Compare Value. This field is used as the compare threshold for the channel | |

## 17.7.2 Battery Voltage Upper Trip Point Register (ADC_BAT_COMP_OVER)

The Battery Voltage Upper Trip Point Register (ADC_BAT_COMP_OVER) contains the value that is compared with sampled data from the Battery Reference Voltage channel to set the upper trip point, i.e., a compare flag is set if the sampled data value exceeds this value.

| ADC_BAT_COMP_OVER | | | | | | | | | | | | | Addr Base+0x10 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | Reserved | | | | Value | | | | | | | | | | | |
| TYPE | r | r | r | r | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 17-6. ADC_BAT_COMP_OVER Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 15-12 | Reserved | |
| 11-0 | **Value[11:0] -** Compare value. This field is used as the compare threshold for the upper battery reference voltage trip point. | |

### 17.7.3 Battery Voltage Lower Trip Point Register (ADC_BAT_COMP_UNDER)

The Battery Voltage Lower Trip Point Register (ADC_BAT_COMP_UNDER) contains the value that is compared with sampled data from the Battery Reference Voltage channel to set the lower trip point, i.e., a compare flag is set if the sampled data value is lower than this value.

| ADC_BAT_COM_UNDER | | | | | | | | | | | | | Addr Base+0x12 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | Reserved | | | | Value | | | | | | | | | | | |
| TYPE | r | r | r | r | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 17-7. ADC_BAT_COMP_UNDER Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 15-12 | Reserved | |
| 11-0 | **Value[11:0] -** Compare value. This field is used as the compare threshold for the lower battery reference voltage trip point. | |

### 17.7.4 Sequencer 1 Mode Control Register (ADC_SEQ_1)

The Sequencer 1 Mode Control Register (ADC_SEQ_1) configures the operation of Sequencer 1 (ADC_1) when the sequencer is active:

- ADC module is On - Register ADC_CONTROL[0] = 1
- The ADC module is NOT in soft reset - Register ADC_CONTROL[3] = 0
- The override mode is NOT active - Register ADC_MODE[0] = 0
- If the **Seq_Mode** control bit is set for Timer 1 events -
  - Timer 1 must be enabled - Register ADC_CONTROL[1] = 1
  - Timing parameters must be pre-programmed

| ADC_SEQ_1 | | | | | | | | | | | | | Addr Base+0x14 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | Seq_ Mode | Reserved | | | | | | Batt | Ch7 | Ch6 | Ch5 | Ch4 | Ch3 | Ch2 | Ch1 | Ch0 |
| TYPE | rw | r | r | r | r | r | r | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 17-8. ADC_SEQ_1 Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 15 | Seq_Mode- Sequence Mode. This bit determines the mode through which Sequencer 1 will advance through active channels.<br>**Note:** If Timer 1 mode is selected, Timer 1 must be configured and enabled | 1 = Sequence to next channel based on Timer 1 event.<br>0 = Sequence to next channel based on convert time (typically 20µs); default |
| 14-9 | Reserved | |
| 8 | **Batt -** Battery Reference Voltage Channel Enable. Available on ADC_1. | 1 = Channel Enabled<br>0 = Disabled (default) |
| 7 | **Ch7 -** ADC Input Channel 7 Enable. | 1 = Channel Enabled<br>0 = Disabled (default) |
| 6 | **Ch6-** ADC Analog Input Channel 6 Enable. | 1 = Channel Enabled<br>0 = Disabled (default) |
| 5 | **Ch5 -** ADC Analog Input Channel 5 Enable. | 1 = Channel Enabled<br>0 = Disabled (default) |
| 4 | **Ch4 -** ADC Analog Input Channel 4 Enable. | 1 = Channel Enabled<br>0 = Disabled (default) |
| 3 | **Ch3 -** ADC Analog Input Channel 3 Enable. | 1 = Channel Enabled<br>0 = Disabled (default) |
| 2 | **Ch2 -** Adenoidal Input Channel 2 Enable. | 1 = Channel Enabled<br>0 = Disabled (default) |
| 1 | **Ch1 -** ADC Analog Input Channel 1 Enable. | 1 = Channel Enabled<br>0 = Disabled (default) |
| 0 | **Ch0 -** ADC Analog Input Channel 0 Enable. | 1 = Channel Enabled<br>0 = Disabled (default) |

## 17.7.5 Sequencer 2 Mode Control Register (ADC_SEQ_2)

The Sequencer 2 Mode Control Register (ADC_SEQ_2) configures the operation of Sequencer 2 (ADC_2) when the sequencer is active:

- ADC module is On - Register ADC_CONTROL[0] = 1
- The ADC module is NOT in soft reset - Register ADC_CONTROL[3] = 0
- The override mode is NOT active - Register ADC_MODE[0] = 0
- If the **Seq_Mode** control bit is set for Timer 2 events -
  - Timer 2must be enabled - Register ADC_CONTROL[2] = 1
  - Timing parameters must be pre-programmed.

| ADC_SEQ_2 | | | | | | | | | | | | | Addr Base+0x16 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | Seq_ Mode | Reserved | | | | | | | Ch7 | Ch6 | Ch5 | Ch4 | Ch3 | Ch2 | Ch1 | Ch0 |
| TYPE | rw | r | r | r | r | r | r | r | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 17-9. ADC_SEQ_2 Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 15 | Seq_Mode- Sequence Mode. This bit determines the mode through which Sequencer 2 will advance through active channels.<br>**Note:** If Timer 1 mode is selected, Timer 1 must be configured and enabled | 1 = Sequence to next channel based on Timer 2 event.<br>0 = Sequence to next channel based on convert time (typically 20µs); default |
| 14-8 | Reserved | |
| 7 | **Ch7 -** ADC Input Channel 7 Enable. | 1 = Channel Enabled<br>0 = Disabled (default) |
| 6 | **Ch6-** ADC Analog Input Channel 6 Enable. | 1 = Channel Enabled<br>0 = Disabled (default) |
| 5 | **Ch5 -** ADC Analog Input Channel 5 Enable. | 1 = Channel Enabled<br>0 = Disabled (default) |
| 4 | **Ch4 -** ADC Analog Input Channel 4 Enable. | 1 = Channel Enabled<br>0 = Disabled (default) |
| 3 | **Ch3 -** ADC Analog Input Channel 3 Enable. | 1 = Channel Enabled<br>0 = Disabled (default) |
| 2 | **Ch2 -** Adenoidal Input Channel 2 Enable. | 1 = Channel Enabled<br>0 = Disabled (default) |
| 1 | **Ch1 -** ADC Analog Input Channel 1 Enable. | 1 = Channel Enabled<br>0 = Disabled (default) |
| 0 | **Ch0 -** ADC Analog Input Channel 0 Enable. | 1 = Channel Enabled<br>0 = Disabled (default) |

## 17.7.6 ADC Module Primary Control Register (ADC_CONTROL)

The ADC Module Primary Control Register (ADC_CONTROL) provides the highest level of control for the ADC module.

| ADC_CONTROL | | | | | | | | | | | | | Addr Base+0x18 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | FIFO_ IRQ_ Mask | Seq2_ IRQ_ Mask | Seq1_ IRQ_ Mask | Comp are_ IRQ_ Mask | Reserved | | | | | | AD2_ Vrefhl _En | AD1_ Vrefhl _en | Soft_ Reset | Timer 2_On | Timer 1_On | On |
| TYPE | rw | rw | rw | rw | r | r | r | r | r | r | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |

**Table 17-10. ADC_CONTROL Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 15 | FIFO_IRQ_Mask - FIFO IRQ Mask. This bit enables an IRQ generated by the FIFO status of the ADC_IRQ Register. | 1 = FIFO IRQ enabled<br>0 = IRQ masked/disabled (default) |
| 14 | Seq2_IRQ_Mask - Sequencer 2 IRQ Mask. This bit enables an IRQ generated by the Seq2 status of the ADC_IRQ Register. | 1 = Seq2 IRQ enabled<br>0 = IRQ masked/disabled (default) |
| 13 | **Seq1_IRQ_Mask -** Sequencer 1 IRQ Mask. This bit enables an IRQ generated by the Seq1 status of the ADC_IRQ Register. | 1 = Seq1 IRQ enabled<br>0 = IRQ masked/disabled (default) |
| 12 | **Compare_IRQ_Mask -** Compare IRQ Mask. This bit enables an IRQ generated by the Compare status of the ADC_IRQ Register. | 1 = Compare IRQ enabled<br>0 = IRQ masked/disabled (default) |
| 11-6 | Reserved | |
| 5 | AD2_Vrefhl_En - ADC_2 Vref Voltage Select. The ADC_2 reference voltages can be supplied from either:<br>• The external pins ADC2_VREFH and ADC2_VREFL<br>• Internal connection to VBATT and VSS (GND) | 1 = External reference selected (default)<br>0 = Internal reference selected |
| 4 | AD1_Vrefhl_En - ADC_1 Vref Voltage Select. The ADC_1 reference voltages can be supplied from either:<br>• The external pins ADC1_VREFH and ADC1_VREFL<br>Internal connection to VBATT and VSS (GND) | 1 = External reference selected (default)<br>0 = Internal reference selected |
| 3 | **Soft_Reset -** Soft Reset Enable. This bit enables and holds the ADC module in reset | 1 = Soft reset enabled (and held)<br>0 = Reset released (default) |

**Table 17-10. ADC_CONTROL Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 2 | **Timer2_On -** Timer 2 Enable. This bit enable the 32-bit Timer 2 counter.<br>**Note:** This bit should be enabled if Seq 2 is programmed for Timer mode | 1 = Timer 2 enabled<br>0 = Timer 2 disabled (default) |
| 1 | **Timer1_On -** Timer 1 Enable. This bit enable the 32-bit Timer 1counter.<br>**Note:** This bit should be enabled if Seq 1 is programmed for Timer mode | 1 = Timer 1 enabled<br>0 = Timer 1disabled (default) |
| 0 | **On -** ADC Module Enable. This bit is master enable for the ADC | 1 = ADC enabled<br>0 = ADC disabled (default) |

## 17.7.7 Triggered Comparator Status Register (ADC_TRIGGERS)

The read-only Triggered Comparator Status Register (ADC_CONTROL) displays which comparators are triggered.

- These bits get set when the associated comparator gets a valid compare for the programmed test condition
- Setting a bit in this register also sets the COMP Status
- These bits get cleared with a register read

| ADC_TRIGGERS | | | | | | | | | | | | | Addr Base+0x1A | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | Reserved | | | | | | Bat_ Over | Bat_ Under | Cmp7 | Cmp6 | Cmp5 | Cmp4 | Cmp3 | Cmp2 | Cmp1 | Cmp0 |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 17-11. ADC_TRIGGERS Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 15-10 | Reserved | |
| 9 | Bat_Over - ADC_BAT_COMP_OVER Comparator Status. | 1 = ADC_BAT_COMP_OVER comparator has triggered due to a valid compare<br>0 = Comparator not triggered (default) |
| 8 | **Bat_Under -** ADC_BAT_COMP_Under Comparator Status | 1 = ADC_BAT_COMP_UNDER comparator has triggered due to a valid compare<br>0 = Comparator not triggered (default) |
| 7 | **Ch7 -** Comparator 7 Status. | 1 = ADC_COMP_7 has triggered due to a valid compare<br>0 = Comparator not triggered (default) |

**Table 17-11. ADC_TRIGGERS Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 6 | **Ch6 -** Comparator 6 Status. | 1 = ADC_COMP_6 has triggered due to a valid compare<br>0 = Comparator not triggered (default) |
| 5 | **Ch5 -** Comparator 5 Status. | 1 = ADC_COMP_5 has triggered due to a valid compare<br>0 = Comparator not triggered (default) |
| 4 | **Ch4 -** Comparator 4 Status. | 1 = ADC_COMP_4 has triggered due to a valid compare<br>0 = Comparator not triggered (default) |
| 3 | **Ch3 -** Comparator 3 Status. | 1 = ADC_COMP_3 has triggered due to a valid compare<br>0 = Comparator not triggered (default) |
| 2 | **Ch2 -** Comparator 2 Status. | 1 = ADC_COMP_2 has triggered due to a valid compare<br>0 = Comparator not triggered (default) |
| 1 | **Ch1 -** Comparator 1 Status. | 1 = ADC_COMP_1 has triggered due to a valid compare<br>0 = Comparator not triggered (default) |
| 0 | **Ch0 -** Comparator 0 Status. | 1 = ADC_COMP_0 has triggered due to a valid compare<br>0 = Comparator not triggered (default) |

## 17.7.8 Prescale Clock Register (ADC_PRESCALE)

The Prescale Clock Register (ADC_PRESCALE) sets the divide ratio for the counter that prescalers the peripheral clock to provide the ADC Prescale Clock. This clock that drives Timer 1 and Timer 2 and the sequencers, and is also used to generate the ON_TIME and CONVERT_TIME.

- Typical desired ADC Prescale Clock frequency is 1MHz
- The PRESCALE value is an 8-bit integer that sets the prescale number N for the prescale clock where N = value + 1.

Prescale Clock = Peripheral Clock / (PRESCALE[7:0] + 1)

| ADC_PRESCALE | | | | | | | | | | | | | Addr Base+0x1c | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | Reserved | | | | | | | | Prescale | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 17-12. ADC_PRESCALE Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 15-8 | Reserved | |
| 7-0 | Prescale[7:0] - Prescale Clock Divide Ratio. This field sets the prescale divide ratio N where: N = Prescale[7:0] + 1 | Default = 0x00 |

## 17.7.9 ADC FIFO Read Register (ADC_FIFO_READ)

The ADC FIFO Read Register (ADC_FIFO_READ) is a 16-bit read-only register for accessing the contents of the ADC FIFO:

- The FIFO data is loaded as the result of Sequencer 1 and Sequencer 2 activity only.
- No order of the data is implied if Seq 1 and Seq 2 data are mixed.
- Each read of this register updates the FIFO pointers and returns the sampled data value stored in the FIFO (along with its channel tag)

| ADC_FIFO_READ | | | | | | | | | | | | | Addr Base+0x20 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Bit 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | Channel | | | | Value | | | | | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 17-13. ADC_FIFO_READ Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 15-12 | Channel[3:0] - Analog Channel. This field is a tag to the analog channel from which the data were sampled. See Table 17-14 for the mapping of channel number versus value. | |
| 7-0 | Value[11:0] - Sampled Data Value. This field of the value of the sampled data. An unsigned 12-bit binary number where 0xFFF is full range. | |

**Table 17-14. Channel Number for Analog Input Channel**

| Channel[3:0] | Analog Input Channel |
|---|---|
| 0000 | ADC0 |
| 0001 | ADC1 |
| 0010 | ADC2 |
| 0011 | ADC3 |
| 0100 | ADC4 |
| 0101 | ADC5 |
| 0110 | ADC6 |

**Table 17-14. Channel Number for Analog Input Channel**

| Channel[3:0] | Analog Input Channel |
|---|---|
| 0000 | ADC0 |
| 0111 | ADC7 |
| 1000 | Battery Reference Voltage |

## 17.7.10 ADC FIFO Control Register (ADC_FIFO_CONTROL)

The ADC FIFO Control Register (ADC_FIFO_Control) sets the level at which the FIFO contents will generate an IRQ. The FIFO is 8 words deep, and once the FIFO data has reach the programmed level, a FIFO status will be set in the ADC_IRQ Register and a FIFO IRQ will be generated if enabled.

| ADC_FIFO_CONTROL | | | | | | | | | | | | | Addr Base+0x22 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | Reserved | | | | | | | | | | | | Level | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 17-15. ADC_FIFO_CONTROL Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 15-4 | Reserved | |
| 3-0 | Level[3:0] - FIFO IRQ Level. When the FIFO data reaches this level, a FIFO status will be set.<br>**Note:** Only values 0x1 to 0x8 are valid. | |

## 17.7.11 ADC FIFO Status Register (ADC_FIFO_STATUS)

The ADC FIFO Status Register (ADC_FIFO_STATUS) reflects the current status (capacity) of the FIFO.

| adc_fifo_status | | | | | | | | | | | | | Addr Base+0x24 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | Reserved | | | | | | | | | | Empty | Full | Level | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 17-16. ADC_FIFO_STATUS Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 15-6 | Reserved | |
| 5 | Empty - FIFO Empty Status. This bit indicates FIFO empty condition | 1 = FIFO is empty<br>0 = FIFO has data |
| 4 | Full - FIFO Full Status. This bit indicates FIFO full condition. If FIFO is full additional writes to the FIFO are disabled. | 1 = FIFO is full<br>0 = FIFO is not full |
| 3-0 | Level[3:0] - FIFO Level. This field reflects the current level of words in FIFO. | |

## 17.7.12 ADC_1 Sample Rate High Register (ADC_SR_1_HIGH)

The ADC_1 Sample Rate High Register (ADC_SR_1_HIGH) contains the upper 16 bits of the Timer 1 compare value used by the ADC_1 sequencer.

- Timer 1 uses the Prescale Clock as the input clock
- Timer 1 is a 32-bit counter with ADC_SR_1_LOW as the lower 16-bit compare value
- When Timer 1 is used by the ADC_1 sequencer, it must be enabled

| ADC_SR_1_HIGH | | | | | | | | | | | | | Addr Base+0x30 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | Sample Rate 1 High | | | | | | | | | | | | | | | |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 17-17. ADC_SR_1_HIGH Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 15-0 | Sample Rate 1 High[15:0] - This field is the upper 16 bits of the Timer 1 compare value used to establish the ADC_1 sequencer sample rate | |

## 17.7.13 ADC_1 Sample Rate Low Register (ADC_SR_1_LOW)

The ADC_1 Sample Rate Low Register (ADC_SR_1_LOW) contains the lower 16 bits of the Timer 1 compare value used by the ADC_1 sequencer.

- Timer 1 uses the Prescale Clock as the input clock
- Timer 1 is a 32-bit counter with ADC_SR_1_HIGH as the upper 16-bit compare value
- When Timer 1 is used by the ADC_1 sequencer, it must be enabled

| ADC_SR_1_LOW | | | | | | | | | | | | | | Addr Base+0x32 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | Sample Rate 1 Low | | | | | | | | | | | | | | | |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 17-18. ADC_SR_1_LOW Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 15-0 | Sample Rate 1 Low[15:0] - This field is the lower 16 bits of the Timer 1 compare value used to establish the ADC_1 sequencer sample rate | |

## 17.7.14 ADC_2 Sample Rate High Register (ADC_SR_2_HIGH)

The ADC_2 Sample Rate High Register (ADC_SR_2_HIGH) contains the upper 16 bits of the Timer 2 compare value used by the ADC_2 sequencer.

- Timer 2 uses the Prescale Clock as the input clock
- Timer 2 is a 32-bit counter with ADC_SR_2_LOW as the lower 16-bit compare value
- When Timer 2 is used by the ADC_2 sequencer, it must be enabled

| ADC_SR_2_HIGH | | | | | | | | | | | | | | Addr Base+0x34 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | Sample Rate 2 High | | | | | | | | | | | | | | | |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 17-19. ADC_SR_2_HIGH Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 15-0 | Sample Rate 2 High[15:0] - This field is the upper 16 bits of the Timer 2 compare value used to establish the ADC_2 sequencer sample rate | |

## 17.7.15 ADC_2 Sample Rate Low Register (ADC_SR_2_LOW)

The ADC_2 Sample Rate Low Register (ADC_SR_2_LOW) contains the lower 16 bits of the Timer 2 compare value used by the ADC_2 sequencer.

- Timer 2 uses the Prescale Clock as the input clock
- Timer 2 is a 32-bit counter with ADC_SR_2_HIGH as the upper 16-bit compare value
- When Timer 2 is used by the ADC_2 sequencer, it must be enabled

| ADC_SR_2_LOW | | | | | | | | | | | | | | Addr Base+0x36 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | Sample Rate 2 Low | | | | | | | | | | | | | | | |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 17-20. ADC_SR_2_LOW Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 15-0 | Sample Rate 2 Low[15:0] - This field is the lower 16 bits of the Timer 2 compare value used to establish the ADC_2 sequencer sample rate | |

## 17.7.16 ADC Turn-On Time Register (ADC_ON_TIME)

The ADC Turn-On Time Register (ADC_ON_TIME) determines the turn-on time of the ADC analog block before initializing an A-to-D conversion.

- This delay is a function of the Prescale Clock (typically 1 MHz)
- The register must be initialized for proper operation
- The ON-TIME must always be 10µs or greater - typical On-Time value = 0x000A (10dec)

| ADC_ON_TIME | | | | | | | | | | | | | | Addr Base+0x38 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | On-Time | | | | | | | | | | | | | | | |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 17-21. ADC_ON_TIME Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 15-0 | On-Time[15:0] - This field sets the turn-on time delay by its number of Prescale Clock periods. | 0x0000 = Default |

## 17.7.17 ADC Convert Time Register (ADC_CONVERT_TIME)

The ADC Convert Time Register (ADC_CONVERT_TIME) determines the amount of wait time allowed for an A-to-D conversion.

**NOTE**

The ADC analog block requires 6 ADC_Clocks per conversion, and the ADC_Clock must 300kHz or less. With 6 clocks/conversion and a 33.33µs clock period:

- The ADC convert time must always be 20µs or greater
- If the ADC_Clock is a frequency lower than 300kHz, the convert time must always be 6 ADC_Clock periods or greater
- For override mode, extend convert time to 40µs minimum or greater

For the convert time:

- This delay is a function of the Prescale Clock (typically 1 MHz)
- The register must be initialized for proper operation
- For a 20µs convert time with 1MHz, value = 0x0014 (20dec)
- If convert time is insufficient, inaccurate sample data will result

| ADC_CONVERT_TIME | | | | | | | | | | | | | Addr Base+0x3A | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | Convert-Time | | | | | | | | | | | | | | | |
| TYPE | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 17-22. ADC_CONVERT_TIME Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 15-0 | Convert-Time[15:0] - This field sets the convert time delay by its number of Prescale Clock periods. | 0x0000 = Default |

## 17.7.18 ADC Clock Divider Register (ADC_CLOCK_DIVIDER)

The ADC Clock Divider Register (ADC_CLOCK_DIVIDER) determines the frequency of the ADC analog clock (ADC_Clock). The ADC_Clock is generated from the peripheral clock

- ADC_Clock must be less than or equal to 300kHz
- ADC_CLOCK_DIVIDER Register - contains a 16-bit values that sets the divide value of the ADC_Clock divider:
- ADC_Clock = Peripheral Clock / ADC_CLOCK_DIVIDER

As an example for 24MHz peripheral clock:

300kHz = 24Mhz / 80

Set ADC_CLOCK_DIVIDER = 0x0050 (80dec)

| ADC_CLOCK_DIVIDER | | | | | | | | | | | | | Addr Base+0x3c | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | Reserved | | | | | | | | Clock Divider | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 17-23. ADC_CLOCK_DIVIDER Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 15-8 | Reserved | |
| 7-0 | Clock_Divider[7:0] - This field sets the ADC_Clock frequency:<br>ADC_Clock = Peripheral Clock / Clock_Divider[7:0] | 0x0000 = Default |

## 17.7.19 ADC Manual Control Register (ADC_OVERRIDE)

The ADC Manual Control Register (ADC_OVERRIDE) controls operation of the ADC module if the automated mode(s) are disabled.

**NOTE**

For the ADC_OVERIDE register to be active, the Override bit must be set in the ADC_MODE Register.

| ADC_OVERRIDE | | | | | | | | | | | | | Addr Base+0x40 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | Reserved | | | | | | AD2_ On | AD1_ On | Mux2 | | | | Mux1 | | | |
| TYPE | r | r | r | r | r | r | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 17-24. ADC_TRIGGERS Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 15-10 | Reserved | |
| 9 | AD2_On - ADC_2 On. This bit activates ADC_2. | 1 = ADC_2 on<br>0 = ADC_2 off (default) |
| 8 | **AD1_On -** ADC_1 On. This bit activates ADC_1. | 1 = ADC_1 on<br>0 = ADC_1 off (default) |

**Table 17-24. ADC_TRIGGERS Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 7-4 | **Mux2[3:0] -** ADC_2 Analog Mux Select. This field selects the analog channel for A-to-D conversion. See Table 17-25. | 0x0 = Default |
| 3-0 | **Mux1[3:0] -** ADC_1 Analog Mux Select. This field selects the analog channel for A-to-D conversion. See Table 17-26. | 0x0 = Default |

**Table 17-25. ADC_2 MUX2 Analog Input Channel Select**

| MUX2 value | Selected Analog Channel |
|---|---|
| 0000 | ADC0 |
| 0001 | ADC1 |
| 0010 | ADC2 |
| 0011 | ADC3 |
| 0100 | ADC4 |
| 0101 | ADC5 |
| 0110 | ADC6 |
| 0111 | ADC7 |

**Table 17-26. ADC_1 MUX1 Analog Input Channel Select**

| MUX1 value | Selected Analog Channel |
|---|---|
| 0000 | ADC0 |
| 0001 | ADC1 |
| 0010 | ADC2 |
| 0011 | ADC3 |
| 0100 | ADC4 |
| 0101 | ADC5 |
| 0110 | ADC6 |
| 0111 | ADC7 |
| 1000 | Battery Reference Voltage |

## 17.7.20 ADC IRQ Status Register (ADC_IRQ)

The ADC IPQ Status Register (ADC_IRQ) can be read to determine the active status that can generate an IRQ, if enabled. A write of 1 to the respective bit clears the IRQ source status.

**NOTE**

- If using Compare IRQ, read the ADC_TRIGGERS register for information on which channel generated the compare IRQ.
- Read the ADC_FIFO_STATUS register if using the FIFO IRQ.

| ADC_IRQ | | | | | | | | | | | | | Addr Base+0x42 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | FIFO | Seq2 | Seq1 | Com-pare | Reserved | | | | | | | | | | | |
| TYPE | rw | rw | rw | rw | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 17-27. ADC_IRQ Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 15 | FIFO - FIFO Status. This bit indicates when FIFO contents have reached preset level.<br>**Note:** Read ADC_FIFO_STATUS Register for status. | 1 = FIFO contents have reached preset level<br>0 = FIFO contents below preset level |
| 14 | Seq2 - Sequencer 2 Status. This bit indicates that Sequencer 2 has completed a cycle. | 1 = Seq 2 has completed a conversion cycle of all enabled channels<br>0 = Seq 2 has not recycled |
| 13 | **Seq1 -** Sequencer 1 Status. This bit indicates that Sequencer 1has completed a cycle. | 1 = Seq 1has completed a conversion cycle of all enabled channels<br>0 = Seq 1has not recycled |
| 12 | **Compare -** Comparator Status. This bit indicates that one or more comparators has triggered based on a compare condition.<br>**Note:** Read ADC_TRIGGERS Register for channel status. | 1 = Comparator(s) have triggered due to a compare condition met<br>0 = No new compare condition |
| 11-0 | Reserved | |

## 17.7.21 ADC Mode Register (ADC_MODE)

The ADC Mode Register (ADC_MODE) is used to enable override or manual mode.

| ADC_MODE | | | | | | | | | | | | | Addr Base+0x44 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | Reserved | | | | | | | | | | | | | | | Over-ride |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | rw |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 17-28. ADC_MODE Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 15-1 | Reserved | |
| 0 | Override - Override Control. Setting this bit disables automated mode and enables manual (override) mode. | 1 = Override mode.<br>0 = Automated mode (default) |

## 17.7.22 ADC_1 Result Register (ADC_1_RESULT)

The read-only ADC_1 Result Register (ADC_1_RESULT) returns the sampled data result of an ADC_1 conversion while in override mode.

| ADC_1_RESULT | | | | | | | | | | | | | Addr Base+0x46 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | Reserved | | | AD1_Result | | | | | | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 17-29. ADC_MODE Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 15-12 | Reserved | |
| 11-0 | AD1_Result[11:0] - ADC_1 Conversion Result. Returns a 12-bit unsigned binary number where 0xFFF is full scale. | |

## 17.7.23 ADC 2 Result Register (ADC_2_RESULT)

The read-only ADC_2 Result Register (ADC_2_RESULT) returns the sampled data result of an ADC_2 conversion while in override.

| ADC_2_RESULT | | | | | | | | | | | | | Addr Base+0x48 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BIT 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | BIT 0 |
| | Reserved | | | | AD2_Result | | | | | | | | | | | |
| TYPE | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r | r |
| RESET | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 17-30. ADC_MODE Register Bit Descriptions**

| Bit Number | Description | Operation |
|---|---|---|
| 15-12 | Reserved | |
| 11-0 | AD2_Result[11:0] - ADC_2 Conversion Result. Returns a 12-bit unsigned binary number where 0xFFF is full scale. | |

# Chapter 18
# Development and Debug Support

The MC1322x family is supported by a full set of hardware/software evaluation and development tools.

## 18.1 Hardware Development Interfaces

The ARM debug environment supports both a JTAG debug interface and an extended capability Nexus interface.

### 18.1.1 JTAG Hardware Debug Port

The JTAG port is the simpler and more common debug port for the ARM core. A standard 20-pin connector as described in Section 2.3.1, "ARM JTAG Interface Connector", is connected to the TDI, TMS, TCK, TDO, and RTCK signals of the MC1322x. Through the JTAG serial interface, standard debug and development activities such as accessing memory and registers, control of the CPU, download of FLASH memory, and software debug can be accomplished.

**NOTE**

Although designated as a JTAG port, the MC1322x JTAG debug port does not support and cannot be used for true JTAG system boundary scan testing.

### 18.1.2 A7S Nexus3 (NEX) ARM7 Core Development Interface

The development and debug environment of the ARM7TDMI-S core is based on the A7S Nexus3 interface (compliant with a Class 3 device of the IEEE-ISTO 5001 standard for real-time embedded system design). This interface allows expansion of the development features of the JTAG port (through the addition of auxiliary signals, see Section 2.3.2, "Nexus Mictor Interface Connector"). Development features include:

- Program Trace via Branch Trace Messaging (BTM). Branch trace messaging displays program flow discontinuities (direct and indirect branches, exceptions, etc.), allowing the development tool to interpolate what transpires between the discontinuities. Thus static code may be traced.
- Data Trace via Data Write Messaging (DWM) and Data Read Messaging (DRM). This provides the capability for the development tool to trace reads and/or writes to (selected) internal memory resources.
- Ownership Trace via Ownership Trace Messaging (OTM). OTM facilitates ownership trace by providing visibility of which process ID or operating system task is activated. An Ownership Trace Message is transmitted when a new process/task is activated, allowing the development tool to trace ownership flow.

- Run-time access to the memory map via the JTAG port. This allows for enhanced download/upload capabilities
- Watchpoint Messaging (WPM) via the auxiliary pins
- Watchpoint Trigger enable of Program and/or Data Trace Messaging
- Auxiliary interface for higher data input/output
- Registers for Program Trace, Ownership Trace, Watchpoint Trigger, and Read/Write Access
- Programmable processor stall function to mitigate message queue overrun risk
- All features controllable and configurable via the JTAG port

## 18.2 Software Development Tools

An Integrated Development Environment (IDE) is available to facilitate the development of embedded applications targeting the MC1322x platform. Features of the IDE include:

- Project management tools and code editor
- Highly optimizing ARM compiler supporting C and C++
- Extensive JTAG and RDI debugger support
- Run-time libraries including source code
- Relocating ARM assembler
- Linker and librarian tools
- Debugger with ARM simulator, JTAG support and support for RTOS-aware debugging on hardware
- RTOS plug-ins available
- Code templates for commonly used code constructs
- Sample projects for evaluation boards
- User and reference guides, both printed and in PDF format
- Context-sensitive online help

The IDE is complemented by BeeKit™. BeeKit is a stand alone software application targeting Windows® operating systems. BeeKit™ provides a graphical user interface (GUI) in which the users can create, modify, save, and update wireless networking solutions. With the solution explorer property list windows, users will be able to set configuration parameters that will control the setup and execution behavior of the wireless link within their application. The configuration parameters can be validated inside BeeKit™ to ensure all values provided are within acceptable ranges prior to generation of a workspace. All this functionality provides a mechanism for developers to configure and validate their network parameters without having to navigate through multiple source files to configure the same parameters. BeeKit™ supports Freescale's Simple MAC (SMAC), IEEE 802.15.4-compliant MAC, and BeeStack™.

## 18.3 Development Hardware

Several different development modules and kits will be available to allow evaluation of ZigBee and IEEE 802.15.4 applications. The modules will provide capabilities for Coordinator, Router, and End Device

nodes. Reference designs will be available for RF design and low power applications including 2-layer and 4-layer PCBs.

# Appendix A
# MC1322x Register Address Map

**Table A-1. Register Address Memory Map**

| Base Address | Register Address | Mnemonic | Name | Type | Width (Bits) |
|---|---|---|---|---|---|
| **0x8000_0000** | | | **GPIO** | | |
| | + 0x00 | GPIO_PAD_DIR0 | GPIO Pad Direction for GPIO 00-31 | R/W | 32 |
| | + 0x04 | GPIO_PAD_DIR1 | GPIO Pad Direction for GPIO 32-63 | R/W | 32 |
| | + 0x08 | GPIO_DATA0 | GPIO Data for GPIO 00-31 | R/W | 32 |
| | + 0x0C | GPIO_DATA1 | GPIO Data for GPIO 32-63 | R/W | 32 |
| | + 0x10 | GPIO_PAD_PU_EN0 | GPIO Pad Pull-up Enable for GPIO 00-31 | R/W | 32 |
| | + 0x14 | GPIO_PAD_PU_EN1 | GPIO Pad Pull-up Enable for GPIO 32-63 | R/W | 32 |
| | + 0x18 | GPIO_FUNC_SEL0 | GPIO Function Select for GPIO 00-15 | R/W | 32 |
| | + 0x1C | GPIO_FUNC_SEL1 | GPIO Function Select for GPIO 16-31 | R/W | 32 |
| | + 0x20 | GPIO_FUNC_SEL2 | GPIO Function Select for GPIO 32-47 | R/W | 32 |
| | + 0x24 | GPIO_FUNC_SEL3 | GPIO Function Select for GPIO 48-63 | R/W | 32 |
| | + 0x28 | GPIO_DATA_SEL0 | GPIO Data Select for GPIO 00-31 | R/W | 32 |
| | + 0x2C | GPIO_DATA_SEL1 | GPIO Data Select for GPIO 32-63 | R/W | 32 |
| | + 0x30 | GPIO_PAD_PU_SEL0 | GPIO Pad Pull-up Select for GPIO 00-31 | R/W | 32 |
| | + 0x34 | GPIO_PAD_PU_SEL1 | GPIO Pad Pull-up Select for GPIO 32-63 | R/W | 32 |
| | + 0x38 | GPIO_PAD_HYST_EN0 | GPIO Pad Hysteresis Enable for GPIO 00-31 | R/W | 32 |
| | + 0x3C | GPIO_PAD_HYST_EN1 | GPIO Pad Hysteresis Enable for GPIO 32-63 | R/W | 32 |
| | + 0x40 | GPIO_PAD_KEEP0 | GPIO Pad Keeper Enable for GPIO 00-31 | R/W | 32 |
| | + 0x44 | GPIO_PAD_KEEP1 | GPIO Pad Keeper Enable for GPIO 32-63 | R/W | 32 |
| | + 0x48 | GPIO_DATA_SET0 | GPIO Data Set for GPIO 00-31 | W | 32 |
| | + 0x4C | GPIO_DATA_SET1 | GPIO Data Set for GPIO 32-63 | W | 32 |
| | + 0x50 | GPIO_DATA_RESET0 | GPIO Data Reset for GPIO 00-31 | W | 32 |
| | + 0x54 | GPIO_DATA_RESET1 | GPIO Data Reset for GPIO 32-63 | W | 32 |
| | + 0x58 | GPIO_PAD_DIR_SET0 | GPIO Pad Direction Set for GPIO 00-31 | W | 32 |
| | + 0x5C | GPIO_PAD_DIR_SET1 | GPIO Pad Direction Set for GPIO 32-63 | W | 32 |
| | + 0x60 | GPIO_PAD_DIR_RESET0 | GPIO Pad Direction Reset for GPIO 00-31 | W | 32 |

**Table A-1. Register Address Memory Map (continued)**

| Base Address | Register Address | Mnemonic | Name | Type | Width (Bits) |
|---|---|---|---|---|---|
| | + 0x64 | GPIO_PAD_DIR_RESET1 | GPIO Pad Direction Reset for GPIO 32-63 | W | 32 |
| **0x8000_1000** | | | **SSI** | | |
| | + 0x00 | SSI_STX | SSI Transmit Data Register | R/W | 32 |
| | + 0x04 | | Reserved | | |
| | + 0x08 | SSI_SRX | SSI Receive Data Register | R | 32 |
| | + 0x0C | | Reserved | | |
| | + 0x10 | SSI_SCR | SSI Control Register | R/W | 32 |
| | + 0x14 | SSI_SISR | SSI Interrupt Status Register | R | 32 |
| | + 0x18 | SSI_SIER | SSI Interrupt Enable Register | R/W | 32 |
| | + 0x1C | SSI_STCR | SSI Transmit Configuration Register | R/W | 32 |
| | + 0x20 | SSI_SRCR | SSI Receive Configuration Register | R/W | 32 |
| | + 0x24 | SSI_STCCR | SSI Transmit Clock Control Register | R/W | 32 |
| | + 0x28 | | Reserved | | |
| | + 0x2C | SSI_SFCSR | SSI FIFO Control Status Register | R/W | 32 |
| | + 0x30 to + 0x44 | | Reserved | | |
| | + 0x48 | SSI_STMSK | SSI Transmit Time Slot Mask Register | R/W | 32 |
| | + 0x4C | SSI_SRMSK | SSI Receive Time Slot Mask Register | R/W | 32 |
| | + 0x50 to + 0x58 | | Reserved | | |
| **0x8000_2000** | | | **SPI** | | |
| | + 0x00 | SPI_TX_DATA | SPI Transmit Data register | R/W | 32 |
| | + 0x04 | SPI_RX_DATA | SPI Received Data register | R/W | 32 |
| | + 0x08 | SPI_CLK_CTRL | SPI Status and Clock Control register | R/W | 32 |
| | + 0x0C | SPI_SETUP | SPI Setup register | R/W | 32 |
| | + 0x10 | SPI_STATUS | SPI Status register | R/W | 32 |
| **0x8000_3000** | | | **Clock/Reset/Power Module (CRM)** | | |
| | + 0x00 | SYS_CNTL | CRM System Control | R/W | 32 |
| | + 0x04 | WU_CNTL | CRM Wake up Control | R/W | 32 |
| | + 0x08 | SLEEP_CNTL | CRM Sleep Control | R/W | 32 |
| | + 0x0C | BS_CNTL | CRM Bus Stealing Control | R/W | 32 |
| | + 0x10 | COP_CNTL | CRM COP Control | R/W | 32 |

**Table A-1. Register Address Memory Map (continued)**

| Base Address | Register Address | Mnemonic | Name | Type | Width (Bits) |
|---|---|---|---|---|---|
| | + 0x14 | COP_SERVICE | CRM COP Service | R/W | 32 |
| | + 0x18 | STATUS | CRM Event Status | R/W | 32 |
| | + 0x1C | MOD_STATUS | CRM Module Enable Status | R | 32 |
| | + 0x20 | WU_COUNT | CRM wake-up Count | R | 32 |
| | + 0x24 | WU_TIMEOUT | CRM wake-up Time-out | R/W | 32 |
| | + 0x28 | RTC_COUNT | CRM Real Time Count | R/W | 32 |
| | + 0x2C | RTC_TIMEOUT | CRM RTC Periodic Wake up Time-out | R/W | 32 |
| | + 0x30 | | Reserved | | |
| | + 0x34 | CAL_CNTL | CRM Calibration Control | R/W | 32 |
| | + 0x38 | CAL_COUNT | CRM Calibration XTAL Count | R | 32 |
| | + 0x3C | RINGOSC_CNTL | CRM 2kHz Ring Oscillator Control | R/W | 32 |
| | + 0x40 | XTAL_CNTL | CRM Reference XTAL Control | R/W | 32 |
| | + 0x44 | XTAL32_CNTL | CRM 32kHz XTAL Control | R/W | 32 |
| | + 0x48 | VREG_CNTL | CRM Voltage Regulator Control | R/W | 32 |
| | + 0x4C | | Reserved | | |
| | + 0x50 | SW_RST | CRM Software Reset | R/W | 32 |
| | + 0x54 to 0x5C | | Reserved | | |
| **0x8000_4000** | | | **MAC Accelerator (MACA)** | | |
| | + 0x00 | | Reserved | | |
| | + 0x04 | MACA_RESET | Reset | R/W | 32 |
| | + 0x08 | MACA_RANDOM | Random number | R/W | 32 |
| | + 0x0C | MACA_CONTROL | Control | W | 32 |
| | + 0x10 | MACA_STATUS | Status | R | 32 |
| | + 0x14 | MACA_FRMPND | Frame Pending | R/W | 32 |
| | + 0x18 | MACA_MC1322x_ID | MC1322X_ID | R/W | 32 |
| | + 0x1C to 0x3C | | Reserved | | |
| | + 0x40 | MACA_TMREN | MACA Enable Timers | R/W | 32 |
| | + 0x44 | MACA_TMRDIS | MACA Disable Timers | R/W | 32 |
| | + 0x48 | MACA_CLK | MACA Clock | R/W | 32 |
| | + 0x4C | MACA_STARTCLK | MACA Start Clock | R/W | 32 |
| | + 0x50 | MACA_CPLCLK | MACA Complete Clock | R/W | 32 |

**Table A-1. Register Address Memory Map (continued)**

| Base Address | Register Address | Mnemonic | Name | Type | Width (Bits) |
|---|---|---|---|---|---|
| | + 0x54 | MACA_SFTCLK | MACA Soft Time-out Clock | R/W | 32 |
| | + 0x58 | MACA_CLKOFFSET | MACA Clock offset | R/W | 32 |
| | + 0x5C | MACA_RELCLK | MACA Relative clock | R/W | 32 |
| | + 0x60 | MACA_CPLTIM | MACA Action Complete Timestamp | R | 32 |
| | + 0x64 | MACA_SLOTOFFSET | MACA Tx Slot Offset Adjustment | R/W | 32 |
| | + 0x68 | MACA_TIMESTAMP | MACA Receive Time Stamp | R | 32 |
| | + 0x6C to 0x7C | | Reserved | | |
| | + 0x80 | MACA_DMARX | MACA DMA Rx Data Pointer | R/W | 32 |
| | + 0x84 | MACA_DMATX | MACA DMA Tx Data Pointer | R/W | 32 |
| | + 0x88 | MACA_DMAPOLL | MACA DMA Tx Poll Response Pointer | R/W | 32 |
| | + 0x8C | MACA_TXLEN | MACA Tx Length | R/W | 32 |
| | + 0x90 | MACA_TXSEQNR | MACA Tx Sequence Number | R/W | 32 |
| | + 0x94 | MACA_SETRXLVL | MACA Set Rx Level Interrupt | R/W | 32 |
| | + 0x98 | MACA_GETRXLVL | MACA Read Number of Received Bytes | R | 32 |
| | + 0x9C to 0xBC | | Reserved | | |
| | + 0xC0 | MACA_IRQ | MACA Interrupt Status | R | 32 |
| | + 0xC4 | MACA_CLRIRQ | MACA Interrupt Clear | W | 32 |
| | + 0xC8 | MACA_SETIRQ | MACA Interrupt Set | W | 32 |
| | + 0xCC | MACA_MASKIRQ | MACA Interrupt Mask | W | 32 |
| | + 0xD0 to 0xFC | | Reserved | | |
| | + 0x100 | MACA_MACPANID | MACA PAN ID | R/W | 32 |
| | + 0x104 | MACA_MAC16ADDR | MACA Short Address | R/W | 32 |
| | + 0x108 | MACA_MAC64HI | MACA High Extended Address | R/W | 32 |
| | + 0x10C | MACA_MAC64LO | MACA Low Extended Address | R/W | 32 |
| | + 0x110 | MACA_FLTREJ | MACA Filter Rejection Mask | R/W | 32 |
| | + 0x114 | MACA_CLKDIV | MACA Clock Divider | R/W | 32 |
| | + 0x118 | MACA_WARMUP | MACA Warmup | R/W | 32 |
| | + 0x11C | MACA_PREAMBLE | MACA Preamble | R/W | 32 |
| | + 0x120 | MACA_WHITESEED | Reserved | | |
| | + 0x124 | MACA_FRAMESYNC0 | MACA Frame Sync Word 0 | R/W | 32 |

**Table A-1. Register Address Memory Map (continued)**

| Base Address | Register Address | Mnemonic | Name | Type | Width (Bits) |
|---|---|---|---|---|---|
| | + 0x128 | MACA_FRAMESYNC1 | MACA Frame Sync Word 1 | R/W | 32 |
| | + 0x12C to 0x13C | | Reserved | | |
| | + 0x140 | MACA_TXACKDELAY | MACA Tx Acknowledgement Delay | R/W | 32 |
| | + 0x144 | MACA_RXACKDELAY | MACA Rx Acknowledgement Delay | R/W | 32 |
| | + 0x148 | MACA_EOFDELAY | MACA End of Frame Delay | R/W | 32 |
| | + 0x14C | MACA_CCADELAY | MACA CCA Delay | R/W | 32 |
| | + 0x150 | MACA_RXEND | MACA Rx End | R/W | 32 |
| | + 0x154 | MACA_TXCCADELAY | MACA Tx CCA Delay | R/W | 32 |
| | + 0x158 | MACA_KEY3 | MACA Key3 | R | 32 |
| | + 0x15C | MACA_KEY2 | MACA Key2 | R | 32 |
| | + 0x160 | MACA_KEY1 | MACA Key1 | R | 32 |
| | + 0x164 | MACA_KEY0 | MACA Key0 | R | 32 |
| | + 0x180 | MACA_OPTIONS | MACA Options | W | 32 |
| **0x8000_5000** | | | **UART1** | | |
| | + 0x00 | UART1_UCON | UART Control | R/W | 8 |
| | + 0x04 | UART1_USTAT | UART Status | R | 8 |
| | + 0x08 | UART1_UDATA | UART Data | R/W | 8 |
| | + 0x0C | UART1_URXCON | UART RxBuffer Control | R/W | 8 |
| | + 0x10 | UART1_UTXCON | UART TxBuffer Control | R/W | 8 |
| | + 0x14 | UART1_UCTS | UART CTS Level Control | R/W | 8 |
| | + 0x18 | UART1_UBR | UBRINC — Fractional Divider/ UBRMOD — Fractional Divider | R/W | 8 |
| **0x8000_6000** | | | **$I^2C$** | | |
| | + 0x00 | I2C_ADR | $I^2C$ Address Register | R/W | 8 |
| | + 0x04 | I2C_FDR | $I^2C$ Frequency Divider Register | R/W | 8 |
| | + 0x08 | I2C_CR | $I^2C$ Control Register | R/W | 8 |
| | + 0x0C | I2C_SR | $I^2C$ Status Register | R/W | 8 |
| | + 0x10 | I2C_DR | $I^2C$ Data Register | R/W | 8 |
| | + 0x14 | I2C_DFSRR | $I^2C$ Digital Filter Sampling Rate Register | R/W | 8 |
| | + 0x18 | I2C_CKER | $I^2C$ Clock Enable Register | R/W | 8 |
| **0x8000_7000** | | | **TIMER** | | |

**Table A-1. Register Address Memory Map (continued)**

| Base Address | Register Address | Mnemonic | Name | Type | Width (Bits) |
|---|---|---|---|---|---|
| | + 0x00 | TMR0_COMP1 | TMR Channel 0 Compare Register 1 | R/W | 16 |
| | + 0x02 | TMR0_COMP2 | TMR Channel 0 Compare Register 2 | R/W | 16 |
| | + 0x04 | TMR0_CAPT | TMR Channel 0 Capture Register | R/W | 16 |
| | + 0x06 | TMR0_LOAD | TMR Channel 0 Load Register | R/W | 16 |
| | + 0x08 | TMR0_HOLD | TMR Channel 0 Hold Register | R/W | 16 |
| | + 0x0A | TMR0_CNTR | TMR Channel 0 Counter Register | R/W | 16 |
| | + 0x0C | TMR0_CTRL | TMR Channel 0 Control Register | R/W | 16 |
| | + 0x0E | TMR0_SCTRL | TMR Channel 0 Status and Control Register | R/W | 16 |
| | + 0x10 | TMR0_CMPLD1 | TMR Channel 0 Comparator Load Register 1 | R/W | 16 |
| | + 0x12 | TMR0_CMPLD2 | TMR Channel 0 Comparator Load Register 2 | R/W | 16 |
| | + 0x14 | TMR0_CSCTRL | TMR Channel 0 Comparator Status and Control Register | R/W | 16 |
| | + 0x16 to + 0x1C | | Reserved | | |
| | + 0x1E | TMR0_ENBL | TMR Channel Enable Register | R/W | 16 |
| | + 0x20 | TMR1_COMP1 | TMR Channel 1 Compare Register 1 | R/W | 16 |
| | + 0x22 | TMR1_COMP2 | TMR Channel 1 Compare Register 2 | R/W | 16 |
| | + 0x24 | TMR1_CAPT | TMR Channel 1 Capture Register | R/W | 16 |
| | + 0x26 | TMR1_LOAD | TMR Channel 1 Load Register | R/W | 16 |
| | + 0x28 | TMR1_HOLD | TMR Channel 1 Hold Register | R/W | 16 |
| | + 0x2A | TMR1_CNTR | TMR Channel 1 Counter Register | R/W | 16 |
| | + 0x2C | TMR1_CTRL | TMR Channel 1 Control Register | R/W | 16 |
| | + 0x2E | TMR1_SCTRL | TMR Channel 1 Status and Control Register | R/W | 16 |
| | + 0x30 | TMR1_CMPLD1 | TMR Channel 1 Comparator Load Register 1 | R/W | 16 |
| | + 0x32 | TMR1_CMPLD2 | TMR Channel 1 Comparator Load Register 2 | R/W | 16 |
| | + 0x34 | TMR1_CSCTRL | TMR Channel 1 Comparator Status and Control Register | R/W | 16 |
| | + 0x36 to + 0x3C | | Reserved | | |
| | + 0x40 | TMR2_COMP1 | TMR Channel 2 Compare Register 1 | R/W | 16 |
| | + 0x42 | TMR2_COMP2 | TMR Channel 2 Compare Register 2 | R/W | 16 |
| | + 0x44 | TMR2_CAPT | TMR Channel 2 Capture Register | R/W | 16 |
| | + 0x46 | TMR2_LOAD | TMR Channel 2 Load Register | R/W | 16 |
| | + 0x48 | TMR2_HOLD | TMR Channel 2 Hold Register | R/W | 16 |

**Table A-1. Register Address Memory Map (continued)**

| Base Address | Register Address | Mnemonic | Name | Type | Width (Bits) |
|---|---|---|---|---|---|
| | + 0x4A | TMR2_CNTR | TMR Channel 2 Counter Register | R/W | 16 |
| | + 0x4C | TMR2_CTRL | TMR Channel 2 Control Register | R/W | 16 |
| | + 0x4E | TMR2_SCTRL | TMR Channel 2 Status and Control Register | R/W | 16 |
| | + 0x50 | TMR2_CMPLD1 | TMR Channel 2 Comparator Load Register 1 | R/W | 16 |
| | + 0x52 | TMR2_CMPLD2 | TMR Channel 2 Comparator Load Register 2 | R/W | 16 |
| | + 0x54 | TMR2_CSCTRL | TMR Channel 2 Comparator Status and Control Register | R/W | 16 |
| | + 0x56 to + 0x5C | | Reserved | | |
| | + 0x60 | TMR3_COMP1 | TMR Channel 3 Compare Register 1 | R/W | 16 |
| | + 0x62 | TMR3_COMP2 | TMR Channel 3 Compare Register 2 | R/W | 16 |
| | + 0x64 | TMR3_CAPT | TMR Channel 3 Capture Register | R/W | 16 |
| | + 0x66 | TMR3_LOAD | TMR Channel 3 Load Register | R/W | 16 |
| | + 0x68 | TMR3_HOLD | TMR Channel 3 Hold Register | R/W | 16 |
| | + 0x6A | TMR3_CNTR | TMR Channel 3 Counter Register | R/W | 16 |
| | + 0x6C | TMR3_CTRL | TMR Channel 3 Control Register | R/W | 16 |
| | + 0x6E | TMR3_SCTRL | TMR Channel 3 Status and Control Register | R/W | 16 |
| | + 0x70 | TMR3_CMPLD1 | TMR Channel 3 Comparator Load Register 1 | R/W | 16 |
| | + 0x72 | TMR3_CMPLD2 | TMR Channel 3 Comparator Load Register 2 | R/W | 16 |
| | + 0x74 | TMR3_CSCTRL | TMR Channel 3 Comparator Status and Control Register | R/W | 16 |
| | + 0x76 to + 0x7E | | Reserved | | |
| **0x8000_8000** | | | **Advanced Security Module (ASM)** | | |
| | + 0x00 | ASM_KEY0 | 128-BIT ENCRYPTION KEY (1 of 4) | R/W | 32 |
| | + 0x04 | ASM_KEY1 | 128-BIT ENCRYPTION KEY (2 of 4) | R/W | 32 |
| | + 0x08 | ASM_KEY2 | 128-BIT ENCRYPTION KEY (3 of 4) | R/W | 32 |
| | + 0x0C | ASM_KEY3 | 128-BIT ENCRYPTION KEY (4 of 4) | R/W | 32 |
| | + 0x10 | ASM_DATA0 | 128-BIT DATA Register (1 of 4) | R/W | 32 |
| | + 0x14 | ASM_DATA1 | 128-BIT DATA Register (2 of 4) | R/W | 32 |
| | + 0x18 | ASM_DATA2 | 128-BIT DATA Register (3 of 4) | R/W | 32 |
| | + 0x1C | ASM_DATA3 | 128-BIT DATA Register (4 of 4) | R/W | 32 |
| | + 0x20 | ASM_CTR0 | 128-BIT COUNTER Register (1 of 4) | R/W | 32 |

**Table A-1. Register Address Memory Map (continued)**

| Base Address | Register Address | Mnemonic | Name | Type | Width (Bits) |
|---|---|---|---|---|---|
| | + 0x24 | ASM_CTR1 | 128-BIT COUNTER Register (2 of 4) | R/W | 32 |
| | + 0x28 | ASM_CTR2 | 128-BIT COUNTER Register (3 of 4) | R/W | 32 |
| | + 0x2C | ASM_CTR3 | 128-BIT COUNTER Register (4 of 4) | R/W | 32 |
| | + 0x30 | ASM_CTR0_RESULT | 128-BIT COUNTER RESULT Register (1 of 4) | R | 32 |
| | + 0x34 | ASM_CTR1_RESULT | 128-BIT COUNTER RESULT Register (2 of 4) | R | 32 |
| | + 0x38 | ASM_CTR2_RESULT | 128-BIT COUNTER RESULT Register (3 of 4) | R | 32 |
| | + 0x3C | ASM_CTR3_RESULT | 128-BIT COUNTER RESULT Register (4 of 4) | R | 32 |
| | + 0x40 | ASM_CBC0_RESULT | 128-BIT CBC MAC RESULT Register (1 of 4) | R | 32 |
| | + 0x44 | ASM_CBC1_RESULT | 128-BIT CBC MAC RESULT Register (2 of 4) | R | 32 |
| | + 0x48 | ASM_CBC2_RESULT | 128-BIT CBC MAC RESULT Register (3 of 4) | R | 32 |
| | + 0x4C | ASM_CBC3_RESULT | 128-BIT CBC MAC RESULT Register (4 of 4) | R | 32 |
| | + 0x50 | ASM_CONTROL0 | CONTROL 0 Register (Self Clearing) | W | 32 |
| | + 0x54 | ASM_CONTROL1 | CONTROL 1 Register | R/W | 32 |
| | + 0x58 | ASM_STATUS | STATUS Register | R | 32 |
| | + 0x5C | | Reserved | | |
| | + 0x60 | ASM_MAC0 | 128-BIT CBC MAC Register (1 of 4) | R/W | 32 |
| | + 0x64 | ASM_MAC1 | 128-BIT CBC MAC Register (2 of 4) | R/W | 32 |
| | + 0x68 | ASM_MAC2 | 128-BIT CBC MAC Register (3 of 4) | R/W | 32 |
| | + 0x6C | ASM_MAC3 | 128-BIT CBC MAC Register (4 of 4) | R/W | 32 |
| **0x8000_9000** | | | **Modem Clock Synthesizer** | | |
| | + 0x00 | SYN_ENABLE | Enable and Override Register | R/W | 32 |
| | + 0x04 | | Reserved | | |
| | + 0x08 | SYN_REFDIV | Reference Loop Divider Register | R/W | 32 |
| | + 0x0C | SYN_VCODIV | VCO Loop Divider Register | R/W | 32 |
| | + 0x10 to + 0x28 | | Reserved | | |
| **0x8000_9200 - 0x8000_A000** | | | **Reserved** | | |
| **0x8000_B000** | | | **UART2** | | |
| | + 0x00 | UART2_UCON | UART2 Control | R/W | 8 |
| | + 0x04 | UART2_USTAT | UART2 Status | R | 8 |
| | + 0x08 | UART2_UDATA | UART2 Data | R/W | 8 |

**Table A-1. Register Address Memory Map (continued)**

| Base Address | Register Address | Mnemonic | Name | Type | Width (Bits) |
|---|---|---|---|---|---|
| | + 0x0C | UART2_URXCON | UART2 RxBuffer Control | R/W | 8 |
| | + 0x10 | UART2_UTXCON | UART2 TxBuffer Control | R/W | 8 |
| | + 0x14 | UART2_UCTS | UART2 TxBuffer Control | R/W | 8 |
| | + 0x18 | UART2_UBR | UBRINC — Fractional Divider/ UBRMOD — Fractional Divider | R/W | 8 |
| **0x8000_C000** | | | **Reserved** | | |
| | + 0x00-0x20 | | | | |
| **0x8000_D000** | | | **ADC** | | |
| | + 0x00 | ADC_COMP_0 | Monitor GP-ADC Pins Threshold detection w/o MCU intervention to generate IRQ | R/W | 16 |
| | + 0x02 | ADC_COMP_1 | Monitor GP-ADC Pins Threshold | R/W | 16 |
| | + 0x04 | ADC_COMP_2 | Monitor GP-ADC Pins Threshold | R/W | 16 |
| | + 0x06 | ADC_COMP_3 | Monitor GP-ADC Pins Threshold | R/W | 16 |
| | + 0x08 | ADC_COMP_4 | Monitor GP-ADC Pins Threshold | R/W | 16 |
| | + 0x0A | ADC_COMP_5 | Monitor GP-ADC Pins Threshold | R/W | 16 |
| | + 0x0C | ADC_COMP_6 | Monitor GP-ADC Pins Threshold | R/W | 16 |
| | + 0x0E | ADC_COMP_7 | Monitor GP-ADC Pins Threshold | R/W | 16 |
| | + 0x10 | ADC_BAT_COMP_OVER | Battery Voltage Upper Trip Point (ADC1 only) | R/W | 16 |
| | + 0x12 | ADC_BAT_COMP_UNDER | Battery Voltage Lower Trip Point (ADC1 only) | R/W | 16 |
| | + 0x14 | ADC_SEQ_1 | Monitor GP-ADC Pins for ADC1 | R/W | 16 |
| | + 0x16 | ADC_SEQ_2 | Monitor GP-ADC Pins for ADC2 | R/W | 16 |
| | + 0x18 | ADC_CONTROL | Primary Module Control Register | R/W | 16 |
| | + 0x1A | ADC_TRIGGERS | Triggered Channels Register | R | 16 |
| | + 0x1C | ADC_PRESCALE | Bus Clock Divide Register (8-bit prescaler for use with 26MHz) | R/W | 16 |
| | + 0x1E | | Reserved | | |
| | + 0x20 | ADC_FIFO_READ | ADC FIFO Read (8 deep FIFO) | R | 16 |
| | + 0x22 | ADC_FIFO_CONTROL | ADC Interrupt Level Control | R/W | 16 |
| | + 0x24 | ADC_FIFO_STATUS | ADC FIFO Status | R | 16 |
| | + 0x26 to 0x2E | | Reserved | | |
| | + 0x30 | ADC_1_SR_HIGH | ADC 1 Sample Rate Low | R/W | 16 |
| | + 0x32 | ADC_1_SR_LOW | ADC 1 Sample Rate High | R/W | 16 |

**Table A-1. Register Address Memory Map (continued)**

| Base Address | Register Address | Mnemonic | Name | Type | Width (Bits) |
|---|---|---|---|---|---|
| | + 0x34 | ADC_2_SR_HIGH | ADC 2 Sample Rate Low | R/W | 16 |
| | + 0x36 | ADC_2_SR_LOW | ADC 2 Sample Rate High | R/W | 16 |
| | + 0x38 | ADC_ON_TIME | ADC TurnOn Time | R/W | 16 |
| | + 0x3A | ADC_CONVERT_TIME | ADC Convert Time | R/W | 16 |
| | + 0x3C | ADC_CLOCK_DIVIDER | ADC Clock Divider | R/W | 16 |
| | + 0x3E | | Reserved | | |
| | + 0x40 | ADC_OVERRIDE | ADC Manual Control | R/W | 16 |
| | + 0x42 | ADC_IRQ | ADC Read/Clear Active Interrupts | R/W | 16 |
| | + 0x44 | ADC_MODE | ADC Mode | R/W | 16 |
| | + 0x46 | ADC_1_RESULT | ADC 1 Result | R | 16 |
| | + 0x48 | ADC_2_RESULT | ADC 2 Result | R | 16 |
| **0x8002_0000** | | | **Interrupt Controller** | | |
| | + 0x00 | INTCNTL | ITC Interrupt Controller | R/W | 32 |
| | + 0x04 | NIMASK | ITC Normal Interrupt Mask | R/W | 32 |
| | + 0x08 | INTENNUM | ITC Interrupt Enable Number | W | 32 |
| | + 0x0c | INTDISNUM | ITC Interrupt Disable Number | W | 32 |
| | + 0x10 | INTENABLE | ITC Interrupt Enable | R/W | 32 |
| | + 0x14 | INTTYPE | ITC Interrupt Type | R/W | 32 |
| | + 0x18 | | Reserved | N/A | 32 |
| | + 0x1C | | Reserved | N/A | 32 |
| | + 0x20 | | Reserved | N/A | 32 |
| | + 0x24 | | Reserved | N/A | 32 |
| | + 0x28 | NIVECTOR | ITC Normal Interrupt | R | 32 |
| | + 0x2C | FIVECTOR | ITC Fast Interrupt | R | 32 |
| | + 0x30 | INTSRC | ITC Interrupt Source | R | 32 |
| | + 0x34 | INTFRC | ITC Interrupt Force | R/W | 32 |
| | + 0x38 | NIPEND | ITC Normal Interrupt Pending | R | 32 |
| | + 0x3C | FIPEND | ITC Fast Interrupt Pending | R | 32 |

# Appendix B
# MC1322x Software Driver Utilities

## B.1 Overview

The MC1322x is a full Platform-in-Package device having a complex set of transceiver and MCU modules and peripherals. To simplify software development, Freescale has written and provides an extensive set of driver utilities for the platform that are available through the on board ROM.

## B.2 Driver Summary

The drivers are documented in the *MC1322x Software Driver Reference Manual*, (22XDRVRRM). Those drivers described in the reference manual include:

- Clock and Reset Module (CRM) Driver
- GPIO Drivers
- Interrupt Controller (ITC) Driver
- Non-Volatile Memory (NVM) Driver
- UART Driver
- Timer (TMR) Driver
- Inter-integrated Circuit ($I^2C$) Driver
- Synchronous Serial Interface (SSI) Driver
- Analog to Digital Converter (ADC) Driver

## B.3 Using the Drivers

The Freescale BeeKit development suite provides access to the drivers through the BeeStack platform components or as function calls.

**NOTE**

For reference, the user is directed to the following documents provided with Freescale's BeeKit development suite:

- *Freescale BeeStack™ Application Development Guide for ZigBee 2007*, (22XDRVRRM)
- *Freescale Platform Reference Manual for ZigBee 2007*, (FSPRMZB2007)
- *MC1322x Software Driver Reference Manual*, (22XDRVRRM)

# Appendix C
# Bootloader Reference

## C.1 Overview

The MC1322x has a ROM-based bootloader that is described in Section 3.12, “Bootloader”. Common usage employs the on board FLASH and the file image format used is transparent to the user. The FLASH image is generated through development tools and loaded through the debug port, typically the JTAG port.

The bootloader does allow alternative means of supplying the RAM boot image. This appendix provides format and protocol information to utilize these other sources for the RAM target code image.

## C.2 Alternative Load Ports

The bootstrap flow chart as shown in Figure 3-23 and its accompanying explanation describes use of the following boot ports:

- Load the target image from UART1 port
- Load the target image from the SPI port (as slave) attached to a master device
- Initiate the SPI port as a master and load target image from external slave (serial FLASH)
- Load the target image from $I^2C$ (serial EEPROM)

**NOTE**

External pull-ups on “test” pins (such as UART1_RTS and SPI_SS) are not required as the on board pull-ups are enabled.

## C.3 Booting from UART1

If it has been determined that onboard FLASH does not contain valid code, the bootstrap will first try to boot from UART1.

### C.3.1 Procedure

The UART1 boot procedure includes:

1. The UART1 signals get initiated as:
   - GPIO14 becomes UART1_TX output
   - GPIO15 becomes UART1_RX input
   - GPIO16 becomes UART1_CTS output
   - GPIO17 becomes UART1_RTS input

2. Control input UART1_RTS is tested and must have been driven low externally before being tested. This indicates a device is attached ready to supply data.

**NOTE**

If UART1_RTS is tested and is not driven low, the procedure exits and moves to the SPI boot flow, Section C.4, “Booting using SPI on the MC1322x”.

3. Auto-baud Detection - In order to properly boot with any baudrate selected by the attached host (up to 2M baud), the bootstrap determines the host baudrate based on a “zero” character.
   - The host must first transmit a single ASCII NULL (numeric “zero” or all low) byte -
     - Format is one start bit and an 8-bit character. This results in a 9-bit wide pulse (start bit plus the 8 zero bits) that the MC1322x uses to test the baud rate.
     - The host can repeat this single byte transmission if the MC1322x does not respond, however, the host should wait a sufficient delay to allow the MC1322x to respond.
   - Once the MC1322x has determined the baud rate, it responds back with the message “CONNECT”. The response to the “zero” or null byte takes around 25 to 26 µs. This serves two purposes:
     - Confirms the baud rate
     - Notifies the host device that it can now continue.
4. The host sends the length field and RAM data in the format defined in Section C.3.2, “Data Format”.

Figure C-1 shows the message sequence chart for the interaction between the host and the MC1322x.

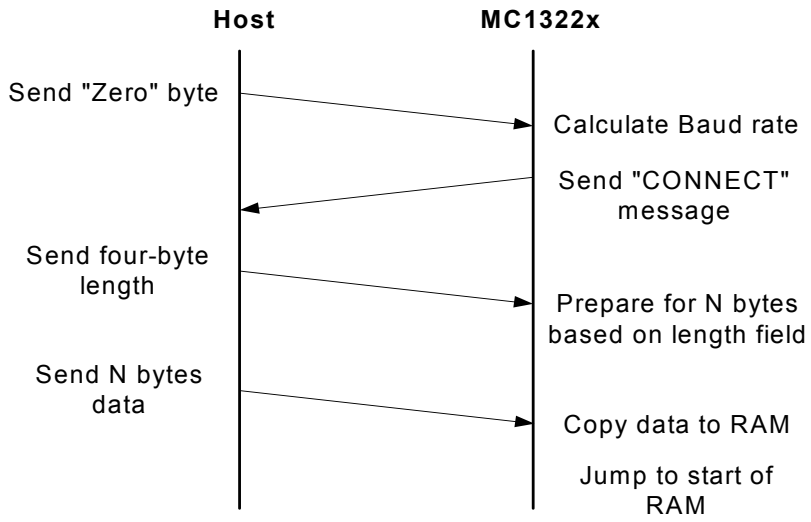



**Figure C-1. Host/MC1322x UART1 Boot Sequence**

### C.3.2 Data Format

After completing the connection sequence (transmitting the “CONNECT” message):

1. The MC1322x first expects a four-byte length field (32-bit little-endian) from the host. This four-byte value determines the number of expected bytes from the host that will get loaded into RAM starting at address 0x0040_0000.
2. The host then sends the stream of bytes which are loaded into RAM (also little-endian).

After the MC1322x gets the expected number of bytes, the bootloader jumps to the RAM start address to begin program execution.

## C.4 Booting using SPI on the MC1322x

If UART1_RTS is tested and is not driven low, the procedure exits UART1 consideration and moves to the SPI boot flow. At this point the SPI_SS is initiated as an input and tested for its state:

- If SPI_SS is low - It is assumed an external SPI master is present, the MC1322x configures its SPI to slave mode, and waits for the SPI master to clock the SPI (SPI_SS is asserted).
- If SPI_SS is high - It is assumed that an external SPI slave is present (typically a serial FLASH device), the MC1322x configures its SPI to master mode, and tries to boot from the external SPI device.

**NOTE**

- SPI data is always transferred as bytes with MSB first.
- Clock mode is SCK_PHASE = 0 and SCK_POL = 0

### C.4.1 MC1322x Is SPI Slave

The MC1322x SPI slave boot procedure includes:

1. The SPI signals are initiated as:
   — GPIO4 is SPI_SS input
   — GPIO5 is SPI_MISO output
   — GPIO6 is SPI_MOSI input
   — GPIO77 is SPI_SCK input
2. SPI_SS is presently asserted (low) and the MC1322x SPI gets configured as a slave and waits for the external SPI master to clock the port to transfer data (8-bit transfers are assumed). Also during configuration, the slave data buffer gets loaded with the four ASCII bytes for “RDY!”
3. The master transfers four bytes; sending the binary equivalent of the four-byte length field (32-bit little endian) and receiving the “RDY!” message.
4. The master continues to transfer the RAM data.

Figure C-2 shows the message sequence chart for the interaction between the SPI master and the MC1322x.

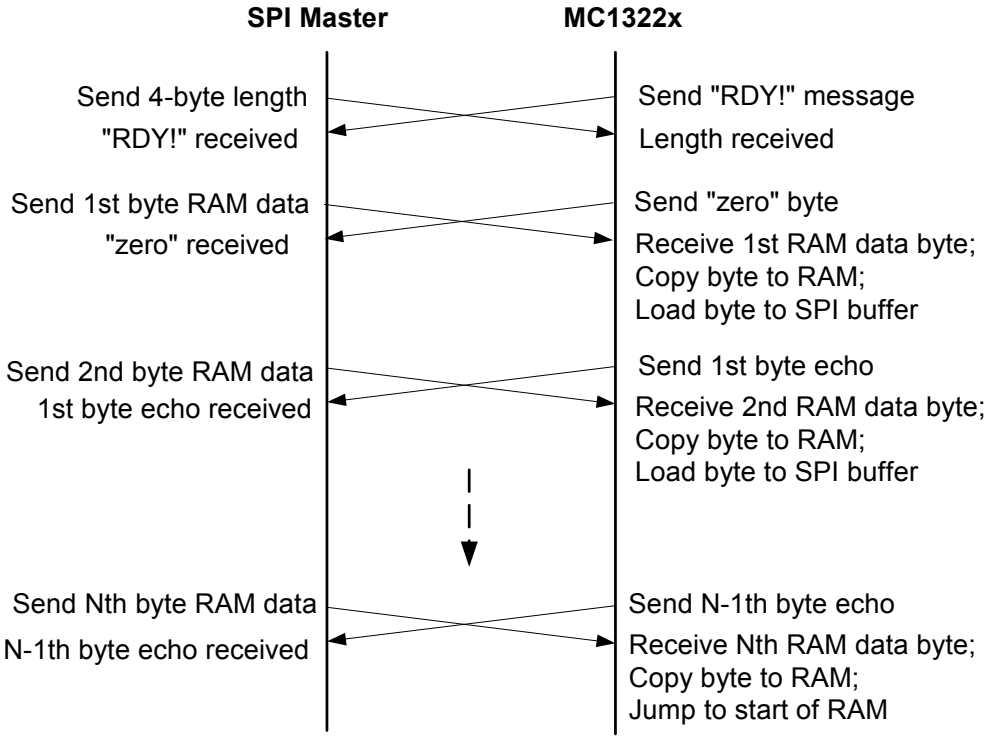



**Figure C-2. SPI Master /MC1322x Boot Sequence**

## C.4.2 MC1322x Is SPI Slave Data Format

The exchange format is as follows:

1. Complete the connection sequence - Master transmits the length field and receives “RDY!” from slave (four ASCII characters “R”, “D”, “Y”, “!” in the order shown from left to right)
2. The MC1322x as Slave loads the data buffer with a binary zero (byte = 0x00) in anticipation of receiving the RAM data stream.
3. The external master sends the first RAM data byte and this zero-data is transmitted to the master.
4. On each subsequent byte received from the master, the MC1322x will transmit the previously received byte. This is used as a simple "acknowledgment", and validates the just received data to be correct.
5. The external master sends the expected number of bytes as determined by the length field

The bootloader finishes loading the RAM, and then jumps to the RAM start address to begin program execution.

**NOTE**

If SPI master fails to send expected number of bytes, the MC1322x will “hang” waiting for the RAM data transfer to complete.

### C.4.3 MC1322x Is SPI Master (Boot from External FLASH)

If the SPI_SS has not tested as low, the next trial assumes that an external SPI slave is present (typically a serial FLASH device)

**NOTE**

The MC1322x supports FLASH memories formatted similar to the devices below:

- 1 Mbit, serial FLASH memory M25P10-A from ST Microelectronics
- 1 Mbit, serial FLASH memory AT25F1024 from Atmel Corp

The MC1322x as SPI master boot procedure includes:

1. The SPI signals are initiated as:
   — GPIO4 is SPI_SS output
   — GPIO5 is SPI_MISO input
   — GPIO6 is SPI_MOSI output
   — GPIO7 is SPI_SCK output
2. The MC1322x configures the on board SPI as master and asserts SPI_SS (low) -
   — SCK_PHASE = 0 and SCK_POL = 0
   — SPI bit clock frequency = 12 MHz
   — Transfer bit length = 8 bits
3. The MC1322x reads the first four bytes from the external SPI slave and validates that the contents are the ASCII string “OKOK” (addr 0x0 = “O”, addr 0x1 = “K”, addr 0x2 = “O”, addr 0x3 = “K”)

**NOTE**

If any other content is read, the FLASH is assumed to be corrupted or not present. The boot flow then moves to look for an $I^2C$ EEPROM.

4. Upon reading the validation code, the bootstrap will load the subsequent four bytes (32-bit, little-endian), and use this as the length field indicator for how many bytes to load into RAM.
5. The MC1322x continues reading bytes from FLASH and loading them into RAM starting with the first RAM address (0x0040_0000).

Once all code is loaded into RAM , the bootstrap will jump to the first RAM address (0x00400000) and continue from there. Figure C-3 shows the interaction between SPI master and external FLASH.

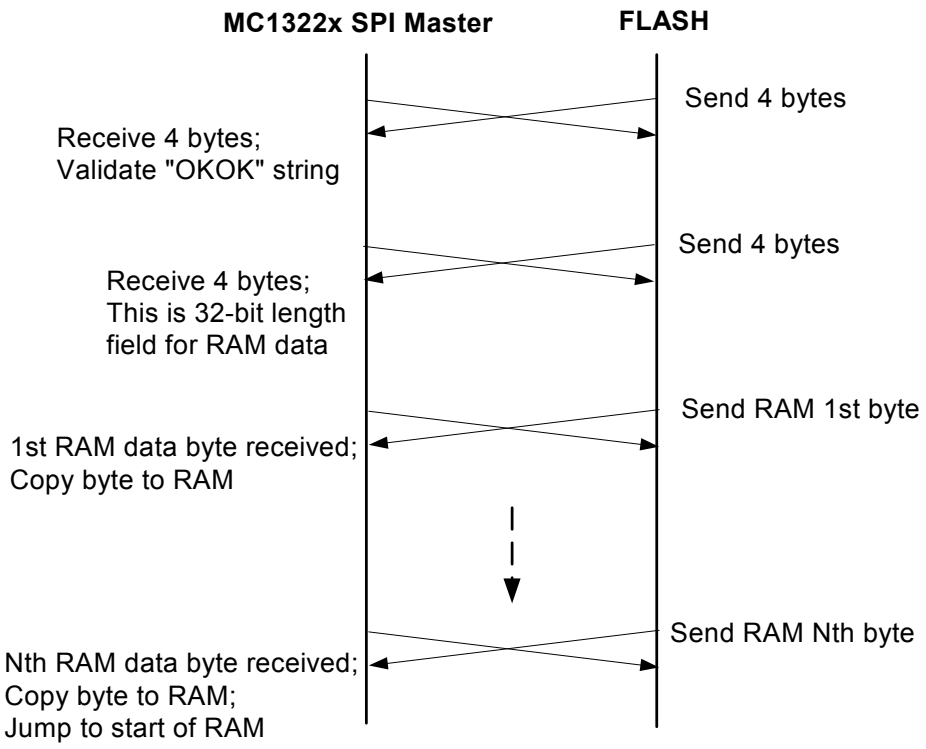



**Figure C-3. MC1322x SPI Master and external FLASH Boot Sequence**

### C.4.4 MC1322x Is SPI Master Data Format

Similar to other sequences, the 4-byte length field and RAM data are supplied in little-endian format.

## C.5 Booting from $I^2C$ (Boot from External Serial EEPROM)

If it has been determined that an external FLASH does not contain valid code (or is non-existent), the bootstrap will finally try to boot from a serial EEPROM through the $I^2C$ port. The port is configured as bus master

### C.5.1 Procedure

The $I^2C$ boot procedure includes:

1. The $I^2C$ signals get initiated as:
   - GPIO12 becomes I2C_SCL input/output
   - GPIO13 becomes I2C_SDA input/output
2. The MC1322x configures the $I^2C$ as master.
3. The MC1322x reads the first four bytes from the external $I^2C$ slave (EEPROM) and validates that the contents are the ASCII string “OKOK” (addr 0x0 = “O”, addr 0x1 = “K”, addr 0x2 = “O”, addr 0x3 = “K”)

**NOTE**

- The first byte of the $I^2C$ slave address is the designated as the Device Select Code or Device Address. The MC1322x addresses the EEPROM with the following described content. **The EEPROM must respond to this addressing byte**.

  It consists of the following fields:

  - Device Type Identifier [B7:B4] = 4b1010; for a serial EEPROM
  - Lower device address or chip enable [B3:B1] = 3b000; device must wired to be addressed at this lowest address.
  - R/W bit [B0]; programmed as read

- If any other content is read, the EEPROM is assumed to be corrupted or not present. The boot flow then moves to an endless loop. Either a debug port intervention or reset is required to recover control.

4. Upon reading the validation code, the bootstrap will load the subsequent four bytes (32-bit, little-endian), and use this as the length field indicator for how many bytes to load into RAM.
5. The MC1322x continues reading bytes from FLASH and loading them into RAM starting with the first RAM address (0x0040_0000).

Once all code is loaded into RAM , the bootstrap will jump to the first RAM address (0x00400000) and continue from there.

### C.5.2 MC1322x Is $I^2C$ Master Data Format

Similar to other sequences, the 4-byte length field and RAM data are supplied in little-endian format.

## C.6 Information Available to the User Program

After the boot process completes and the user program starts to run, the following information is available in the CPU registers:

- The boot source in r0 - The boot source can take one of the values:
  - 0 for secured internal flash
  - 1 for non-secured internal flash
  - 2 for UART
  - 3 for SPI slave
  - 4 for SPI master
  - 5 for $I^2C$
- The program size in r1

**NOTE**

Bootstrap does not change the ARM mode. The boot source and the program source and size will be returned in r0 and r1 of the current ARM mode. So, if the boot process occurs after a POR event the ARM will be in Supervisor mode. If the boot process occurs after a jump to 0x00000000 event, the ARM will be in the mode set before the jump instruction.

## C.7 Loading a Valid FLASH Image

If the MC1322x has booted from an alternative port (not from FLASH), the application loaded via the alternative port is commonly used to write a valid image to FLASH. It is the user's responsibility to implement this function. Some guidelines include:

- Be sure FLASH is erased before attempting to write new data
- The FLASH image can be sourced from any alternative port
- Verify all written data to FLASH
- Write the "OKOK" or "SECU" validation code ONLY after all FLASH has been written and verified.
- There is an NVM driver available to assist in writing this application.