MAC7200RM
Rev. 2
04/2007

MAC7200 Microcontroller Family Reference Manual

Devices Supported:

PAC7202 PAC7212
MAC7242

PAC7201 PAC7211
MAC7241

This document covers the following mask sets:

MAC72x2 – 0M34A, 1M34A, 0M84D, 1M84D

MAC72x1 – 0M19G

Contents

Chapter 2
Modes of Operation

Chapter 3
Low Power Modes

Chapter 4
Signal Description

Chapter 5
System Clock Description

Chapter 6
Resets

Chapter 7
Exceptions

Chapter 8
Debug

Chapter 9
Device Memory Map

Chapter 10
ARM7TDMI-S Core

Chapter 11
A7S Nexus3 Module

Chapter 12
Enhanced DMA Controller (eDMA) Module

Chapter 13
Miscellaneous Control Module (MCM)

Chapter 14
SPP Interrupt Controller Module for ARM (SPP_INTC_ARM)

Chapter 15
MAC7200 Crossbar Switch (AXBS)

Chapter 16
AHB to IPI Bridge (AIPS)

Chapter 17
External Bus Interface (FlexBus)

Chapter 18
FLASH (H7Fb) and FLASH Controller (PFLASH)

Chapter 19 SRAM and SRAM Controller

Chapter 20
Boot Assist Module (BAM)

Chapter 21
IPI Subsystem (IPSS)

Chapter 22
Dual-Output Voltage Regulator (VREG_HIP7A)

Chapter 23
Clock and Reset Generator (CRG)

Chapter 24
Oscillator (OSC)

Chapter 25
System Service Module (SSM_MAC7202)

Chapter 26
Periodic Interrupt Timer (PIT_RTI)

Chapter 27
Enhanced Direct Memory Access (DMA Channel MUX)

Chapter 28
FlexCAN2

Chapter 29
Inter-Integrated Circuit Bus Controller Module (I2C_DMA)

Chapter 30
Deserial Serial Peripheral Interface (DSPI)

Chapter 31
Enhanced Serial Communications Interface (eSCI)

Chapter 32
Modular I/O Subsystem (eMIOS)

Chapter 33
A/D Converter (ATD)

Chapter 34
Port Integration Module (PIM_MAC7202)

Chapter 35
Test Controller (PTI)

Appendix A
Electrical Characteristics

Appendix B
Mechanical Information

Figures

Tables

Preface

This reference manual provides information about the MAC7200 family of microcontroller devices, which are made up of standard System-on-a-Chip modules and an ARM7TDMI-S™ processor core.

Document Structure

This document is part of the documentation needed to complete a design using a MAC7200 family device. A complete set of device manuals also includes the ARM7TDMI-S core manuals:

- *ARM Architecture Reference Manual* (ARM DDI-0100)
- *ARM7TDMI-S (Rev 4) Technical Reference Manual* (ARM DDI 0234A)
- *MAC7200 Microcontroller Family Hardware Specifications* (MAC7200EC)

NOTE

The document MAC7200EC identified above is not currently available. Preliminary electrical and mechanical specifications for the MAC7200 family can be found in this reference manual in Chapter 4, “Signal Description”, Appendix A, “Electrical Characteristics” and Appendix B, “Mechanical Information”.

How To Use This Document

If the reader is new to the MAC7200 family of devices, it is recommended that the following list of sections be read before bringing up a MAC7200 family device:

- Chapter 1, “Introduction” — Describes the features of the MAC7200 family.
- Chapter 4, “Signal Description” — Describes the functionality of MAC7200 family device pins
- Chapter 5, “System Clock Description” — Describes clock generation and distribution to modules on MAC7200 family devices.
- Chapter 6, “Resets” — Describes the reset functionality of the MAC7200 family.
- Chapter 7, “Exceptions” — Describes the system and interrupt exceptions of the MAC7200 family.
- Chapter 2, “Modes of Operation” — Describes the operational modes of the MAC7200 family.
- Chapter 9, “Device Memory Map” — Describes the memory map of the MAC7200 family devices in various operating modes.

If the functionality of a particular peripheral is of interest, refer to the appropriate module description (12.2, “The SPP DMA Controller Module (SPP_DMA2)” through Chapter 34, “Port Integration Module (PIM_MAC7202)”).

Conventions

The following table gives conventions for terms used throughout this document.

Table i. Conventions

Terms	Description
logic level one	The voltage level that corresponds to a Boolean true (1) state.
logic level zero	The voltage level that corresponds to a Boolean false (0) state.
ACTIVE_HIGH	Names for signals that are active high are shown in uppercase text without an overbar. Signals that are active high are referred to as asserted when they are logic 1 and negated when they are logic 0.
$\overline{\text{ACTIVE\_LOW}}$	A bar over a signal name indicates that the signal is active low. Active-low signals are referred to as asserted when they are logic 0 and negated when they are logic 1.
asserted	Signal is in the active logic state. In active high logic, the signal is asserted when it changes to logic level one; in active low logic, the signal is asserted when it changes to logic zero.
negated	Signal is in the inactive logic state. In active high logic, the signal is negated when it changes to logic level zero; in active low logic, the signal is negated when it changes to logic level one.
set	To establish logic level one on a bit or bits
clear	To establish logic level zero on a bit or bits
0x0000	Hexadecimal numbers
0b0000	Binary numbers
n	Indicates a numeric place holder. In register field contexts, indicates a value that may be written or read. In register names, indicates any one of a set of multiple, identical registers. For example, UCCR*n* indicates a reference to any one of the eMIOS Channel Control Registers, UCCR0 through UCCR15.
x	In certain contexts, such as bit or signal encoding, this indicates a don't care. For example, if a four-bit binary field is represented as 0bx001, the state of the first bit is a don't care. In other contexts, such as module or register names, this is a place holder for a letter to designate a module instantiation. For example, eSCI_*x* indicates a reference to either eSCI_A or eSCI_B.
b	Bit place holder
Byte	8 bits
Half-word	16 bits
Word	32 bits

Terminology

The following table lists definitions for abbreviations and names used throughout this document

Table ii. Terminology

Terms	Description
Active Message Buffer	A Message Buffer is said to be “active” if it can participate in the current matching or arbitration process. An Rx MB with a ‘0000’ code is inactive. Similarly, a Tx MB with a ‘1000’ code is also inactive. An MB is temporarily deactivated when the CPU writes to the C/S field (see Section 28.2.7.6.1, “Message Buffer Deactivation).”
ADB	Allowable disconnect boundary
ADC	Analog to Digital converter: A module to convert analog signals into digital (binary) values
AIPS	AMBA™ to IPS interface unit
ALC	Amplitude Limitation Control
ATD	Analog-to-digital (converter). Frequently used synonymously with ADC
ATM	Asynchronous transfer mode
ATMU	Address translation and mapping unit
Auxiliary Port	Refers to Nexus auxiliary port. Used as auxiliary port to the IEEE 1149.1 JTAG interface.
Baud Rate	Rate of data transmission in bits per second.
BD	Buffer descriptor
BDM	Background Debug Module
BIU	Bus Interface Unit, contains all system level customization required to make the H7Fb module part of an SoC.
Branch Trace Messaging (BTM)	Visibility of addresses for taken branches and exceptions, and the number of sequential instructions executed between each taken branch.
Breakpoint	Processor is halted when all previous instructions are retired and just prior to any architectural state change by the instruction associated with a pre-selected address.
CAN	Controller Area Network, a serial communication protocol.
CFM	Common Flash Module. Acronym used throughout this document to reference the Flash memory module. The CFM includes the Common Flash bus interface, IP bus interface, Flash command controller, Flash memory controller, and Flash arrays.
Clock Phase	Determines when the data should be sampled relative to the active edge of SCK
Clock Polarity	Determines the idle state of the SCK signal.
CM	Clock Monitor
coherency / coherent access	Coherent access is used to indicate an action to guarantee data consistency, preventing data from being accessed simultaneously using different methods in such a way that it is not completely updated before being used.
Command Write Sequence	A three-stop command instruction sequence to program, erase, or verify the Flash memory.
CP	Clock Prescaler
CPI	CAN Protocol Interface, a FlexCAN sub-module containing the CAN protocol engine.
CPU	Central Processor Unit
CRC	Cyclic Redundancy Check.

Table ii. Terminology (Continued)

Terms	Description
CRG	Clock and Reset Generator module
CS	Chip Select. In Master Mode, the CS signal is used to select which slave device to talk to.
CSI	Combined Serial Interface. DSPI configuration that alternates DSI and SPI frames.
CSM	Conversion state machine
DAC	Digital to Analog Converter: Converts a binary value into a voltage
DAIC	Double Action Input Capture
DAOC	Double Action Output Compare
Debug Mode	This is a system mode intended for debugging operations. When this mode is triggered, a global Debug Mode Request signal is sent to all modules, so that they can prepare themselves with debugging capabilities.
Deserialize	To convert data from a serial format to a parallel format.
DMA	Direct memory access
DMA Mux	Direct Memory Access Multiplexer module
DMADC1032	ATD with DMA interface
Dominant Bit	A dominant bit wins the arbitration on the CAN bus. It is transmitted as ‘0.’
Doze Mode	This is a system low power mode in which the CPU bus is kept alive and a global Doze Mode request is sent to all peripherals asking them to enter low power mode. Typically, when Doze Mode is requested, each peripheral can be enabled individually to enter or not low power mode.
Drain	To remove entries from a FIFO by software or hardware.
DSI	Deserial Serial Interface. DSPI configuration that serializes and deserializes registers or purpleline signals to allow for pin reduction.
DSI Frame	Collection of serialized or deserialized pin states or register bits transferred over the serial link
DSPI	(Deserialized) Serial Peripheral Interface
ECC	Error Correction Code. Internally used to correct single bit errors, or detect double errors within a 64 bit double word.
ECLK	E clock
eDMA	Enhanced Direct Memory Access controller module
EmbeddedICE	The ARM7 EmbeddedICE debug module. This module integrated with each ARM7 core provides all static (core halted) debug functionality. This module is compliant with Class1 of the IEEE-ISTO 5001 standard.
eMIOS	Enhanced Modular Input/Output Subsystem
EOQ	End of Queue
Erase State	Flash array bit state that reads as a “1.”
eSCI	Enhanced SCI module with LIN hardware and DMA support

Table ii. Terminology (Continued)

Terms	Description
FC	Flash Core, contains all addressable non-volatile storage elements. This includes Low Address Space, Mid Address Space, High Address Space, and Shadow Space. Also known as Flash Array.
Field	Two or more register bits grouped together.
Fill	To add entries to a FIFO by software or hardware.
Flash Array	A non-volatile SuperFlash® memory array used to build the Program Flash blocks, which includes a Flash memory core with built-in high voltage generation and parametric features.
Flash Logical Page	4096 bytes of contiguous Flash memory consisting of two interleaved Flash physical blocks representing the smallest section of the Flash memory that can be erased.
Flash Logical Sector	Section of contiguous Flash memory that can be protected from program, erase, and unauthorized access.
Flash User Mode	Flash module operations defined for User/Normal mode.
Frame	The data content of a serial transmission. Also referred to as DSPI Data.
FSM	Finite State Machine
GPIO	General purpose Input/Output
H7Fb	HiP7a Low Cost Flash EEPROM Module.
Hard Reset	Reset coming from external pin and/or following power-on. It resets everything.
HAS	High Address Space. If it exists, it contains 128k Byte subdivisions of the H7Fb array.
Host	Refers to the MCU or other bus master module
IEEE-ISTO 5001	Consortium & standard for real-time embedded system design. World wide Web documentation at http://www.nexus5001.org/.
IIB	Internal Interface Bus
input capture	Sampling of a time base value upon the occurrence of an input signal transition.
IPI	Internal Peripheral Interface - a Freescale peripheral slave bus
IPF	Input Programmable Filter
IPM	Input Period Measurement
IPS	Intelligent Peripheral Subsystem bus interface
JTAG Compliant	Device complying to IEEE 1149.1 JTAG standard
JTAG IR & DR Sequence	JTAG Instruction Register (IR) scan to load an opcode value for selecting a development register. The selected development register is then accessed via a JTAG Data Register (DR) scan.
LAS	Low Address Space. 256KByte in size. Various configurations exist for Low Address Space. May also be 128KByte in size if total memory size is 128KByte (i.e. No MAS or HAS).
LC	Loop Control
LIN	Local Interconnect Network – A protocol for low-cost automobile networks
LIN FSM	LIN Finite State Machine – The control logic of the LIN hardware

Table ii. Terminology (Continued)

Terms	Description
LSB	Least Significant Bit
LVR	Low Voltage Reset
MAC	Multiply and accumulate
MADD	Multiply-and-Add unit which is responsible for the raw result adjustment
MAS	Mid Address Space. 256KByte in size. If it exists, it may contain 4 x 64KByte subdivisions of the H7Fb Array, or 2 x 128KByte subdivisions.
match	Match an event that occurs when the value of a match register becomes equal to the value of the selected time-base.
MC	Modulus Counter
MCU	Microcontroller Unit.
Message Buffer (MB)	Internal FlexCAN data structure containing bytes received or to be transmitted to the CAN line, as well as information about this data.
MI	Memory Interface, contains all state machines and control logic needed for operation of the H7Fb module.
MSB	Most Significant Bit
MUX	Multiplexer
NSM	New command word state machine
NVM	Non-Volatile Memory.
OpAmp	Operational Amplifier
OPWFM	Output Pulse Width and Frequency Modulation
OPWM	Output Pulse Width Modulation
OPWMC	Center Aligned Output Pulse Width Modulation
OSCCLK	Oscillator clock
Output Compare	The modification of an output signal due to a time base match.
Ownership Trace Messaging (OTM)	Visibility of process/function that is currently executing.
PCS	Peripheral Chip Select
PEA	Pulse/Edge Accumulation
PEC	Pulse/Edge Counting
PFLASH	Platform Flash Controller
Pipeline	Act of initiating a bus cycle while another bus cycle is in progress. Thus the bus can have multiple bus cycles pending at one time.
PIT	Periodic Interrupt Timer
PLL	Phase Locked Loop
POR	Power on Reset

Table ii. Terminology (Continued)

Terms	Description
Program State	Flash array bit state that reads as a “0.”
Public Messages	Messages on the auxiliary pins for accomplishing common visibility and controllability requirements
QDEC	Quadrature Decode
RC	Resistor-Capacitor
Receive or RX FIFO	First-In-First-Out buffer for received data
Recessive Bit	A recessive bit loses the arbitration on the CAN bus. It is transmitted as ‘1.’
Reset Sequence	Coming out of reset, the CFM will read the Flash configuration field and load specific registers.
RQB	Interrupt Request Bus
RSD	Redundant Signed Digit: architecture how the ATD does the conversion
RSM	Result saving state machine
RTI	Real Time Interrupt - A timer with an independent clock which can run in system Doze or Pseudo-Stop mode, and can be used for system wakeup.
RWCB	Read/Write Control Bus
RWW	Read While Write.
SAIC	Single Action Input Capture
SAOC	Single Action Output Compare
SAR	Successive Approximation Register: A method to adjust a reference voltage to an input voltage
SCI	Serial Communications Interface
SCK	Serial Communications Clock
Sclock	Serial clock (FlexCAN). This is the clock obtained by dividing the clock feeding the CAN engine (either oscillator or bus clock) by a prescaler factor. The Sclock period defines the time quantum for CAN protocol timing.
SCM	Self Clock Mode
Serialize	To convert data from a parallel format to a serial format.
Slave	A bus slave is a device that responds to a bus transaction, but never initiates a cycle on the bus.
SMB	Serial Message Buffer, an internal buffer not accessible by the end user.
SoC	System-on-a-chip.
Soft Reset	Global reset typically used by peripherals to re-initialize some of its registers, but not all of them.
SPI	Serial Peripheral Interface
SS	Slave Select. Signal from the SPI master to the SPI slave indicating which SPI slave device the Master want to communicate with.
Stop Mode	This is a system low power mode in which all MCU clocks are stopped for maximum power savings. Typically, when Stop Mode is requested, each module will put itself in a known state and then send a Stop Acknowledge signal to inform the CPU that it can stop the clocks.

Table ii. Terminology (Continued)

Terms	Description
SWT	Software Watchdog Timer
SYSCLK	System clock (f_{SYS}) is the clock used by the core CPU (peripherals operate at $f_{SYS} \div 2$)
Time Quantum	This is equal to the Sclock period. It is the minimum time period used to compose the CAN protocol bit timing.
Transaction	A bus transaction consists of an address transfer (address phase) and one or more data transfer(s) (data phase).
Transfer Code (TCODE)	Message header that identifies the number and/or size of packets to be transferred, and how to interpret each of the packets.
Transfer Format	The combination of SCK polarity, SCK phase, data MSB/LSB first, and associated CS signal timing during a serial transmission
Transmit or TX FIFO	First-In-First-Out buffer for transmit data
TUE	Total Unadjusted Error: difference between the expected (ideal) conversion result and the result delivered by the ATD
UC*n*	Unified channel *n*, submodule that performs timed input or output functions supported by the eMIOS
VCO	Voltage Controlled Oscillator
VREG	Voltage regulator
Watchpoint	A Data or Instruction Breakpoint which does not cause the processor to halt. Instead a pin is used to signal that the condition occurred. A Watchpoint Message is also generated.
WPTA	Windowed Programmable Time Accumulation

Register Descriptions

Each peripheral module chapter (12.2, “The SPP DMA Controller Module (SPP_DMA2)” through Chapter 34, “Port Integration Module (PIM_MAC7202)”) contains a register description subsection that details the location and definition of the user-accessible control and status bits and fields for the peripheral. All register descriptions in this manual use bit 31, 15 or 7 (depending on the register size) to represent the most significant bit and bit 0 to represent the least significant bit. Refer to Section 9.8, “Accessing registers” for details on how registers may be accessed.

Each register description subsection includes a register diagram figure showing bit field mnemonic names and locations followed by a table containing the full name for each bit field, a short description of operational characteristics and exact bit value definitions. The figure and tables below show an example of the format used for register figures, field descriptions and the conventions used to specify bit fields.

Row Label	Column Content
xx	Bit number, specifies the location of the bit or field within the register.
R	Behavior for read accesses. If named, the description table below the register diagram specifies the definition. If zero, the bit will always read as zero. If shaded, the read value is undefined and the bit position is reserved and must be ignored for future compatibility.
W	Behavior for write accesses. If named, the description table below the register diagram specifies the definition. If shaded, the bit position is reserved and must be written as zero for future compatibility.
Reset	Bit state immediately after a reset operation. Zero or one indicates how the bit state is affected by reset, an emdash indicates that the bit state is not affected by reset.
Reg Addr	Specifies the address of the most significant byte of the register (for a 16- or 32-bit register). Base is the module address as specified in Chapter 9, “Device Memory Map’, Table 9-4 on page 9-108. Offset is a hexadecimal number in the format 0x0000, or a formula used to calculate the offset, that is added to Base to calculate the address of the register.

Table iii. Register Diagram Conventions

Column Label	Row Content
Bits	Bit number, specifies the exact location of the bit field within the register. Always listed from the most significant bit in the register to the least significant. For bit fields, the most significant and least significant bit numbers are shown.
Name	Mnemonic name of the bit or field. For bit fields, the mnemonic is followed by bracketed numbers specifying the size of the field.
Description	Full name of the bit or field. Includes a short definition of operational characteristics. Often contains cross-references to detailed functional descriptions of module behaviors that are affected by a control field or that are reflected in a status field. Followed by paragraphs listing definitions for each possible value of the bit or field. For more complex definitions, or where bits or fields interact with other bits or fields in various modes, tables are often embedded within the field description cell for more detailed information. 0 single-bit value definitions 1 single-bit value definitions 0..0 first in a list of multi-bit field value definitions . . . 1..1 last in a list of multi-bit-field value definitions

Table iv. Register Field Description Conventions

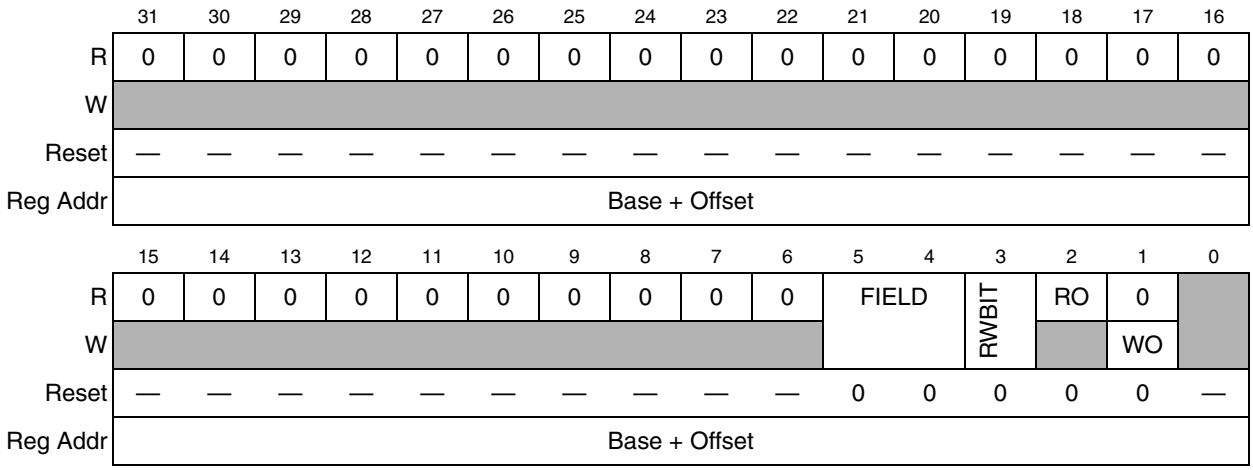


Figure i. Example Register (EXREG)

Table v. EXREG Field Descriptions

Bits	Name	Description
15–6	—	Reserved; always reads as zero, must be written as zero.
5–4	FIELD[1:0]	Bit field. This is a two-bit read/write field. 00 Behavior when field is written as 0b00, or status indicated when read as 0b00 01 Behavior when field is written as 0b01, or status indicated when read as 0b01 10 Behavior when field is written as 0b10, or status indicated when read as 0b10 11 Behavior when field is written as 0b11, or status indicated when read as 0b11

Table v. EXREG Field Descriptions (Continued)

Bits	Name	Description
3	RWBIT	Read/write bit. This is a writable control bit that is used to specify certain behavior of the module. Reading the bit position will return the last value written. 0 Behavior when bit is cleared 1 Behavior when bit is set
2	RO	Read-only. Normally used for status bits that reflect operating characteristics at the time of the read access. Writes are ignored, but for future compatibility zero should always be written. 0 Status definition if bit is clear 1 Status definition if bit is set
1	WO	Write-only bit. Normally used for control bits that trigger an event when written as one, but always read as zero. 0 Writing zero has no effect 1 Writing one triggers event
0	—	Undefined bit. Normally indicates bits that are used only for factory testing and must not be modified by customer code, and whose read contents are indeterminate. Writing non-zero values may cause erratic device behavior.

Revision History

Content Changes by Document Version

Version No. Release Date	Description of Changes	Page Numbers
Rev. 0 Nov-06	First customer release of Preliminary version.	All
Rev. 1 Mar-07	First public release of Preliminary version.	All
Rev. 2 Apr-07	Table A-5 MAC7200 Family Device Operating Condition. C2 Digital Logic Supply Voltage V_{DD}15, changed min limit from 1.35 to 1.45 Table A-12 Vreg operating conditions. G2 Output Voltage Core (1.5 V) changed min from 1.35 to 1.45	861 868

Chapter 1
Introduction

1.1 Overview

Designed for automotive applications, MAC72xx devices are members of a family of 32-bit Flash-based microcontrollers. The MAC7200 family's pin compatibility enables users to choose between different memory and peripheral options for scalable designs. All MAC72xx devices are composed of a 32-bit central processing unit (ARM7TDMI-S), up to 512K bytes of embedded Flash memory, as well as additional memory blocks (4 x 16K bytes) to support software emulation of EEPROM and (32K bytes) Bootloader code. The devices also have up to 32K bytes of System RAM and are implemented with an eDMA (enhanced Direct Memory Access) controller to enable transfers between memories and standard on-chip peripherals.

The peripheral set includes up to two asynchronous serial communications interfaces (eSCI), up to three serial peripheral interfaces (DSPI), one I^2C™ bus, a configurable 16-bit Timer (eMIOS) enabling functions such as dual action Capture/Compare and output Pulse Width Modulation, up to sixteen 12-bit analog-to-digital (ATD) converter channels, and two CAN 2.0 A/B software compatible modules (FlexCAN). In addition, a large number of General Purpose Input Output (I/O) pins are also offered. All of these I/O pins are bidirectional, and are available with interrupt capability. The internal data paths between the DMA, memory, core and the peripherals are all 32-bits wide, further improving performance for 32-bit applications.

The inclusion of a Phase Locked Loop (PLL) circuit allows power consumption and performance to be adjusted to suit operational requirements.

The MAC72xx device members are offered in 100-pin LQFP or 144-pin LQFP. The devices can be operated over a temperature range of -40°C to 150°C junction temperature.

NOTE

The 144LQFP packages are not currently qualified; any future qualification will be based upon customer demand and will be subject to specified leadtimes. This package option is supported for limited samples only, and does not include burn-in testing.

1.2 Features

- 32-bit ARM7 TDMI-S RISC Core
  - — Up to 70MHz operating frequency
  - — Efficient code density through 16-bit instructions.
  - — Alternate general purpose registers.

  - Byte (8-bit), Halfword (16-bit), Word (32-bit) data types supported.
  - Cores and Memory connected using high performance AMBA AHB bus.
  - Integrated E-ICE module for debug
- Memory size
  - 320k Bytes (MAC72x2) or 512k Bytes (MAC72x1) of wide accessed Program Flash EEPROM.
    - 4 Flash page buffers all 128-bits wide and individually configurable as either Instruction or Data buffers.
    - Page buffers can be configured to enable fetch ahead and either least recently used or counter based buffer replacement schemes.
    - Program & erase operations controlled by state machine.
    - Internally generated program and erase voltages.
    - ECC enabled array with 2-bit error detection, 1-bit error correction providing transparent operation.
    - 64-bit minimum write size.
    - Flash configuration suitable for EEPROM emulation.
    - Read or Program/Erase access on flash partition basis.
    - Flash BIU support for access protection for User/Supervisor mode and Instruction/Data accesses.
    - Protection violation flag.
    - Flash Lockout recovery mechanism though JTAG interface.
    - 100K W/E endurance.
    - 20 year data retention.
  - 20k Byte (MAC72x2) or 32k Bytes (MAC72x1) RAM
    - Single cycle accesses to RAM for Byte, Halfword and Word reads and writes.
    - ECC enabled array with 2-bit error detection, 1-bit error correction.
- Interrupt Controller
  - 64 vectored interrupt sources.
  - Interrupt sources available from internal peripherals, eDMA controller, software watchdog timer and external sources.
  - Supports 16 interrupt levels (0-15) with 64 priorities per level (1-64), to allow maximum flexibility in configuring the system. Every interrupt source can be programmed to any interrupt level. Priorities within a level are hard-coded in the MCU.
  - Multiple level interrupt nesting.
  - Hardware support for first nesting level.
  - Normal and Fast interrupts supported with software programmability of sources for both Fast and Normal Interrupts
- Enhanced Direct Memory Access Controller (eDMA)

  - DMA transfers possible between system memories, SPIs, SCIs, I^2C, ATD, eMIOS and General Purpose I/Os
  - Programmable DMA Channel Mux allows assignment of any DMA source to any of the 16 available DMA channels.
  - All DMA transfers use dual address format.
  - Programmable Transfer Control Descriptor stored in local DMA memory.
  - Programmable Source and Destination address with configurable offset.
  - Independent 32-bit Minor and 16-bit Major loop counters for “nested” transfers.
  - Different final Source and Destination addresses allow circular Queue operation.
  - Programmable priority levels for each channel.
  - Bandwidth control for each channel.
  - Programmable transfer sizes through Major and Minor loop counters.
  - Independently Programmable read/write sizes.
  - Periodic triggering of up to 8 channels.
  - Round Robin channel prioritization
  - Scatter-Gather functionality
  - Inner Loop channel pre-emption
  - Channel to Channel linking
  - Software or Hardware start
- Analog-to-Digital Converter
  - Up to 16 analog input channels.
  - 12-bit resolution 9-bit accuracy.
  - 2µS minimum conversion time.
  - Internal sample and hold circuitry.
  - Pre-measurement discharge of internal Sample and Hold circuit possible for all channels.
  - Programmable input sample time for various source impedances.
  - Queued conversion sequences supported by DMA controller.
  - Analog inputs configurable as external sample triggers.
  - On-chip timer triggers for sampling.
- Two 1M bit per second, CAN 2.0 A/B software compatible modules (FlexCAN)
  - Full implementation of the CAN 2.0 protocol specification.
  - Programmable bit rate up to 1Mbps.
  - 32 flexible Mail Boxes of 0-8 bytes data length on all modules.
  - All Mail Boxes configurable for either Rx/Tx.
  - Unused Mail Boxes space can be used as general purpose RAM.
  - Supports standard or extended messages.
  - “Time Stamp”, based on a 16-bit free-running counter.
  - Maskable interrupts.

  - Programmable I/O modes.
  - External transceiver assumed.
- Enhanced Modular I/O Subsystem features (eMIOS)
  - 8 unified channels (MAC72x2) / 16 unified channels (MAC72x1), with every channel able to provide all timer functions and modes.
  - All channels can be enabled for eDMA service.
  - Channels can be individually disabled to assist with power saving.
  - Two or three 16-bit counter buses (A and B, or A, B, and C) for sharing time base around the module.
  - One global prescaler and an individual prescaler available for each channel.
  - Modulus counter mode on all channels.
  - Single action input capture or output compare modes on all channels.
  - Input pulse width and period measurement modes on all channels.
  - Double action output compare mode on all channels.
  - Output pulse width and frequency modulation modes on all channels.
  - Output pulse width modulation modes on all channels.
  - Center aligned output pulse width modulation with dead time insertion modes on all channels.
  - Pulse or edge accumulation and counting modes on all channels.
  - Windowed programmable time accumulation mode on all channels.
  - Quadrature decode modes on all channels.
  - General purpose I/O available on unused eMIOS pins.
  - Center-aligned PWM with dead-time insertion
- Three Serial Peripheral Interface features (DSPI)
  - Full Duplex, Synchronous Transfers.
  - Master or Slave Operation.
  - Programmable Master Bit Rates.
  - Programmable Clock Polarity and Phase.
  - End-of-Transmission Interrupt Flag.
  - Programmable transfer Baud rate.
  - Programmable data frames from 4-bits to 16-bits.
  - Up to 6 chip select lines enable 64 external devices to be selected using external multiplexing from a single DSPI.
  - 8 Clock and Transfer Attributes registers.
  - Chip Select Strobe available as alternate function on one of the Chip Select pins for de-glitching.
  - Two dedicated DMA request lines on each peripheral for receive and transmit data.
  - FIFO for buffering up to 4 transfers on the transmit and receive side.
  - Queueing operation possible through use of the DMA controllers channels.

  - General purpose I/O functionality on pins when not used for SPI
- Two Asynchronous Enhanced Serial Communications Interface features (eSCI)
  - Standard Non return-to-Zero (NRZ) Mark/Space Format.
  - Full-Duplex Operation.
  - Software Selectable Word Length (8-bit or 9-bit words).
  - 10/11 or 13/14 bit Break Character possible.
  - 13-Bit Programmable Baud-Rate Modulus Counter.
  - Separately Enabled Transmitter and Receiver.
  - Separate receiver and transmitter CPU interrupt requests.
  - Programmable transmitter output polarity.
  - Two Receiver wake-up methods.
  - Interrupt-driven operation with eight flags.
  - Receiver framing error detection.
  - Hardware Parity Checking.
  - 1/16 bit time noise reduction.
  - LIN Master mode state machine
    - Supports generation of LIN message header.
    - Detection and flagging of LIN errors.
  - Two DMA request lines on each peripheral for receive and transmit data.
- Inter IC Bus module features (IIC)
  - Compatibility with I^2C Bus standard.
  - Two wire bi-directional serial bus for on board communications.
  - Multimaster operation.
  - Software-programmable for one of 256 different serial clock frequencies.
  - Software-selectable acknowledge bit.
  - Interrupt-driven byte-by-byte data transfer.
  - Arbitration-lost interrupt with automatic mode switching from master to slave.
  - Calling address identification interrupt.
  - Start and stop signal generation/detection.
  - Repeated START signal generation.
  - Acknowledge bit generation/detection
  - Bus-busy detection.
  - Two DMA request lines to receive and transmit data
- Oscillator
  - Low power Amplitude Level Control (ALC) Oscillator.
  - Selectable oscillator mode at powerup or after a loss of clock
- Clock generation

- Clock generation and Reset control performed in CRG.
- Phase-locked loop clock frequency multiplier.
- Self clocking mode available in absence of external clock.
- Software Watchdog Timer (SWT) watchdog.

- Periodic Interrupt Timer (PIT) Module
  - Independent timeout period for each timer.
  - Four 32-bit general purpose timers, configurable to generate DMA trigger pulses.
  - Four dedicated 32-bit timers to generate DMA trigger pulses.
  - Two 32-bit timers that can be configured to generate ATD trigger pulses.
  - One 24-bit real-time interrupt (RTI) timer.
  - RTI operates from oscillator output clock.
- Miscellaneous Control Module (MCM)
  - Software watchdog timer with programmable system reset or interrupt response.
  - Watchdog with optional Windowed mode.
  - Access address information for faulted memory accesses.
  - NMI configuration
- System Services Module (SSM)
  - System configuration and status.
    - Memory sizes and status
    - Security status
    - Device mode
    - eDMA status
    - Debug Port.
    - Nexus Status.
    - System Reset.
- General Purpose Input/Output
  - Select between GPIO, interrupt or peripheral functionality on a pin-by-pin basis
  - Register interface for both port wide and pin data reads and writes
  - 6 independent 16-bit ports (Port A, B, C, D, F, G), with each pin having the following features:
    - Peripheral or GPIO Mode selection, with up to 3 peripherals per pin
    - Input/Output selection
    - 5V output drive with three selectable slew rates (Disabled, Slow, Fast)
    - 5V digital inputs
    - Selectable pull-up or pull-down
    - Selectable open drain for wired-or connections
    - Selectable interrupt capability (with glitch filtering and interrupt mask)
  - Control of ATD digital inputs (Port E)

  - Selectable pull-up or pull-down
- Control of the TCK, TMS, TDI and TDO pads
  - Input/Output selection
  - 5V output drive with three selectable slew rates (Disabled, Slow, Fast) (TDO only)
  - 5V digital inputs
  - Selectable pull-up or pull-down (TCK, TMS and TDI only)

- Internal Voltage Regulators.
  - On-chip Voltage Regulators generate all necessary internal supply voltages from 5V only input voltage, including Flash, Oscillator, PLL and core supply voltage.
  - Bypass mode allows off-chip supply of all on-chip voltages.
  - Power On Reset (POR) and Low Voltage Reset (LVR) detection with independent flags for full reset source reporting.
- 144-Pin LQFP, 100-Pin LQFP
  - I/O lines with 5V input and drive capability.
  - Programmable pull-up pull-down or no-pull on all port pins.
  - Programmable slew rate on all bidirectional port pins.
  - 5V ATD converter inputs.
  - 1.5V logic supply.
- Development support
  - Real Time Instruction Trace Support via Nexus Class 3.
  - Alternate Nexus port pin position selectable at Reset
  - ARM Embedded ICE debug support on all devices.
  - JTAG Test Access Port (TAP) Interface.
  - Debug mode access to CPU registers.
  - Real Time memory access.
  - Hardware Breakpoints.

Table 1-1. List of MAC72xx Devices

Flash	RAM	Package	Device	FlexCan	ESCI	DSPI	IIC	ATD[1]	eMIOS	Nexus3	I/O[2]
320k	20k	144LQFP	7212	2	2	3	1	1/8	8	Yes	102
		144LQFP	7202	2	2	3	1	1/16	8	Yes	102
		100LQFP	7242	2	2	3	1	1/15	8	Yes	66
512k	32k	144LQFP	7211	2	2	3	1	1/8	16	Yes	102
		144LQFP	7201	2	2	3	1	1/16	16	Yes	102
		100LQFP	7241	2	2	3	1	1/15	16	Yes	66

1. ATD column shows: number of modules/total number of ATD channels.
2. I/O column shows: sum of ports able to act as digital input or output.

- 144 Pin Packages:
  - PAC7211/PAC7212
    - Port A = 16, B = 16, C = 16, D = 14, E = 8, F = 16, G = 16.
    - 101 inputs provide interrupt capability (Ports, $\overline{\text{XIRQ}}$/NMI).
  - PAC7201/PAC7202
    - Port A = 16, B = 16, C = 16, D = 12, E = 16, F = 16, G = 10.
    - 101 inputs provide interrupt capability (Ports, $\overline{\text{XIRQ}}$/NMI).

NOTE

The 144LQFP packages are not currently qualified; any future qualification will be based upon customer demand and will be subject to specified leadtimes. This package option is supported for limited samples only, and does not include burn-in testing.

- 100 Pin Packages:
  - MAC7242/MAC7241
    - Port A = 3, B = 16, C = 0, D = 8, E = 8, F = 16, G = 15.
    - 66 inputs provide interrupt capability (Ports, $\overline{\text{XIRQ}}$/NMI).

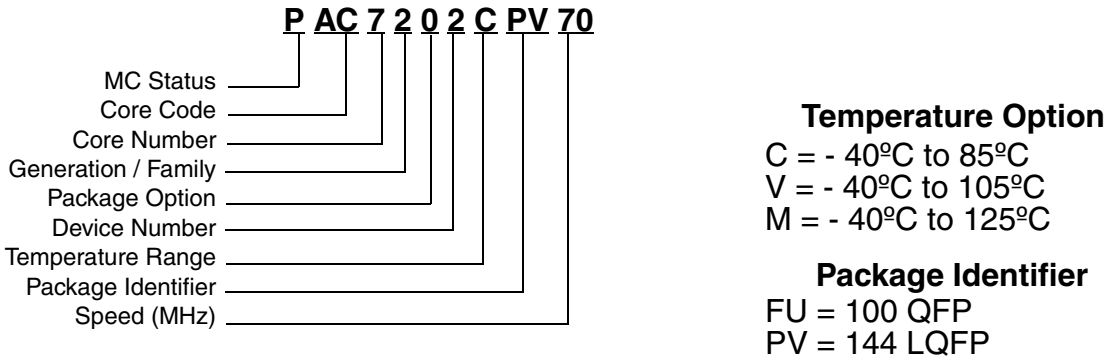


Figure 1-1. Orderable Part Number Example

1.2.1 Performance Summary

Max Processor/System Bus Speed: 70MHz (T=14.3ns)

Max Peripheral Bus Speed: 35MHz (T=28.6ns) (1/2 Max Processor Speed)

Context Switching: Assumed to be about 25 cycles for all ISRs

Table 1-2. Typical System Access Summary

Port	Typical (cycles)	Purpose	Comments
ARM7 Core			
Flash (Port M0=>S0)[1]	RD: 1.2 (per instr.) WR: N/A	Instruction Fetch	
SRAM[2] (Port M0=>S3)[1]	RD: 1 WR: 1	Operand Read/Write	
External Bus (Port M0=>S1)[1]	RD: 2 WR: 2	Operand Read/Write	
Peripheral Bus Periphs (Port M0=>S2)[1]	RD: 3 WR: 4	Peripheral Config	
DMA			
Flash (Port M1=>S0)[1]	RD: 3 WR: 4	Program Flash	
SRAM[2] (Port M1=>S3)[1]	RD: 1 WR: 1	Periph to/from SRAM	
External Bus (Port M1=>S1)[1]	RD: 2 WR: 2	Periph to/from Ext. Memory	
Peripheral Bus Periphs (Port M1=>S2)[1]	RD: 3 WR: 4	Periph to/from SRAM	

1. Port notations refer to Crossbar ports, and are explained in Chapter 15, “MAC7200 Crossbar Switch (AXBS)”.
2. Assumes 0 programmed wait-states.

1.3 Modes of Operation

This section describes the various functional modes of the MAC72xx devices. A functional mode is selected by asserting the MODA and MODB pins (PD1 and PD0, respectively) while the $\overline{\text{RESET}}$ pin is asserted. Because there is no way to determine when these values are latched inside the device, the MODA/MODB pins must be asserted the entire time that the $\overline{\text{RESET}}$ pin is asserted. External (weak) pull-ups and/or pull-downs may be used for this purpose. Table 1-3 shows the values of the MODA/MODB pins for each chip mode. Note that these encodings are only valid when the TEST pin is negated (tied low).

Once a chip mode has been set, it may only be changed by any of the following:

- Securing the flash, and resetting the device. In this case the mode changes from “XXX Unsecured Mode” to “XXX Secured Mode”.
- Unsecuring the flash, and resetting the device. In this case the mode changes from “XXX Secured Mode” to “XXX Unsecured Mode”.
- Changing the **MODA**/**MODB** pins and resetting the device. In this case the mode changes from “XXX Secured/Unsecured Mode” to “YYY Secured/Unsecured Mode”.

Note that resetting the device is mandatory in all three cases.

Table 1-3. Chip Mode Encodings

MODA (PD1)	MODB (PD0)	Flash Secured ?	Chip Mode
0	0	Yes	Single Chip Secured Mode
0	1	Yes	Expanded Chip Secured Mode
1	0	Yes	PBL Secured Mode
1	1	Yes	Reserved
0	0	No	Single Chip Unsecured Mode
0	1	No	Expanded Chip Unsecured Mode
1	0	No	PBL Unsecured Mode
1	1	No	Reserved

1.3.1 Single Chip mode (Unsecured)

- EICE/Nexus/JTAG active
  - Enable TCK, TDI, TDO and DBGEN
- FlexBus available at $2000 0000
- Boot from Flash main array at $0000 0000
  - Shadow Block available at $00f0 0000
- JTAG Lockout Recovery is available

1.3.2 Single Chip mode (Secured)

- No EICE/Nexus/JTAG
  - Disable DBGEN and Nexus clock(s)
- No FlexBus
- Boot from Flash main array at $0000 0000
  - Shadow Block available at $00f0 0000
- JTAG Lockout Recovery is available

1.3.3 PBL Chip mode (Secured)

- Same as Single Chip mode, except boot from the Shadow Block
- No EICE/Nexus/JTAG
  - Disable DBGEN and Nexus clock(s)
- No FlexBus
- Relocate Shadow Block to $0000 0000

- Relocate Flash main array to $2000 0000
  - Shadow Block available at $20f0 0000
- JTAG Lockout Recovery is available

1.3.4 PBL Chip mode (Unsecured)

- Same as Single Chip mode, except boot from the Shadow Block
- EICE/Nexus/JTAG active
  - Enable TCK, TDI, TDO and DBGEN
- No FlexBus
- Relocate Shadow Block to $0000 0000
- Relocate Flash main array to $2000 0000
  - Shadow Block available at $20f0 0000
- JTAG Lockout Recovery is available

1.3.5 Expanded Chip mode (Secured)

- EICE/Nexus/JTAG active
  - Enable TCK, TDI, TDO and DBGEN
- Disable Flash Main Array and Shadow Block access
- Relocate FlexBus to $0000 0000
- Boot from FlexBus

1.3.6 Expanded Chip mode (Unsecured)

- EICE/Nexus/JTAG active
  - Enable TCK, TDI, TDO and DBGEN
- Relocate Flash to $2000 0000
  - Shadow Block available at $20f0 0000
- Relocate FlexBus to $0000 0000
- Boot from FlexBus

See Chapter 9, “Device Memory Map” for more detailed information on how each chip mode affects relocation of system resources.

1.3.7 Low Power Modes

The microcontroller features only a single low power mode:

- DOZE Mode. In this mode, individual peripherals may be shut down to allow a completely customizable power consumption profile tailored to the requirements of the application.

In addition to the above mode, each peripheral may be individually disabled in order to minimize power consumption for those peripherals not used in a particular application.

See Chapter 3, “Low Power Modes” for more detailed information on low power modes.

1.3.8 Debug Mode

In addition to the functional chip modes and low power modes described above, the MAC72xx supports a Debug mode. By placing the system in Debug mode, the following actions occur:

- The ARM7 core is placed into Halt Mode or Monitor Mode, depending on the state of the DCR[4] bit.
- The Real Time Interrupt (RTI) clock is shut off
- The Software Watchdog Timer (SWT) clock is shut off
- All peripheral level debug features are enabled
  - DMA2
  - FlexCAN2
  - DSPI
  - eMIOS
  - ATD
  - MCM (Software Watchdog Timer)

More information on system debug features can be found in Chapter 8, “Debug”.

1.4 Block Diagram

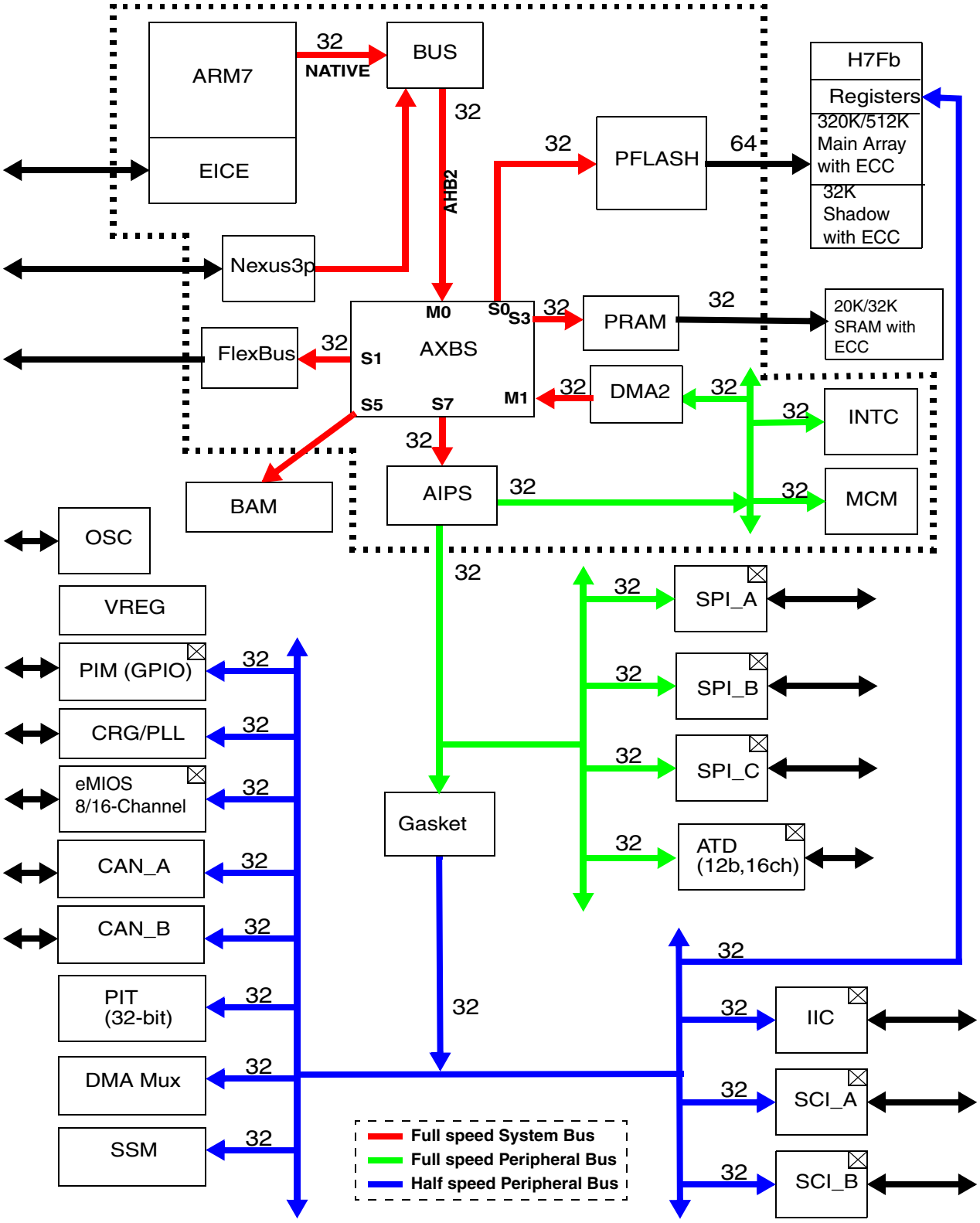


Figure 1-2. MAC72xx Architecture Overview

NOTES:
1 Arrow directions represent bus mastership, not R/W capability. Unless otherwise noted, all buses are R/W
2 Each bus includes a number (8,16 or 32), which represents the width of the data bus
3 The heavy dotted black line represents the boundary of the SPP Platform
4 Modules with a ☒ represent modules with DMA interfaces. All other IPI modules have no DMA connection.

1.5 System Memory Map

Please also refer to Chapter 2, “Modes of Operation” and Chapter 9, “Device Memory Map”.

NOTE

Some system resources may be dynamically relocated after reset. refer to Section 9.16, “Memory Map Relocation for more information on this topic.

Table 1-4. System Memory Maps

	Single Chip Unsecured	Single Chip Secured	Expanded Unsecured	Expanded Secured	PBL Unsecured	PBL Secured	Lockout Recovery
$0000 0000	Flash Main Array	Flash Main Array	FlexBus	FlexBus	Shadow Block	Shadow Block	BAM
$00F0 0000	Shadow Block	Shadow Block					
$2000 0000	FlexBus		Flash Main Array		Flash Main Array	Flash Main Array	Flash Main Array
$20F0 0000			Shadow Block		Shadow Block	Shadow Block	Shadow Block
$4000 0000	SRAM	SRAM	SRAM	SRAM	SRAM	SRAM	SRAM
$6000 0000							
$8000 0000							
$A000 0000	BAM	BAM	BAM	BAM	BAM	BAM	BAM
$C000 0000							
$E000 0000	Peripheral Bus	Peripheral Bus	Peripheral Bus	Peripheral Bus	Peripheral Bus	Peripheral Bus	Peripheral Bus

Chapter 2
Modes of Operation

2.1 Introduction

Devices in the MAC7200 family can operate in several different modes, depending on the particular application and stage of development. In general, there are six different modes available on the MAC7200 family, which are determined by the MODA and MODB pins as well as the security state of the on-chip Flash memory. The selection of a particular mode affects the following device characteristics:

- The memory map for the device. A detailed description of the memory map in each chip mode can be found in Chapter 9, “Device Memory Map”
- Which debug features are enabled or disabled
- Which security features are enabled or disabled

2.2 MCU Hardware Configuration Summary

On the MAC72xx, there are 7 pins which are used to determine the configuration of the device at reset, as follows:

- MCU Mode (MODA, MODB)
- Oscillator Type ($\overline{\text{XCLKS}}$)
- Nexus Port (NEXPORTSEL, NEXPRESENT)
- External Bus Port Size and Auto Acknowledge (for those devices with an external bus) (PORTSIZE, AUTOACK)

Note that all of the hardware configuration is done during Reset with the TEST pin held low. As such, the values of the pins are latched on the rising edge of the active low device Reset signal $\overline{\text{RESET}}$, and must be kept stable while $\overline{\text{RESET}}$ is low. External (weak) pull-ups and/or pull-downs may be used for this purpose.

2.3 Security

When the Flash is secured (Please refer to Section 18.7.9, “Flash Security” for details), the MCU is considered in a “Secured” state. The purpose of this state is to prevent the application code or data stored in the Flash Main Array and Shadow Block from being read by outside sources.

The device will make available a security feature preventing the unauthorized read and write of the memory contents. This feature allows:

- Protection of the contents of Flash main array,
- Protection of the contents of Shadow Block,
- Operation in Single Chip mode,

- Operation from external memory with internal Flash main array and Shadow Block disabled,
- Disabling of EICE and Nexus access in secured Single Chip and Primary Bootloader modes,
- JTAG Test Registers (See Section 35.1.1, “JTAG Test Register (SC4)) are accessible on a secured device.

Which features are enabled or disabled is determined by the chip mode and security state in effect. Please refer to Section 2.4.1, “Normal Single Chip Mode to Section 2.4.6, “Secured Expanded Mode for complete details.

Note that the security state of the Flash is applied to both the Main Array and to the Shadow Block. It is not possible to unsecure just one or the other.

The user must be reminded that part of the security must lie with the user’s code. An extreme example would be user’s code that dumps the contents of the internal program. This code would defeat the purpose of security. At the same time the user may also wish to put a back door in the user’s program. An example of this is the case in which a user downloads a key through the FlexCAN, allowing access to a programming routine that updates parameters stored in the Flash.

2.3.1 Operation of the Secured Microcontroller

2.3.1.1 Single Chip Secured Mode

In this mode, the MCU will function similarly to a device in Normal (Unsecured) Single Chip Mode. However the FlexBus interface will be disabled, and access to the device via the Debug port will be blocked, thus preventing access to the contents of the memory.

2.3.1.2 Executing from External Memory

The user may wish to execute from external space with a secured microcontroller. This is accomplished by resetting directly into expanded mode. With the MCU in secure mode, access to the Flash main array and Shadow Block will be disabled.

2.3.2 Securing the Microcontroller

Once the user has written the contents of the Flash main array and Shadow Block (if desired), the part can be secured by programming the security bits located in the Shadow Block. These non-volatile bits will keep the part secured through resetting the part and through powering down the part. When security is enabled both the Flash main array and Shadow Block memories are secured (i.e.-it is not possible to configure security independently for these two memories).

For further information on the security of the Flash, consult Section 18.7.9, “Flash Security”.

2.3.3 Unsecuring the Microcontroller

There are two methods to unsecure the MCU:

- Software unsecure

- JTAG Lockout Recovery

2.3.3.1 Software Unsecure

Since the security state of the device is determined solely by a user programmable location in the Shadow Block, any application may choose to implement a software unsecure method, through any interface. There are no built-in features, such as a backdoor access key, on the MAC72xx.

2.3.3.2 JTAG Lockout Recovery

If a software unsecure of the device is not possible, an alternative method exists for unsecuring the device. In this method, called Lockout Recovery, the Flash main array and Shadow Block are first erased before the device is unsecured. This procedure can be performed with a JTAG instruction over the JTAG interface of the device. The recommended procedure is as follows:

1. Assert the RESET to the device
2. While the RESET is asserted, set the “Start Flash Lockout Recovery” bit (bit 9) in JTAG Test Register SC4
3. Release the RESET
4. The JTAG Lockout Recovery procedure will run automatically, erasing both the Flash main array and the Shadow Block, followed by a re-programming of the security bits to an unsecured state.
5. Sample the JTAG Test Register SC4 until the “Flash Lockout Recovery is complete” bit (bit 8) is set
6. If the “Flash Security Request” bit (bit 19) is cleared, the device is now unsecured
7. Reset the device again in order to leave JTAG Lockout Recovery mode. There is no need to clear the “Start Flash Lockout Recovery” or “Flash Lockout Recovery is complete” bits, as they are self-clearing.

Note that performing Lockout Recovery erases both the Flash main array and the Shadow Block.

2.4 MCU Mode Selection

The chip operating mode out of Reset is determined by the states of the MODA and MODB pins (Port D[1:0]) at Reset and the security status of the Program Flash.

Table 2-1. MCU Mode Selection Signals

Signal	Function
MODA/MODB (PD1/PD0)	These two pins are latched at reset to determine the chip mode (See Table 2-2).

Table 2-2. MCU Mode Selection

MODA (PD1)	MODB (PD0)	Flash Secured ?	Chip Mode
0	0	No	Normal Single Chip Mode
0	1	No	Normal Expanded Mode
1	0	No	Normal Primary Bootloader Mode
1	1	No	Reserved for future use
0	0	Yes	Secured Single Chip Mode
0	1	Yes	Secured Expanded Mode
1	0	Yes	Secured Primary Bootloader Mode
1	1	Yes	Reserved for future use

Once the MCU Mode has been set, it may only be changed by any of the following:

- Securing the flash, and resetting the device. In this case the mode changes from “Normal XX Mode” to “Secured XXX Mode”.
- Unsecuring the flash, and resetting the device. In this case the mode changes from “Secured XXX Mode” to “Normal XXX Mode”.
- Changing the MODA/MODB pins and resetting the device. In this case the mode changes from “Normal XXX Mode” to “Normal YYY Mode” or from “Secured XXX Mode” to “Secured YYY Mode”.

Note that resetting the device is mandatory in all three cases.

2.4.1 Normal Single Chip Mode

In Normal Single Chip Mode, the system boots from the Flash main array. The system may be configured to enable the external bus.

- EICE/Nexus/JTAG active
  - Enable TCK, TDI, TDO and DBGEN
- FlexBus available at $2000 0000
- Boot from Flash main array at $0000 0000
  - Shadow Block available at $00f0 0000
- JTAG Lockout Recovery is available

2.4.2 Secured Single Chip Mode

In Secured Single Chip Mode, the system boots from the Flash main array. The external bus interface is unavailable.

- No EICE/Nexus/JTAG
  - Disable DBGEN and Nexus clock(s)

- No FlexBus
- Boot from Flash main array at $0000 0000
  - Shadow Block available at $00f0 0000
- JTAG Lockout Recovery is available

2.4.3 Normal Primary Bootloader Mode

In Normal PBL Mode, the memory map of the system is modified to move the location of the Shadow Block from its default Reset location to starting at location $0000 0000. This remapping enables the device to boot from the Shadow Block.

- Same as Single Chip mode, except boot from the Shadow Block
- EICE/Nexus/JTAG active
  - Enable TCK, TDI, TDO and DBGEN
- No FlexBus
- Relocate Shadow Block to $0000 0000
- Relocate Flash main array to $2000 0000
  - Shadow Block available at $20f0 0000
- JTAG Lockout Recovery is available

2.4.4 Secured Primary Bootloader Mode

In Secured PBL Mode, the memory map of the system is modified to move the location of the Shadow Block from its default Reset location to starting at location $0000 0000. This remapping enables the device to boot from the Shadow Block.

- Same as Single Chip mode, except boot from the Shadow Block
- No EICE/Nexus/JTAG
  - Disable DBGEN and Nexus clock(s)
- No FlexBus
- Relocate Shadow Block to $0000 0000
- Relocate Flash main array to $2000 0000
  - Shadow Block available at $20f0 0000
- JTAG Lockout Recovery is available

2.4.5 Normal Expanded Mode

In Normal Expanded Mode, the MCU boots from an external source by accessing external memory and hardware across the external Address and Data bus. In this mode the Flash main array and Shadow Block are accessable.

- EICE/Nexus/JTAG active
  - Enable TCK, TDI, TDO and DBGEN

- Relocate Flash to $2000 0000
  - Shadow Block available at $20f0 0000
- Relocate FlexBus to $0000 0000
- Boot from FlexBus

2.4.6 Secured Expanded Mode

In Secured Expanded Mode, the MCU boots from an external source by accessing external memory and hardware across the external Address and Data bus. In this mode the Flash main array and Shadow Block are unavailable. Neither the Flash main array or the Shadow Block can be accessed, except for the accesses necessary to unsecure the device.

- EICE/Nexus/JTAG active
  - Enable TCK, TDI, TDO and DBGEN
- Disable Flash Main Array and Shadow Block access
- Relocate FlexBus to $0000 0000
- Boot from FlexBus

See Chapter 9, “Device Memory Map” for more detailed information on how each chip mode affects relocation of system resources.

2.5 Oscillator Type Selection

At Reset the type of Oscillator can be selected using the $\overline{\text{XCLKS}}$ pin.

Table 2-3. Oscillator Type Selection

Pin	Reset State Definition
$\overline{\text{XCLKS}}$ (PD2)	This bit determines the mode of the oscillator as follows: 0 External Clock Mode 1 ALC Mode Please refer to Section 24.8.1, “OSC Mode Selection” for more details on the various oscillator modes.

The oscillator mode ($\overline{\text{XCLKS}}$) is latched only during a power-on reset and during a reset while a loss of clock has occurred. It is not latched during a normal system reset. Please refer to Chapter 24, “Oscillator (OSC)” for more information on configuring and using the oscillator.

2.6 Nexus Port Selection

If use of the Nexus Port is required for debugging, it may be connected to either the Nexus Primary Port or the Nexus Secondary Port. The hardware configuration pins associated with the Nexus are:

Table 2-4. Nexus Port Selection

Pin	Reset State Definition
NEXPORTSEL (PF0)	This bit determines whether the Nexus, if present, resides on the Primary or Secondary Port.
NEXPRESENT (PF1)	This bit determines whether an external debug system is physically connected to the MAC72xx. Internally, it is used with **NEXUSPORTSEL** to place the Nexus Primary or Secondary Port pads into the correct configuration.

The Nexus may be enabled or disabled using the settings described in Table 2-5.

Table 2-5. Nexus Hardware Configuration

NEXPORTSEL	NEXPRESENT	Effect
0	0	Nexus is not present.
0	1	Nexus is present on the Primary Port.
1	0	Nexus is not present.
1	1	Nexus is present on the Secondary Port.

The Nexus Port may be changed only by resetting the device. Please refer to Chapter 8, “Debug” for more information on configuring and using Nexus.

2.7 External Bus Configuration

If the device and MCU Mode support an external bus, then it may be configured via the PF2 and PF3 pins.

Table 2-6. External Bus Configuration

Pin	Reset State Definition
AUTOACK (PF2)	This bit enables the Auto Acknowledge feature for the Global Chip Select on the External Bus. Internally, this latched value is tied to **fb_rst_cfg[2]**.
PORTSIZE (PF3)	This bit determines the port size of the External Bus as follows: 0 8-bit 1 16-bit Internally this latched value is driven onto **fb_rst_cfg[1]**, while **fb_rst_cfg[0]** is tied to 1. This precludes the use of a 32-bit Port size, which is not supported on the MAC72xx. The full encoding for **fb_rst_cfg[1:0]** is: 00 32-bit 01 8-bit 1x 16-bit

Please refer to Chapter 17, “External Bus Interface (FlexBus)” for more information on configuring and using the External Bus.

2.8 Low Power Modes

The microcontroller features only a single low power mode:

- DOZE Mode. In this mode, individual peripherals may be shut down to allow a completely customizable power consumption profile tailored to the requirements of the application.

In addition to the above mode, each peripheral may be individually disabled in order to minimize power consumption for those peripherals not used in a particular application.

Please consult the relevant chapter for information on the module behavior in Doze Mode. An important source of information about the clocks in the system is Chapter 23, “Clock and Reset Generator (CRG)”.

See Chapter 3, “Low Power Modes” for more detailed information on low power modes.

2.8.1 Doze

In Doze mode the device can be configured to selectively halt the operation of individual peripherals. In this mode it is possible to maintain operation of the DMA. For further power consumption savings, the Real Time Interrupt and Watchdog can be disabled. This mode is entered by writing to the DOZE register in the CRG. Wake up from this mode can be via a reset, an RTI (if enabled), an SWT event (if enabled), a Self clock mode interrupt, a peripheral interrupt, an external interrupt, or the core clearing the DOZE register.

2.8.2 Run

Although this is not a low power mode, it is possible to enable power savings when operating in Run mode by disabling specific peripherals which are not required in the application. This is done with the MDIS register bit in the particular peripheral.

2.9 Debug Mode

In addition to the functional chip modes and low power modes described above, the MAC72xx supports a Debug mode. By placing the system in Debug mode, the following actions occur:

- The ARM7 core is placed into Halt Mode or Monitor Mode, depending on the state of the DCR[4] bit.
- The Real Time Interrupt (RTI) clock is shut off
- The Software Watchdog Timer (SWT) clock is shut off
- All peripheral level debug features are enabled
  - DMA2
  - FlexCAN2
  - DSPI
  - eMIOS
  - ATD
  - MCM (Software Watchdog Timer)

More information on system debug features can be found in Chapter 8, “Debug”.

Chapter 3
Low Power Modes

3.1 Low Power Modes Introduction

The MAC72xx provides three low power modes to enable the minimum run current for every application. The following low power modes are supported (in order of decreasing power consumption).

Table 3-1. Low Power Modes

Mode	Description
Run Mode	This is the “normal” operating mode of the MAC72xx, in which all clocks are running, and all peripherals are powered up.
Doze Mode	In this system level mode, the clocks to each peripheral may be disabled in order to reduce power consumption. This mode is intended for dynamic power management.
Disabled Mode	In this mode, the peripheral is completely disabled, typically because it is not used for the particular application being executed. This mode is intended for static power management.

3.2 Run Mode

This is the “normal” operating mode of the MAC72xx, in which all clocks are running, and all peripherals are powered up.

Entering Run mode

- Reset
- Exiting all other current Low Power Modes

In this mode, the following components may be individually controlled:

- Turn off the RTI by writing 0 to the RTI counter register
- Turn off the SWT by writing 0 to the SWT counter register
- Put any peripheral into Disable mode by setting its MDIS bit. Note that after a reset, all peripherals that support a Disable mode are put into this mode by default.

Exiting Run mode

- Run mode is exited by entering any other Low Power Mode

3.3 Doze Mode

In this system level mode, the clocks to each peripheral may be disabled in order to reduce power consumption. This mode is intended for dynamic power management. In Doze mode, each individual peripheral’s clock is gated by the peripheral itself (See Chapter 5, “System Clock Description” for more

details on this). This is typically done by setting or clearing a DOZE bit in one of the peripheral's control registers. Only the core clocks are actually gated in DOZE mode. IPI register clocks are gated only when no register accesses to the module are detected, thus still allowing access to the module's DOZE bit. All protocol clocks are running in DOZE mode in order to enable detection of wakeup events on external interfaces.

Entering Doze mode

- Core sets the SYSTEM_DOZE bit in the CRG

In this mode, the following components may be individually controlled:

- Turn off the RTI clock by setting RTIDOZE
- Turn off the SWT clock by setting SWTDOZE
- Turn off a peripheral by setting its DOZE bit

Exiting Doze mode

Any of the following events can be used to wake the system from DOZE mode:

- External Reset
- Crystal Monitor Reset
- SWT Reset
- Self Clock Mode Interrupt
- Real Time Interrupt (RTI)
- Any Wakeup event in the system
- Core writes $00 to the CRG DOZE register
- Core clears the DOZE bit in the individual peripheral (optional feature)

3.4 Disabled Mode

In this mode, the peripheral is completely disabled, typically because it is not used for the particular application being executed. This mode is intended for static power management.

- Controlled via the MDIS bit in each peripheral
- Software is responsible for setting the appropriate MODE bits in the Port Integration Module (PIM), in order to drive the correct control signals to the pad(s).
- Core enables peripheral by clearing its MDIS bit.
- Note that the peripheral does not finish any current transfers before shutting down.

Entering Disabled mode

- The reset state of all peripherals is disabled (MDIS=1)
- Software configures the appropriate pin(s) in the PIM, including setting the correct MODE bits.
- Core sets the MDIS bit in the peripheral

Exiting Disabled mode

- The reset state of all peripherals is disabled (MDIS=1)

- Core clears the MDIS bit in the peripheral

3.5 System Wakeup

Because the interrupt controller has a combinatorial bypass signal, that is active even during low power modes, all interrupt sources in the system will serve as wakeup signals, with a flexible priority masking scheme (i.e.-only interrupts above a certain priority will wake the system).

This gives us the following sources to wake up the system:

Table 3-2. Wakeup Sources

	Where to enable	Pending Status in INTC ?	Where to clear
Interrupt	Peripheral INTC MCM (MWCR reg)	Yes	Peripheral INTC

The general flow to setup a wakeup is as follows:

1. Set the ENBWCR bit in the MWCR register in the MCM
2. Set the PRILVL[3:0] bits in the MWCR register in the MCM
3. Clear any pending interrupts in the peripheral(s) that will be used to generate wakeup interrupts
4. Set the appropriate level in the ICR register in the interrupt controller corresponding to the interrupt source(s) that will be used for wakeup
5. Write the IMRL and/or IMRH registers in the interrupt controller to enable all interrupts that will be used for wakeup
6. Setup the interrupt(s) in the peripheral(s) that will be used to generate wakeup interrupts. Generally, this involves setting an interrupt enable and clearing an interrupt mask

Once the system has woken up, the general flow to identify the source of the wakeup is as follows:

1. Read the IPRH and IPRL registers in the interrupt controller to determine which (if any) interrupt wakeup sources are pending
2. Once the wakeup sources have been identified, clear all sources

3.6 Low Power Mode Differences from MAC71xx

- STOP mode is not supported
- Wakeup from DOZE is only on interrupts now
  - — Wakeup from VREG API is removed
  - — Wakeup from PIM Keypad Wake-Up (KWU) is removed. Wakeup from an external source can still be done using the normal interrupt functionality of the PIM.

3.7 Low Power Mode Summary

Table 3-3. Low Power Mode Entry Summary

Event	RUN	DOZE	DEBUG
External Reset	X		
Write DOZE register		X	
Set DOZE bit (on a Peripheral-by-Peripheral basis)		X	
Core enters HALT state			X

NOTE

In most cases, entering RUN mode is equivalent to exiting a Low Power Mode, so this is described in Table 3-4.

Table 3-4. Low Power Mode Exit Summary

Event	RUN	DOZE	DEBUG
Enter DOZE Mode	X	--	X
Enter DEBUG Mode	X		--
Core enters HALT state	X		
Core exits HALT state			X
External Reset		X	X
Crystal Monitor Reset		X	
SWT Reset		X	
SWT Interrupt		X	
SCM Interrupt		X	
Real Time Interrupt (RTI)		X	
Any Enabled, Unmasked Interrupt in the Interrupt Controller		X	
$00 written to DOZE reg		X	
DOZE bit cleared (on a Peripheral-by-Peripheral basis)		X	

3.8 Special Notes on Entering and Exiting Power Modes

When entering DOZE mode on the FlexCAN, I^2C or SCI, care should be taken to insure that the corresponding pin-side bus is stopped in a *recessive* state.

Because the ARM7 core has no DOZE (or similar) instruction, this functionality will be enabled via a combination of software and hardware (Graceful Stop module). However, there are some “caveats” that must be paid attention to:

- Disabling of interrupts between checking of task list (in a multi-tasking environment), and the execution of the DOZE sequence. This is the responsibility of the software.
- Disabling of interrupts during the DOZE sequence. This is the responsibility of the software.

Chapter 4
Signal Description

This section describes signals that connect off-chip. It includes pinout diagrams, a table of signal properties, and detailed discussion of the signals.

4.1 Device Pinout

The MAC7200 family is available in 144-pin Quad Flat Package (LQFP), and 100-pin QFP options. The family of devices offer pin-compatible packaged devices to assist with system development and accommodate expansion of the application.

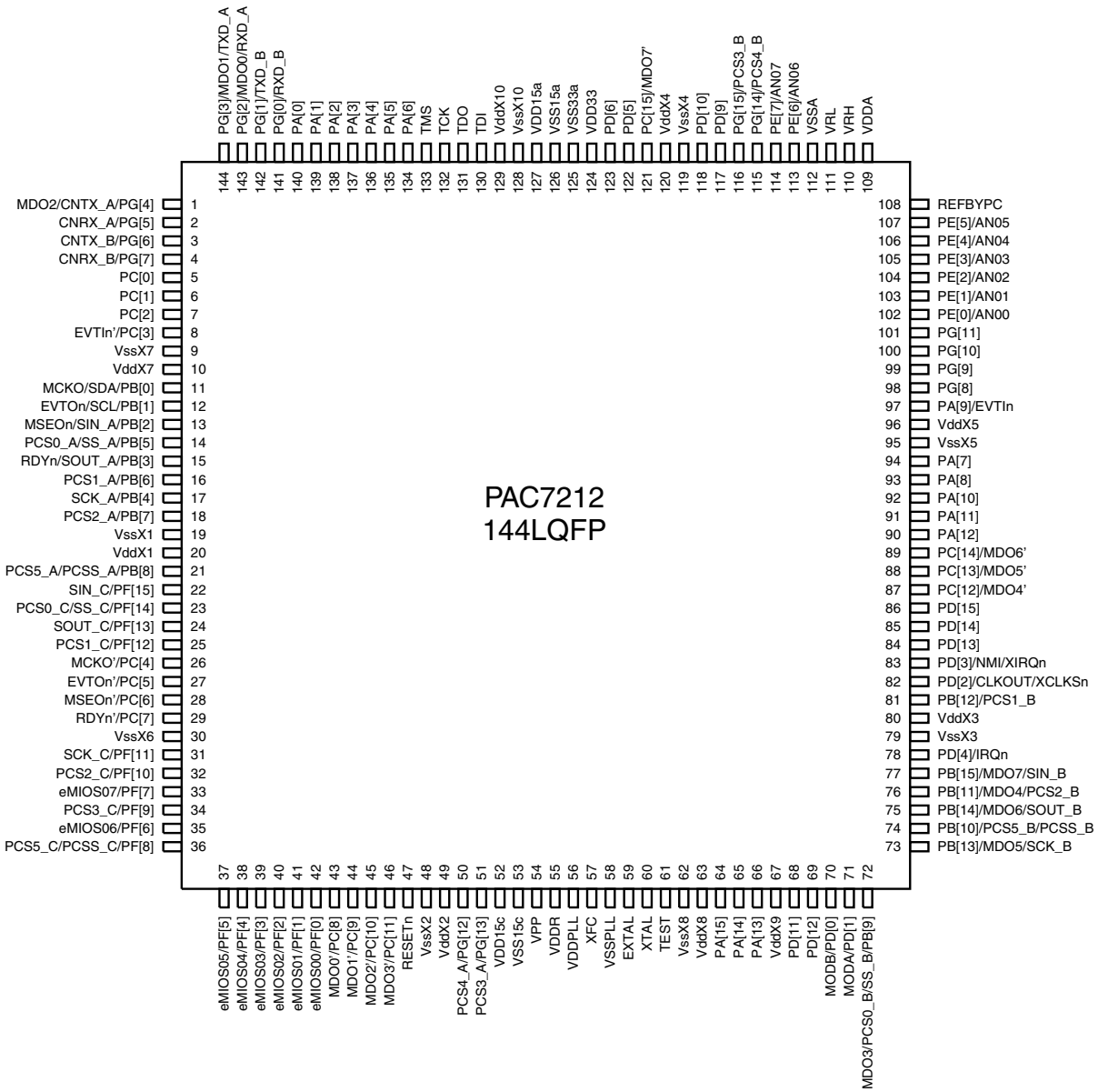
The MAC72xx device is offered in the following options:

- 144-pin LQFP package with 8 ATD channels.
- 144-pin LQFP package with 16 ATD channels.
- 100-pin LQFP package with 15 ATD channels.

Most pins perform two or more functions, which is described in more detail in Section 4.2, “Signal Properties Summary”. Figure 4-1, Figure 4-2, Figure 4-3 show the package pin assignments for the various packages.

NOTE

The 144LQFP packages are not currently qualified; any future qualification will be based upon customer demand and will be subject to specified leadtimes. This package option is supported for limited samples only, and does not include burn-in testing.

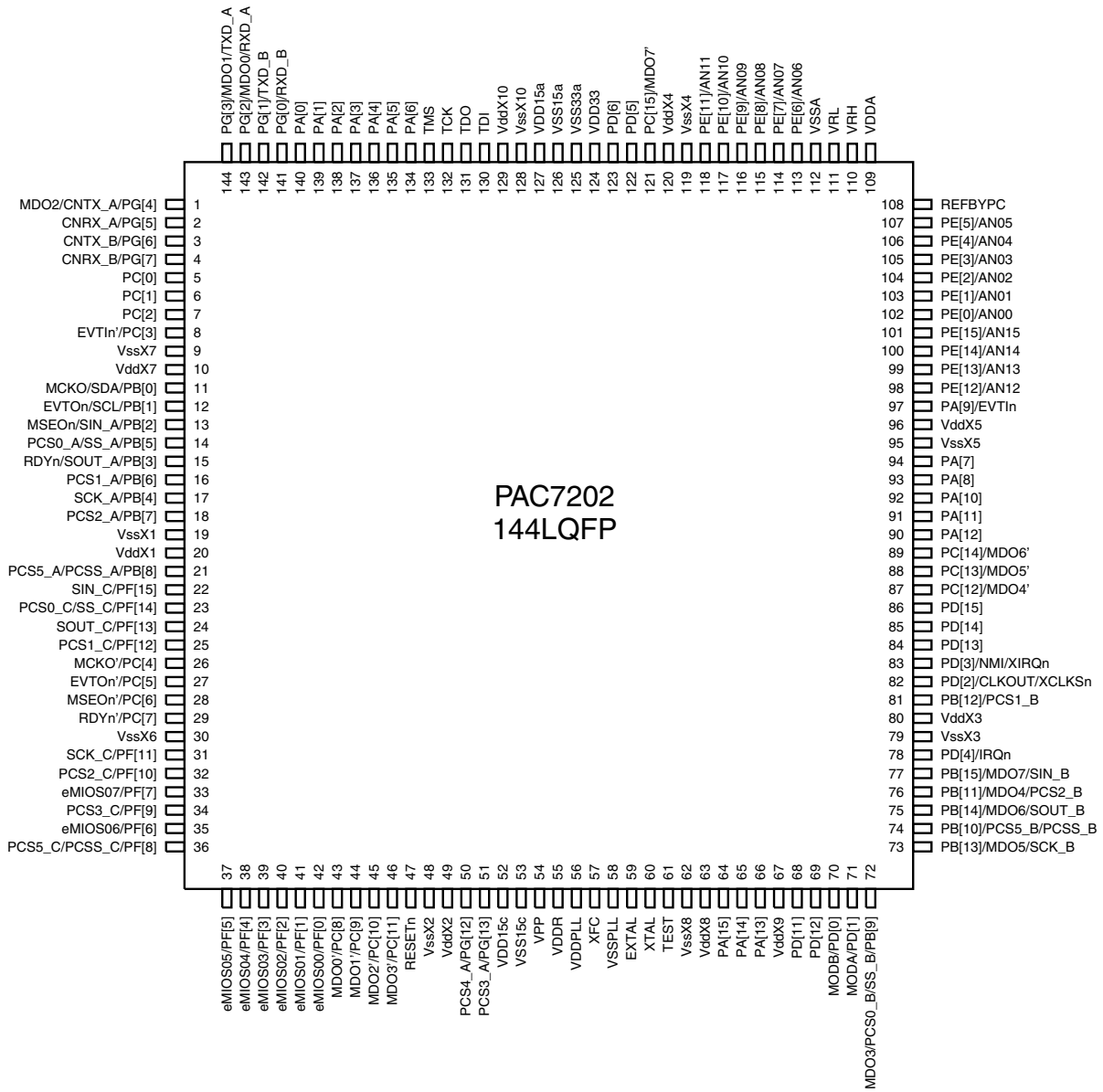


Secondary Mux function provides DSPI_B Chip Select 3 on pin PA[9]
Secondary Mux function provides DSPI_B Chip Select 4 on pin PA[8]

Figure 4-1. PAC7211/PAC7212 Pin Assignment, 144 QFP

NOTE

The 144LQFP packages are not currently qualified; any future qualification will be based upon customer demand and will be subject to specified leadtimes. This package option is supported for limited samples only, and does not include burn-in testing.

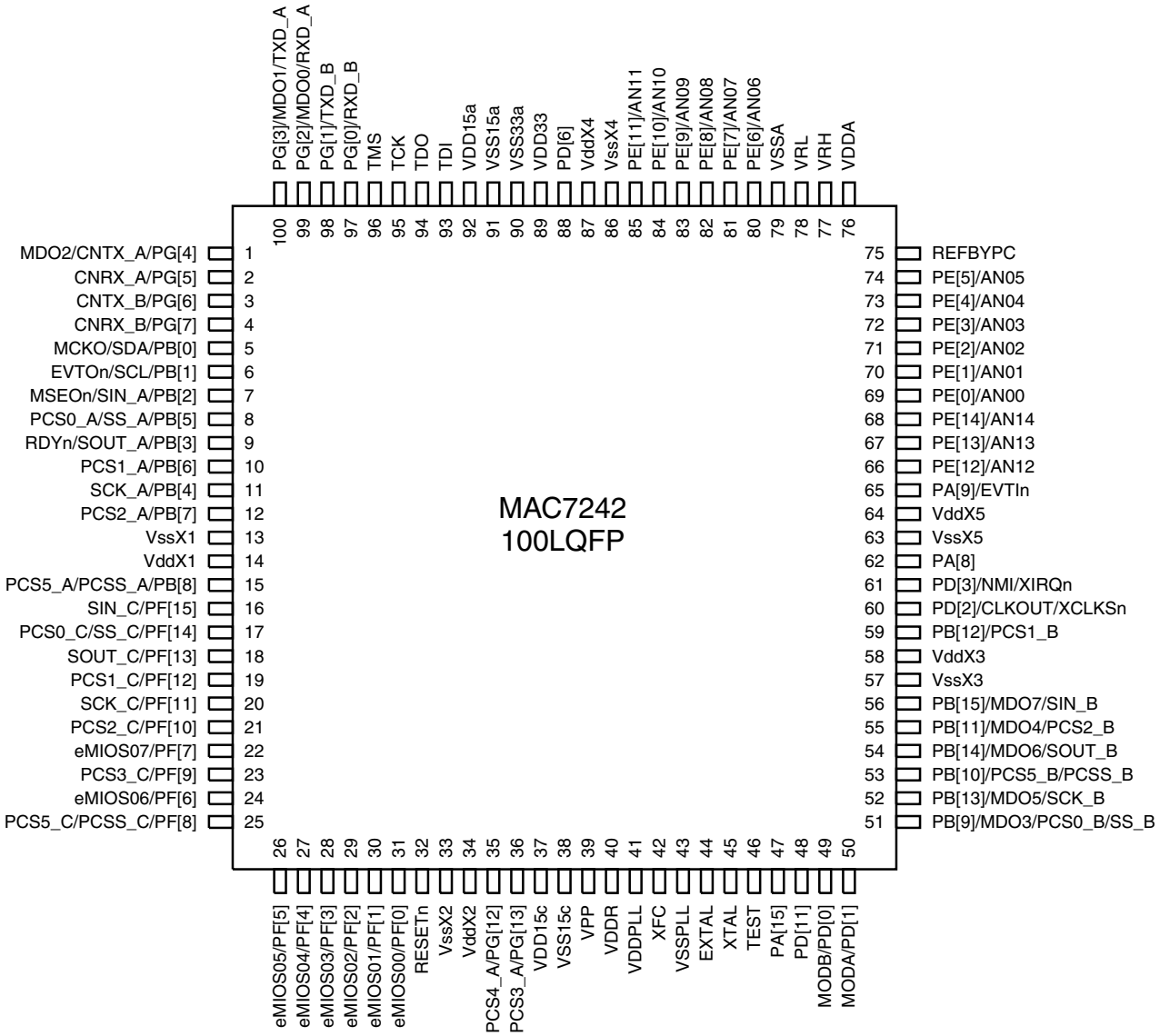


Secondary Mux function provides DSPI_B Chip Select 3 on pin PA[9]
Secondary Mux function provides DSPI_B Chip Select 4 on pin PA[8]

Figure 4-2. PAC7201/PAC7202 Pin Assignment, 144 QFP

NOTE

The 144LQFP packages are not currently qualified; any future qualification will be based upon customer demand and will be subject to specified leadtimes. This package option is supported for limited samples only, and does not include burn-in testing.



Secondary Mux function provides DSPI_B Chip Select 3 on pin PA[9]
Secondary Mux function provides DSPI_B Chip Select 4 on pin PA[8]

Figure 4-3. MAC7241/MAC7242 Pin assignments, 100 Pin LQFP package

4.2 Signal Properties Summary

Table 4-1 summarizes the pin functionality. on the MAC72xx devices.

Table 4-1. Signal Properties

Pin Name Func. 1	Pin Name Func. 2	Pin Name Func. 3	Pin Name Func. 4	Pin Name Func. 5	Power Supply	Internal Pull Resistor		Description
						Ctrl	Reset State	
EXTAL	—	—	—	—	VDDPLL	NA	NA	Oscillator pins
XTAL	—	—	—	—	VDDPLL	NA	NA	
XFC	—	—	—	—	VDDPLL	NA	NA	PLL Loop Filter
RESET	—	—	—	—	VDDx	Pull-down	Disabled	External Reset
TEST[1]	—	—	—	—	NA	—	—	Test Input
TCK	—	—	—	—	VDDx	PIM	Pull-down	JTAG TCK
TMS	—	—	—	—	VDDx	PIM	Pull-up	JTAG TMS
TDI	—	—	—	—	VDDx	PIM	Pull-up	JTAG TDI
TDO	—	—	—	—	VDDx	—	—	JTAG TDO
PA0[2]	DATA[0]	SCITXD_A	None	—	VDDx	PIM	Disabled	Port A0, External Data bus, eSCI_A serial transmit data
PA1[2]	DATA[1]	SCIRXD_A	None	—	VDDx	PIM	Disabled	Port A1, External Data bus, eSCI_A serial receive data
PA2[2]	DATA[2]	None	None	—	VDDx	PIM	Disabled	Port A2, External Data bus
PA3[2]	DATA[3]	None	None	—	VDDx	PIM	Disabled	Port A3, External Data bus
PA4[2]	DATA[4]	None	None	—	VDDx	PIM	Disabled	Port A4, External Data bus
PA5[2]	DATA[5]	None	None	—	VDDx	PIM	Disabled	Port A5, External Data bus
PA6[2]	DATA[6]	None	None	—	VDDx	PIM	Disabled	Port A6, External Data bus
PA7[2]	DATA[7]	None	None	—	VDDx	PIM	Disabled	Port A7, External Data bus
PA8[2]	None	PCS4_B	None	—	VDDx	PIM	Disabled	Port A8, DSPI_B chip select
PA9[2]	None	PCS3_B	None	EVTI[3]	VDDx	PIM	Disabled	Port A9, DSPI_B chip select, Nexus Event In
PA10[2]	None	PCS3_B	None	—	VDDx	PIM	Disabled	Port A10, DSPI_B chip select
PA11[2]	None	None	None	—	VDDx	PIM	Disabled	Port A11
PA12[2]	None	None	None	—	VDDx	PIM	Disabled	Port A12
PA13[2]	None	None	None	—	VDDx	PIM	Disabled	Port A13
PA14[2]	None	None	None	—	VDDx	PIM	Disabled	Port A14
PA15[2]	None	None	None	—	VDDx	PIM	Disabled	Port A15

Table 4-1. Signal Properties

Pin Name Func. 1	Pin Name Func. 2	Pin Name Func. 3	Pin Name Func. 4	Pin Name Func. 5	Power Supply	Internal Pull Resistor		Description
						Ctrl	Reset State	
PB0	SDA	PCS0_A SS_A	None	MCKO3	VDDx	PIM	Disabled	Port B0, IIC serial data, DSPI_A chip select/slave select, Nexus Clock Out
PB1	SCL	PCS1_A	None	EVTO3	VDDx	PIM	Disabled	Port B1, IIC serial data, DSPI_A chip select, Nexus Event Out
PB2	MISO_A	None	None	MSEO3	VDDx	PIM	Disabled	Port B2, DSPI_A serial data, Nexus Message Start/End Out
PB3	MOSI_A	SCL	None	RDY3	VDDx	PIM	Disabled	Port B3, DSPI_A serial data, IIC serial data
PB4	SCK_A	MOSI_A	None	—	VDDx	PIM	Disabled	Port B4, DSPI_A serial clock, DSPI_A serial data
PB5	PCS0_A SS_A	SDA	None	—	VDDx	PIM	Disabled	Port B5, DSPI_A chip select/slave select, IIC serial data
PB6	PCS1_A	MISO_A	None	—	VDDx	PIM	Disabled	Port B6, DSPI_A chip select, DSPI_A serial data
PB7	PCS2_A	SCK_A	None	—	VDDx	PIM	Disabled	Port B7, DSPI_A chip select, DSPI_A serial clock
PB8	PCS5_A	PCS0_A SS_A	PCS2_A	—	VDDx	PIM	Disabled	Port B8, DSPI_A chip select, DSPI_A chip select/slave select, DSPI_A chip select
PB9	PCS0_B SS_B	None	CNTXD_A	MDO33	VDDx	PIM	Disabled	Port B9, DSPI_B chip select/slave select, CAN_A serial transmit data, Nexus Message Data Out
PB10	PCS5_B PCSS_B	PCS2_B	CNTXD_B	—	VDDx	PIM	Disabled	Port B10, DSPI_B chip select, DSPI_B chip select, CAN_B serial transmit data
PB11	PCS2_B	SCK_B	SDA	MDO43	VDDx	PIM	Disabled	Port B11, DSPI_B chip select, DSPI_B serial clock, IIC serial data, Nexus Message Data Out
PB12	PCS1_B	MISO_B	None	—	VDDx	PIM	Disabled	Port B12, DSPI_B chip select, DSPI_B serial data
PB13	SCK_B	PCS5_B PCSS_B	CNRXD_A	MDO53	VDDx	PIM	Disabled	Port B13, DSPI_B serial clock, DSPI_B chip select, CAN_A serial receive data, Nexus Message Data Out
PB14	MOSI_B	PCS1_B	CNRXD_B	MDO63	VDDx	PIM	Disabled	Port B14, DSPI_B serial data, DSPI_B chip select, CAN_B serial receive data, Nexus Message Data Out
PB15	MISO_B	MOSI_B	SCL	MDO73	VDDx	PIM	Disabled	Port B15, DSPI_B serial data, DSPI_B serial data, IIC serial data, Nexus Message Data Out
PC0[2]	ADDR[0]	CNTXD_A	None	—	VDDx	PIM	Disabled	Port C0, External Address bus, CAN_A serial transmit data

Table 4-1. Signal Properties

Pin Name Func. 1	Pin Name Func. 2	Pin Name Func. 3	Pin Name Func. 4	Pin Name Func. 5	Power Supply	Internal Pull Resistor		Description
						Ctrl	Reset State	
PC1[2]	ADDR[1]	CNRXD_A	None	—	VDDx	PIM	Disabled	Port C1, External Address bus, CAN_A serial receive data
PC2[2]	ADDR[2]	CNTXD_B	None	—	VDDx	PIM	Disabled	Port C2, External Address bus, CAN_B serial transmit data
PC3[2]	ADDR[3]	CNRXD_B	None	EVTI[4]	VDDx	PIM	Disabled	Port C3, External Address bus, CAN_B serial receive data, Nexus Event In
PC4[2]	ADDR[4]	EMIOS[6]	None	MCKO4	VDDx	PIM	Disabled	Port C4, External Address bus, eMIOS Channel, Nexus Message Clock Out
PC5[2]	ADDR[5]	EMIOS[7]	None	EVTO4	VDDx	PIM	Disabled	Port C5, External Address bus, eMIOS Channel, Nexus Event Out
PC6[2]	ADDR[6]	None	None	MSEO4	VDDx	PIM	Disabled	Port C6, External Address bus, Nexus Message Start/End Out
PC7[2]	ADDR[7]	None	None	RDY4	VDDx	PIM	Disabled	Port C7, External Address bus, Nexus Ready
PC8[2]	ADDR[8]	EMIOS[0]	None	MDO04	VDDx	PIM	Disabled	Port C8, External Address bus, eMIOS Channel, Nexus Message Data Out
PC9[2]	ADDR[9]	None	None	MDO14	VDDx	PIM	Disabled	Port C9, External Address bus, Nexus Message Data Out
PC10[2]	ADDR[10]	None	None	MDO24	VDDx	PIM	Disabled	Port C10, External Address bus,
PC11[2]	ADDR[11]	None	None	MDO34	VDDx	PIM	Disabled	Port C11, External Address bus, Nexus Message Data Out
PC12[2]	ADDR[12]	None	None	MDO44	VDDx	PIM	Disabled	Port C12, External Address bus, Nexus Message Data Out
PC13[2]	ADDR[13]	None	None	MDO54	VDDx	PIM	Disabled	Port C13, External Address bus, Nexus Message Data Out
PC14[2]	ADDR[14]	None	None	MDO64	VDDx	PIM	Disabled	Port C14, External Address bus, Nexus Message Data Out
PC15[2]	ADDR[15]	None	None	MDO74	VDDx	PIM	Disabled	Port C15, External Address bus, Nexus Message Data Out
PD0[2]	BS0	PCS4_B	None	MODB	VDDx	PIM	Disabled	Port D0, External Bus Byte Select, DSPI_B chip select, Chip Mode select
PD1[2]	BS1	PCS3_B	None	MODA	VDDx	PIM	Disabled	Port D1, External Bus Byte Select, DSPI_B chip select, Chip Mode select
PD2[2]	CLKOUT	None	None	XCLKS[5]	VDDx	PIM	Disabled	Port D2, External Bus Clock, Oscillator mode select
PD3	XIRQ	None	None	NMI	VDDx	PIM	Disabled	Port D3, External interrupt, Non-Maskable Interrupt

Table 4-1. Signal Properties

Pin Name Func. 1	Pin Name Func. 2	Pin Name Func. 3	Pin Name Func. 4	Pin Name Func. 5	Power Supply	Internal Pull Resistor		Description
						Ctrl	Reset State	
PD4	IRQ	BURST	None	—	VDDx	PIM	Disabled	Port D4, External Interrupt, External Bus Burst
PD5[2]	ADDR[16]	None	None	—	VDDx	PIM	Disabled	Port D5, External Address bus
PD6[2]	ADDR[17]	None	None	—	VDDx	PIM	Disabled	Port D6, External Address bus
PD7[2]	ADDR[20]	None	None	—	VDDx	PIM	Disabled	Port D7, External Address bus
PD8[2]	ADDR[21]	TA	None	—	VDDx	PIM	Disabled	Port D8, External Address bus, External Bus Transfer Acknowledge
PD9[2]	ADDR[18]	None	None	—	VDDx	PIM	Disabled	Port D9, External Address bus
PD10[2]	ADDR[19]	TA	None	—	VDDx	PIM	Disabled	Port D10, External Address bus, External Bus Transfer Acknowledge
PD11[2]	OE	None	None	—	VDDx	PIM	Disabled	Port D11, External Bus Output Enable
PD12[2]	BURST	CS2	None	—	VDDx	PIM	Disabled	Port D12, External Bus Burst, External Bus Chip Select
PD13[2]	TA	CS1	None	—	VDDx	PIM	Disabled	Port D13, External Bus Transfer Acknowledge, External Bus Chip Select
PD14[2]	CS0	None	None	—	VDDx	PIM	Disabled	Port D14, External Bus Chip Select
PD15[2]	RW	None	None	—	VDDx	PIM	Disabled	Port D15, External Bus Read/Write
PE0[6]	AN_00	None	None	—	VDDA	PIM	Disabled	Port E0, ATD Channel
PE1[6]	AN_01	None	None	—	VDDA	PIM	Disabled	Port E1, ATD Channel
PE2[6]	AN_02	None	None	—	VDDA	PIM	Disabled	Port E2, ATD Channel
PE3[6]	AN_03	None	None	—	VDDA	PIM	Disabled	Port E3, ATD Channel
PE4[6]	AN_04	None	None	—	VDDA	PIM	Disabled	Port E4, ATD Channel
PE5[6]	AN_05	None	None	—	VDDA	PIM	Disabled	Port E5, ATD Channel
PE6[6]	AN_06	None	None	—	VDDA	PIM	Disabled	Port E6, ATD Channel
PE7[6]	AN_07	None	None	—	VDDA	PIM	Disabled	Port E7, ATD Channel
PE8[6]	AN_08	None	None	—	VDDA	PIM	Disabled	Port E8, ATD Channel
PE9[6]	AN_09	None	None	—	VDDA	PIM	Disabled	Port E9, ATD Channel
PE10[6]	AN_10	None	None	—	VDDA	PIM	Disabled	Port E10, ATD Channel
PE11[6]	AN_11	None	None	—	VDDA	PIM	Disabled	Port E11, ATD Channel
PE12[6]	AN_12	None	None	—	VDDA	PIM	Disabled	Port E12, ATD Channel
PE13[6]	AN_13	None	None	—	VDDA	PIM	Disabled	Port E13, ATD Channel
PE14[6]	AN_14	None	None	—	VDDA	PIM	Disabled	Port E14, ATD Channel

Table 4-1. Signal Properties

Pin Name Func. 1	Pin Name Func. 2	Pin Name Func. 3	Pin Name Func. 4	Pin Name Func. 5	Power Supply	Internal Pull Resistor		Description
						Ctrl	Reset State	
PE15[6]	AN_15	None	None	—	VDDA	PIM	Disabled	Port E15, ATD Channel
PF0	EMIOS[0]	None	None	—	VDDx	PIM		Port F0, eMIOS Channel, Debug Status Port, Nexus Port Select
PF1	EMIOS[1]	None	None	—	VDDx	PIM		Port F1, eMIOS Channel, Debug Status Port, Nexus Present
PF2	EMIOS[2]	None	None	—	VDDx	PIM		Port F2, eMIOS Channel, Debug Status Port, External Bus Auto Acknowledge
PF3	EMIOS[3]	None	None	—	VDDx	PIM		Port F3, eMIOS Channel, Debug Status Port, External Bus Port Size
PF4	EMIOS[4]	None	None	—	VDDx	PIM	Disabled	Port F4, eMIOS Channel, Debug Status Port
PF5	EMIOS[5]	None	PCS5_C PCSS_C	—	VDDx	PIM	Disabled	Port F5, eMIOS Channel, DSPI_C chip select, Debug Status Port
PF6	EMIOS[6]	None	PCS2_C	—	VDDx	PIM	Disabled	Port F6, eMIOS Channel, DSPI_C chip select, Debug Status Port
PF7	EMIOS[7]	None	PCS0_C SS_C	—	VDDx	PIM	Disabled	Port F7, eMIOS Channel, DSPI_C chip select/slave select, Debug Status Port
PF8	PCS5_C PCSS_C	EMIOS[7]	PCS3_C	—	VDDx	PIM	Disabled	Port F8, DSPI_C chip select, eMIOS Channel, DSPI_C chip select
PF9	PCS3_C	EMIOS[1]	PCS1_C	—	VDDx	PIM	Disabled	Port F9, DSPI_C chip select, eMIOS Channel, DSPI_C chip select
PF10	PCS2_C	EMIOS[3]	SCK_C	—	VDDx	PIM	Disabled	Port F10, DSPI_C chip select, eMIOS Channel, DSPI_C serial clock
PF11	SCK_C	EMIOS[4]	MOSI_C	—	VDDx	PIM	Disabled	Port F11, DSPI_C serial clock, eMIOS Channel, DSPI_C serial data
PF12	PCS1_C	EMIOS[5]	MISO_C	—	VDDx	PIM	Disabled	Port F12, DSPI_C chip select, eMIOS Channel, DSPI_C serial data
PF13	MOSI_C	PCS5_A PCSS_A	EMIOS[6]	—	VDDx	PIM	Disabled	Port F13, DSPI_C serial data, DSPI_A chip select, eMIOS Channel
PF14	PCS0_C SS_C	PCS2_A	EMIOS[7]	—	VDDx	PIM	Disabled	Port F14, DSPI_C chip select/slave select, DSPI_A chip select, eMIOS Channel
PF15	MISO_C	PCS1_A	PCS5_A PCSS_A	—	VDDx	PIM	Disabled	Port F15, DSPI_C serial data, DSPI_A chip select, DSPI_A chip select
PG0	SCIRXD_B	CNRXD_B	None	—	VDDx	PIM	Disabled	Port G0, eSCI_B serial receive data, CAN_B serial receive data
PG1	SCITXD_B	CNTXD_B	None	—	VDDx	PIM	Disabled	Port G1, eSCI_B serial transmit data, CAN_B serial transmit data

Table 4-1. Signal Properties

Pin Name Func. 1	Pin Name Func. 2	Pin Name Func. 3	Pin Name Func. 4	Pin Name Func. 5	Power Supply	Internal Pull Resistor		Description
						Ctrl	Reset State	
PG2	SCIRXD_A	CNRXD_A	None	MDO03	VDDx	PIM	Disabled	Port G2, eSCI_A serial receive data, CAN_A serial receive data, Nexus Message Data Out
PG3	SCITXD_A	CNTXD_A	None	MDO13	VDDx	PIM	Disabled	Port G3, eSCI_A serial transmit data, CAN_A serial transmit data, Nexus Message Data Out
PG4	CNTXD_A	SCITXD_A	None	MDO23	VDDx	PIM	Disabled	Port G4, CAN_A serial transmit data, eSCI_A serial transmit data, Nexus Message Data Out
PG5	CNRXD_A	SCIRXD_A	None	—	VDDx	PIM	Disabled	Port G5, CAN_A serial receive data, eSCI_A serial receive data
PG6	CNTXD_B	SCITXD_B	None	—	VDDx	PIM	Disabled	Port G6, CAN_B serial transmit data, eSCI_B serial transmit data
PG7	CNRXD_B	SCIRXD_B	None	—	VDDx	PIM	Disabled	Port G7, CAN_B serial transmit data, eSCI_B serial receive data
PG8	CS1	None	None	—	VDDx	PIM	Disabled	Port G8, External Bus Chip Select
PG9	CS2	None	None	—	VDDx	PIM	Disabled	Port G9, External Bus Chip Select
PG10	None	None	None	—	VDDx	PIM	Disabled	Port G10
PG11	None	None	None	—	VDDx	PIM	Disabled	Port G11
PG12	PCS4_A	SCIRXD_A	SCIRXD_B	—	VDDx	PIM	Disabled	Port G12, DSPI_A chip select, eSCI_A serial receive data, eSCI_B serial receive data
PG13	PCS3_A	SCITXD_A	SCITXD_B	—	VDDx	PIM	Disabled	Port G13, DSPI_A chip select, eSCI_A serial transmit data, eSCI_B serial transmit data
PG14	PCS4_B	None	None	—	VDDx	PIM	Disabled	Port G14, DSPI_B chip select
PG15	PCS3_B	None	None	—	VDDx	PIM	Disabled	Port G15, DSPI_B chip select

1. This pin must always be tied to VSS in user mode.
2. This pin has Function 2 automatically selected when the chip is booted into External Chip mode.
3. Nexus Primary Port
4. Nexus Secondary Port
5. This functionality is for Hardware Configuration only, and therefore active only during reset.
6. No General Purpose Output (GPO) functionality is available on this pin

4.3 Detailed Signal Descriptions

4.3.1 EXTAL, XTAL — Oscillator Pins

EXTAL and XTAL are the crystal driver and external clock pins. On reset all the device clocks are derived from the EXTAL input frequency. XTAL is the crystal output. When the $\overline{\text{XCLKS}}$ pin is driven low, an external 3.3V clock signal can be driven in on the EXTAL pin directly. Otherwise, the $\overline{\text{XCLKS}}$ pin must be driven high until after the $\overline{\text{RESET}}$ pin has been negated.

4.3.2 $\overline{\text{RESET}}$ — External Reset Pin

An open drain, active low bidirectional control signal, it acts as an input to initialize the MCU to a known start-up state, and an output when an internal MCU function causes a Reset. A pull-down resistor can be activated on this pin. This allows the device to enter reset when the connection to an external reset source is lost. This resistor is implemented using active elements, and is approximately equivalent to a 20KOhm resistor under typical operating conditions.

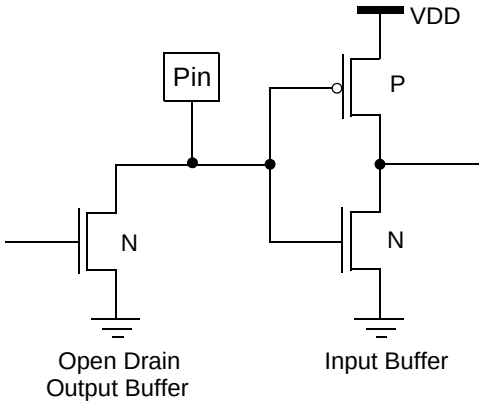


Figure 4-4. Bidirectional Open Drain Pin

4.3.3 XFC — PLL Loop Filter Pin

The XFC pin allows the user to specify the external PLL loop filter components to modify the PLLs response rate and stability. Please refer to Chapter 23, “Clock and Reset Generator (CRG)” for more details on this pin and for the calculation of the component values.

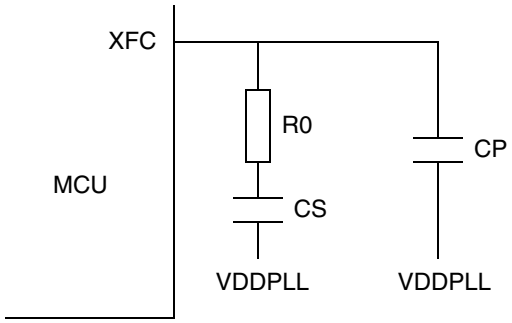


Figure 4-5. PLL Loop Filter Connections

4.3.4 TDI — Test Data In Pin

The TDI pin is a device input for the JTAG port. This serial pin is sampled in the device on the rising edge of the TCK pin. During and after a system reset, this pin will be configured as an input with an internal pull-up active on this signal. This pull-up is implemented using active elements, and is approximately equivalent to a 20KOhm resistor under typical operating conditions.

4.3.5 TDO — Test Data Output Pin

The TDO pin is a device output for the JTAG port. This pin is a three-state test data output pin that is actively driven in the shift-IR and shift-DR controller states. The pin state changes on the falling edge of the TCK pin. During and after a system reset, this pin will be configured as an output with maximum slew rate enabled.

4.3.6 TCK — Test Clock Pin

The TCK pin is a device input for the JTAG port. This pin provides the test clock input to synchronize the device's test logic. TCK is independent of the processor clock, and is used only as the reference clock for the TMS, TDI and TDO pins. During and after a system reset, this pin will be configured as an input with an internal pull-down active on this signal. This pull-down is implemented using active elements, and is approximately equivalent to a 20KOhm resistor under typical operating conditions.

4.3.7 TMS — Test Mode Pin

The TMS pin is a device input for the JTAG port. This pin is used to input the test mode to sequence the TAP controller's state machine. The pin is sampled on the rising edge of the TCK pin. During and after a system reset, this pin will be configured as an input with an internal pull-up active on this signal. This pull-up is implemented using active elements, and is approximately equivalent to a 20KOhm resistor under typical operating conditions.

4.3.8 PA[0:7] / DATA[0:7] — Port A I/O Pins and external Databus

PA[0] to PA[7] are configurable general purpose input or output pins. They can be independently configured to provide either high or reduced output drive, and also to enable or disable either a pull-up or pull-down resistor on the pin. The pin is multiplexed with the external Data bus lines DATA[0] to DATA[7]. Each pin can be independently configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.9 PA[8] / DATA[8] / PCS[4] — Port A I/O Pin, External Databus, and DSPI_B

PA[8] is a configurable general purpose input or output pin. It can be independently configured to provide either high or reduced output drive, and also to enable or disable either a pull-up or pull-down resistor on the pin. The pin is multiplexed in Primary Peripheral mode with the external Data bus line DATA[8]. In Secondary Peripheral mode, it can be configured as a chip select pin PCS[4] when in Master mode for the

Serial Peripheral Interface B (DSPI_B). It can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.10 PA[9] / DATA[9] / PCS[3] / NEX1EVTI — Port A I/O Pin, External Databus, DSPI_B and Nexus Primary

PA[9] is a configurable general purpose input or output pin. It can be independently configured to provide either high or reduced output drive, and also to enable or disable either a pull-up or pull-down resistor on the pin. The pin is multiplexed in Primary Peripheral mode with the external Data bus line DATA[9]. In Secondary Peripheral mode, it can be configured as a chip select pin PCS[3] when in Master mode for the Serial Peripheral Interface B (DSPI_B). It can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality. When the Nexus Primary Port is selected and active, this pin is used as the Event In input.

4.3.11 PA[10:15] / DATA[10:15] — Port A I/O Pins and external Databus

PA[10] to PA[15] are configurable general purpose input or output pins. They can be independently configured to provide either high or reduced output drive, and also to enable or disable either a pull-up or pull-down resistor on the pin. The pin is multiplexed with the external Data bus lines DATA[10] to DATA[15]. Each pin can be independently configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.12 PB[0] / SDA / NEX1MCKO — Port B I/O Pin, IIC and Nexus Primary

PB[0] is a configurable general purpose input or output pin. It can be configured to provide either high or reduced output drive, to enable or disable either a pull-up or pull-down resistor on the pin or to provide either an open drain or push-pull pad. The port pin is multiplexed with the bidirectional IIC serial data pin. It can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality. When the Nexus Primary Port is selected and active, this pin is used as the Nexus Clock output.

4.3.13 PB[1] / SCL / NEX1EVTO — Port B I/O Pin, IIC and Nexus Primary

PB[1] is a configurable general purpose input or output pin. It can be configured to provide either high or reduced output drive, to enable or disable either a pull-up or pull-down resistor on the pin or to provide either an open drain or push-pull pad. The port pin is multiplexed with the bidirectional IIC serial clock pin used to provide the time base for transfers. It can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality. When the Nexus Primary Port is selected and active, this pin is used as the Nexus Event Out output.

4.3.14 PB[2] / SIN_A / NEX1MSEO — Port B I/O Pin, DSPI_A and Nexus Primary

PB[2] is a configurable general purpose input or output pin. It can be configured to provide either high or reduced output drive, to enable or disable either a pull-up or pull-down resistor on the pin or to provide

either an open drain or push-pull pad. The pin can be configured as the input pin of the Serial Peripheral Interface A (DSPI_A). It can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality. When the Nexus Primary Port is selected and active, this pin is used as the Nexus Message Start/End output.

4.3.15 PB[3] / SOUT_A / NEX1RDY — Port B I/O Pin, DSPI_A and Nexus Primary

PB[3] is a configurable general purpose input or output pin. It can be configured to provide either high or reduced output drive, to enable or disable either a pull-up or pull-down resistor on the pin or to provide either an open drain or push-pull pad. The pin can be configured as the output pin of the Serial Peripheral Interface A (DSPI_A). It can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality. When the Nexus Primary Port is selected and active, this pin is used as the Nexus Ready output.

4.3.16 PB[4] / SCK_A — Port B I/O Pin and DSPI_A

PB[4] is a configurable general purpose input or output pin. It can be configured to provide either high or reduced output drive, to enable or disable either a pull-up or pull-down resistor on the pin or to provide either an open drain or push-pull pad. The pin can be configured as serial clock pin SCK of the Serial Peripheral Interface A (DSPI_A). It can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.17 PB[5] / PCS[0] / $\overline{\text{SS}}$[0] — Port B I/O Pin and DSPI_A

PB[5] is a configurable general purpose input or output pin. It can be configured to provide either high or reduced output drive, to enable or disable either a pull-up or pull-down resistor on the pin or to provide either an open drain or push-pull pad. The pin can be configured as chip select pin PCS[0] when in Master mode, or as slave select pin $\overline{\text{SS}}$ when in Slave mode for the Serial Peripheral Interface A (DSPI_A). It can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.18 PB[6:7] / PCS[1:2] — Port B I/O Pin and DSPI_A

PB[6] - PB[7] are configurable general purpose input or output pins. They can be configured to provide either high or reduced output drives, to enable or disable either a pull-up or pull-down resistor on the pins or to provide either open drain or push-pull pads. The pins can be configured as chip select pin PCS[1:2] when in Master mode for the Serial Peripheral Interface A (DSPI_A). Each pin can be independently configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.19 PB[8] / PCS[5] / $\overline{\text{PCSS}}$ — Port B I/O Pin and DSPI_A

PB[8] is a configurable general purpose input or output pin. It can be configured to provide either high or reduced output drive, to enable or disable either a pull-up or pull-down resistor on the pin or to provide

either an open drain or push-pull pad. The pin can be configured as chip select pin PCS[5] when in Master mode for the Serial Peripheral Interface A (DSPI_A), or as a Peripheral Chip Select Strobe to qualify the other Chip Select signals to eliminate any decoding glitches. Each pin can be independently configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.20 PB[9] / PCS0 / $\overline{SS}$[1] / NEX1MDO — Port B I/O Pin, DSPI_B and Nexus Primary

PB[9] is a configurable general purpose input or output pin. It can be configured to provide either high or reduced output drive, to enable or disable either a pull-up or pull-down resistor on the pin or to provide either an open drain or push-pull pad. The pin can be configured as chip select pin PCS[0] when in Master mode, or as slave select pin $\overline{SS}$ when in Slave mode for the Serial Peripheral Interface B (DSPI_B). After Reset the pin is able to be used as the General purpose input or output PB9, or as a DSPI_B control line. This pin can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality. When the Nexus Primary Port is selected and active, this pin is used as a Message Data Out output.

4.3.21 PB[10] / PCS[5] / $\overline{PCSS}$ — Port B I/O Pin and DSPI_B

PB[10] is a configurable general purpose input or output pin. It can be configured to provide either high or reduced output drives, to enable or disable either a pull-up or pull-down resistors on the pin or to provide either an open drain or push-pull pad. The pin can be configured as chip select pin PCS[5] when in Master mode for the Serial Peripheral Interface B (DSPI_B), or as a Peripheral Chip Select Strobe to qualify the other Chip Select signals to eliminate any decoding glitches. It can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.22 PB[11] / PCS[2] / NEX1MDO — Port B I/O Pin, DSPI_B and Nexus Primary

PB[11] is a configurable general purpose input or output pin. It can be configured to provide either high or reduced output drives, to enable or disable either a pull-up or pull-down resistors on the pin or to provide either an open drain or push-pull pad. The pin can be configured as a chip select pin PCS[2] when in Master mode for the Serial Peripheral Interface B (DSPI_B). It can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality. When the Nexus Primary Port is selected and active, this pin is used as a Message Data Out output.

4.3.23 PB[12] / PCS[1] — Port B I/O Pin and DSPI_B

PB[12] is a configurable general purpose input or output pin. It can be configured to provide either high or reduced output drives, to enable or disable either a pull-up or pull-down resistors on the pin or to provide either an open drain or push-pull pad. The pins can be configured as a chip select pin PCS[1] when in Master mode for the Serial Peripheral Interface B (DSPI_B). It can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.24 PB[13] / SCK_B / NEX1MDO — Port B I/O Pin, DSPI_B and Nexus Primary

PB[13] is a configurable general purpose input or output pin. It can be configured to provide either high or reduced output drive, to enable or disable either a pull-up or pull-down resistor on the pin or to provide either an open drain or push-pull pad. The pin can be configured as serial clock pin SCK of the Serial Peripheral Interface B (DSPI_B). It can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality. When the Nexus Primary Port is selected and active, this pin is used as a Message Data Out output.

4.3.25 PB[14] / SOUT_B / NEX1MDO — Port B I/O Pin, DSPI_B and Nexus Primary

PB[14] is a configurable general purpose input or output pin. It can be configured to provide either high or reduced output drive, to enable or disable either a pull-up or pull-down resistor on the pin or to provide either an open drain or push-pull pad. The pin can be configured as the output pin of the Serial Peripheral Interface B (DSPI_B). It can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality. When the Nexus Primary Port is selected and active, this pin is used as a Message Data Out output.

4.3.26 PB[15] / SIN_B / NEX1MDO — Port B I/O Pin, DSPI_B and Nexus Primary

PB[15] is a configurable general purpose input or output pin. It can be configured to provide either high or reduced output drive, to enable or disable either a pull-up or pull-down resistor on the pin or to provide either an open drain or push-pull pad. The pin can be configured as the input pin of the Serial Peripheral Interface B (DSPI_B). It can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality. When the Nexus Primary Port is selected and active, this pin is used as a Message Data Out output.

4.3.27 PC[0:2] / ADDR[0:2] — Port C I/O Pins and External address bus

PC[0] to PC[2] are configurable general purpose input or output pins. The pins can be independently configured to provide either high or reduced output drive, and also to enable or disable either a pull-up or pull-down resistor on the pin. The pin is multiplexed with the external address bus lines Address 0 to Address 2. Each pin can be independently configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.28 PC[3] / ADDR[3] / NEX2EVTI — Port C I/O Pins, External Address Bus and Nexus Secondary

PC[3] is a configurable general purpose input or output pin. It can be independently configured to provide either high or reduced output drive, and also to enable or disable either a pull-up or pull-down resistor on the pin. The pin is multiplexed with the external address bus line Address 3. Each pin can be independently configured to wake the system out of low-power mode (DOZE) when an external signal is applied using

the interrupt functionality. When the Nexus Secondary Port is selected and active, this pin is used as an Event In input.

4.3.29 PC[4] / ADDR[4] / NEX2MCKO — Port C I/O Pins, External Address Bus and Nexus Secondary

PC[4] is a configurable general purpose input or output pin. It can be independently configured to provide either high or reduced output drive, and also to enable or disable either a pull-up or pull-down resistor on the pin. The pin is multiplexed with the external address bus line Address 3. Each pin can be independently configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality. When the Nexus Secondary Port is selected and active, this pin is used as a Message Clock output.

4.3.30 PC[5] / ADDR[5] / NEX2EVTO — Port C I/O Pins, External Address Bus and Nexus Secondary

PC[5] is a configurable general purpose input or output pin. It can be independently configured to provide either high or reduced output drive, and also to enable or disable either a pull-up or pull-down resistor on the pin. The pin is multiplexed with the external address bus line Address 3. Each pin can be independently configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality. When the Nexus Secondary Port is selected and active, this pin is used as a Event Out output.

4.3.31 PC[6] / ADDR[6] / NEX2MSEO — Port C I/O Pins, External Address Bus and Nexus Secondary

PC[6] is a configurable general purpose input or output pin. It can be independently configured to provide either high or reduced output drive, and also to enable or disable either a pull-up or pull-down resistor on the pin. The pin is multiplexed with the external address bus line Address 3. Each pin can be independently configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality. When the Nexus Secondary Port is selected and active, this pin is used as a Message Start/End output.

4.3.32 PC[7] / ADDR[7] / NEX2RDY — Port C I/O Pins, External Address Bus and Nexus Secondary

PC[7] is a configurable general purpose input or output pin. It can be independently configured to provide either high or reduced output drive, and also to enable or disable either a pull-up or pull-down resistor on the pin. The pin is multiplexed with the external address bus line Address 3. Each pin can be independently configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality. When the Nexus Secondary Port is selected and active, this pin is used as a Nexus Ready output.

4.3.33 PC[8:15] / ADDR[8:15] / MDO[0:7] — Port C I/O Pins, External Address Bus and Nexus Secondary

PC[8] to PC[15] are configurable general purpose input or output pins. The pins can be independently configured to provide either high or reduced output drive, and also to enable or disable either a pull-up or pull-down resistor on the pin. The pin is multiplexed with the external address bus lines Address 8 to Address 15. Each pin can be independently configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality. When the Nexus Secondary Port is selected and active, these pin are used as Message Data Out outputs.

4.3.34 PD[0] / $\overline{\text{BWE}}$[0] / MODB — Port D I/O Pin, External Bus Control & Mode Selection

PD[0] is a configurable general purpose input or output pin. The pin can be configured to provide either high or reduced output drive, and also to enable or disable either a pull-up or pull-down resistors on the pin. The state of this pin is latched at the rising edge of $\overline{\text{RESET}}$ to determine the operating mode of the MCU. After Reset the pin is able to be used as the General purpose input or output PD[0], or as a byte select/byte write enable line for devices connected to the external bus. It can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.35 PD[1] / $\overline{\text{BWE}}$[1] / MODA — Port D I/O Pin, External Bus Control & Mode Selection

PD[1] is a configurable general purpose input or output pin. The pin can be configured to provide either high or reduced output drive, and also to enable or disable either a pull-up or pull-down resistors on the pin. The state of this pin is latched at the rising edge of $\overline{\text{RESET}}$ to determine the operating mode of the MCU. After Reset the pin is able to be used as the General purpose input or output PD[1], or as a byte select/byte write enable line for devices connected to the external bus. It can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.36 PD[2] / CLKOUT / $\overline{\text{XCLKS}}$ — Clock Out and Oscillator Selection

PD[2] is a configurable general purpose input pin (Note that general purpose output functionality is not available). The pin can be configured to enable or disable either a pull-up or pull-down resistor on the pin. The state of this pin is latched at the rising edge of $\overline{\text{RESET}}$. Note that, unlike the rest of the hardware configuration pins, the $\overline{\text{XCLKS}}$ pin is not latched during every system reset. It is only latched during a Power On Reset or during a reset after a loss of clock has occurred. After Reset the pin is able to be used as the General purpose input PD[2], or as a system clock for devices connected to the external bus. It can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.37 PD[3] / $\overline{\text{XIRQ}}$ / NMI — Port D I/O Pin, High Priority Interrupt and Non-maskable Interrupt

PD[3] is a configurable general purpose input or output pin. The pin can be configured to provide either high or reduced output drive, and also to enable or disable either a pull-up or pull-down resistors on the pin. It is multiplexed with the high priority interrupt request input that provides a means of applying asynchronous external interrupt requests. It can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality. When the system is in NMI mode, this pin is used exclusively for an external Non-Maskable Interrupt, and may not be used as a GPIO.

4.3.38 PD[4] / $\overline{\text{IRQ}}$ — Port D I/O Pin, and Maskable Interrupt

PD[4] is a configurable general purpose input or output pin. The pin can be configured to provide either high or reduced output drive, and also to enable or disable either a pull-up or pull-down resistors on the pin. It is multiplexed with the low priority interrupt request input that provides a means of applying asynchronous external interrupt requests. It can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.39 PD[5:10] / ADDR[16:21] — Port D I/O Pins and External Address Bus

PD[5] to PD[10] are configurable general purpose input or output pins. They can be independently configured to provide either high or reduced output drives, and also to enable or disable either a pull-up or pull-down resistor on the pins. The pin is multiplexed with the external address bus lines Address 16 to Address 21. Each pin can be independently configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.40 PD[11] / $\overline{\text{OE}}$ — Port D I/O Pin and External Bus Control

PD[11] is a configurable general purpose input or output pin. The pin can be configured to provide either high or reduced output drive, and also to enable or disable either a pull-up or pull-down resistors on the pin. It is multiplexed with the Output Enable output for devices connected to the external bus. It can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.41 PD[12] / $\overline{\text{Burst}}$ — Port D I/O Pin and External Bus Control

PD[12] is a configurable general purpose input or output pin. The pin can be configured to provide either high or reduced output drive, and also to enable or disable either a pull-up or pull-down resistors on the pin. It is multiplexed with the Burst Attribute output for devices connected to the external bus. It can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.42 PD[13] / $\overline{\text{TA}}$ — Port D I/O Pin and External Bus Control

PD[13] is a configurable general purpose input or output pin. The pin can be configured to provide either high or reduced output drive, and also to enable or disable either a pull-up or pull-down resistors on the

pin. It is multiplexed with the Transfer Acknowledge input for devices connected to the external bus. It can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.43 PD[15] / $\overline{\text{CS0}}$ — Port D I/O Pin and External Bus Control

PD[15] is a configurable general purpose input or output pin. The pin can be configured to provide either high or reduced output drive, and also to enable or disable either a pull-up or pull-down resistors on the pin. It is multiplexed with the Chip Select 0 output for devices connected to the external bus. It can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.44 PD[15] / R/$\overline{\text{W}}$ — Port D I/O Pin and External Bus Control

PD[15] is a configurable general purpose input or output pin. The pin can be configured to provide either high or reduced output drive, and also to enable or disable either a pull-up or pull-down resistors on the pin. It is multiplexed with the Read/Write output for devices connected to the external bus. It can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.45 PE[0:15] / AN_A[00:15]— Port E I/O Pins and ATD_A

PE[0] to PE[15] are configurable general purpose input pins (Note that general purpose output functionality is not available). They can be configured to enable or disable either a pull-up or pull-down resistor on the pins. The pins are multiplexed with the analog inputs AN_A[00:15] for the analog to digital converter (ATD_A), and may be used either as an Analog Channel or an external ATD trigger. Each pin can be independently configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.46 PF[0] / EMIOS[0] / NEXPS — Port F I/O Pins, eMIOS Channels and Nexus Port Selection

PF[0] is a configurable general purpose input or output pin. It can be independently configured to provide either high or reduced output drive, and also to enable or disable either a pull-up or pull-down resistor on the pin. The pin is multiplexed with the eMIOS channels for the capturing of timed events or the output of time based signals. At Reset the value of the pin is used to determine the position of the Nexus Port (when enabled). NEXPS cleared low selects the Primary Nexus Port (PA[0:6]), NEXPS set high selects the Secondary Nexus Port (PE[0:6]). The pin can be independently configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.47 PF[1] / EMIOS[1] / NEXPR — Port F I/O Pins, eMIOS Channels and Nexus Present Selection

PF[1] is a configurable general purpose input or output pin. It can be independently configured to provide either high or reduced output drive, and also to enable or disable either a pull-up or pull-down resistor on

the pin. The pin is multiplexed with the eMIOS channels for the capturing of timed events or the output of time based signals. At Reset the value of the pin is used to determine if the Nexus Port can be enabled using the external Debug. The pin can be independently configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.48 PF[2] / EMIOS[2] / AUTOACK — Port F I/O Pins, eMIOS Channels and FlexBus Ack Selection

PF[2] is a configurable general purpose input or output pin. It can be independently configured to provide either high or reduced output drive, and also to enable or disable either a pull-up or pull-down resistor on the pin. The pin is multiplexed with the eMIOS channels for the capturing of timed events or the output of time based signals. At Reset the value of the pin is used to determine whether the AutoAck feature is enabled for the Global Chip Select of the FlexBus. The pin can be independently configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.49 PF[3] / EMIOS[3] / AUTOACK — Port F I/O Pins, eMIOS Channels and FlexBus Port Size

PF[3] is a configurable general purpose input or output pin. It can be independently configured to provide either high or reduced output drive, and also to enable or disable either a pull-up or pull-down resistor on the pin. The pin is multiplexed with the eMIOS channels for the capturing of timed events or the output of time based signals. At Reset the value of the pin is used to determine the Port Size for the Global Chip Select of the FlexBus. The pin can be independently configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.50 PF[4:7] / EMIOS[4:7] — Port F I/O Pins and eMIOS Channels

PF[4] to PF[7] are configurable general purpose input or output pins. They can be independently configured to provide either high or reduced output drives, and also to enable or disable either a pull-up or pull-down resistor on the pins. The pins are multiplexed with the eMIOS channels for the capturing of timed events or the output of time based signals. Each pin can be independently configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.51 PF[8] / PCS[5] / $\overline{\text{PCSS}}$ — Port F I/O Pin and DSPI_C

PF[8] is a configurable general purpose input or output pin. It can be configured to provide either high or reduced output drive, to enable or disable either a pull-up or pull-down resistor on the pin or to provide either an open drain or push-pull pad. The pin can be configured as chip select pin PCS[5] when in Master mode for the Serial Peripheral Interface C (DSPI_C), or as a Peripheral Chip Select Strobe to qualify the other Chip Select signals to eliminate any decoding glitches. Each pin can be independently configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.52 PF[9] / PCS[3] — Port F I/O Pin and DSPI_C

PF[9] is a configurable general purpose input or output pin. It can be configured to provide either high or reduced output drives, to enable or disable either a pull-up or pull-down resistors on the pin or to provide either an open drain or push-pull pad. The pins can be configured as a chip select pin PCS[3] when in Master mode for the Serial Peripheral Interface C (DSPI_C). It can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.53 PF[10] / PCS[2] — Port F I/O Pin and DSPI_C

PF[10] is a configurable general purpose input or output pin. It can be configured to provide either high or reduced output drives, to enable or disable either a pull-up or pull-down resistors on the pin or to provide either an open drain or push-pull pad. The pins can be configured as a chip select pin PCS[2] when in Master mode for the Serial Peripheral Interface C (DSPI_C). It can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.54 PF[11] / SCK_C — Port F I/O Pin and DSPI_C

PF[11] is a configurable general purpose input or output pin. It can be configured to provide either high or reduced output drive, to enable or disable either a pull-up or pull-down resistor on the pin or to provide either an open drain or push-pull pad. The pin can be configured as serial clock pin SCK of the Serial Peripheral Interface C (DSPI_C). It can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.55 PF[12] / PCS[1] — Port F I/O Pin and DSPI_C

PF[12] is a configurable general purpose input or output pin. It can be configured to provide either high or reduced output drives, to enable or disable either a pull-up or pull-down resistors on the pin or to provide either an open drain or push-pull pad. The pins can be configured as a chip select pin PCS[1] when in Master mode for the Serial Peripheral Interface C (DSPI_C). It can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.56 PF[13] / SOUT_C — Port F I/O Pin and DSPI_C

PF[31] is a configurable general purpose input or output pin. It can be configured to provide either high or reduced output drive, to enable or disable either a pull-up or pull-down resistor on the pin or to provide either an open drain or push-pull pad. The pin can be configured as the output pin of the Serial Peripheral Interface C (DSPI_C). It can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.57 PF[14] / PCS[0] / $\overline{\text{SS}}$[0] — Port F I/O Pin and DSPI_C

PF[14] is a configurable general purpose input or output pin. It can be configured to provide either high or reduced output drive, to enable or disable either a pull-up or pull-down resistor on the pin or to provide either an open drain or push-pull pad. The pin can be configured as chip select pin PCS[0] when in Master mode, or as slave select pin $\overline{\text{SS}}$ when in Slave mode for the Serial Peripheral Interface C (DSPI_C). It can

be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.58 PF[15] / SIN_C — Port F I/O Pin and DSPI_C

PF[15] is a configurable general purpose input or output pin. It can be configured to provide either high or reduced output drive, to enable or disable either a pull-up or pull-down resistor on the pin or to provide either an open drain or push-pull pad. The pin can be configured as the input pin of the Serial Peripheral Interface C (DSPI_C). It can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.59 PG[0] / RXD_B — PORT G I/O Pin and ESCI_B

PG[0] is a configurable general purpose input or output pin. The pin can be configured to provide either high or reduced output drive, and also to enable or disable either a pull-up or pull-down resistors on the pin. It is multiplexed with the receive pin for the Enhanced Serial Communications Interface controller B (ESCI_B). It can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.60 PG[1] / TXD_B — PORT G I/O Pin and ESCI_B

PG[1] is a configurable general purpose input or output pin. The pin can be configured to provide either high or reduced output drive, and also to enable or disable either a pull-up or pull-down resistors on the pin. It is multiplexed with the transmit pin for the Enhanced Serial Communications Interface controller B (ESCI_B). It can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.61 PG[2] / RXD_A / NEX1MDO — PORT G I/O Pin, ESCI_A and Nexus Primary

PG[2] is a configurable general purpose input or output pin. The pin can be configured to provide either high or reduced output drive, and also to enable or disable either a pull-up or pull-down resistors on the pin. It is multiplexed with the receive pin for the Enhanced Serial Communications Interface controller A (ESCI_A). It can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality. When the Nexus Primary Port is selected and active, this pin is used as a Message Data Out output.

4.3.62 PG[3] / TXD_A / NEX1MDO — PORT G I/O Pin, ESCI_A and Nexus Primary

PG[3] is a configurable general purpose input or output pin. The pin can be configured to provide either high or reduced output drive, and also to enable or disable either a pull-up or pull-down resistors on the pin. It is multiplexed with the transmit pin for the Enhanced Serial Communications Interface controller A (ESCI_A). It can be configured to wake the system out of low-power mode (DOZE) when an external

signal is applied using the interrupt functionality. When the Nexus Primary Port is selected and active, this pin is used as a Message Data Out output.

4.3.63 PG[4] / TCNTX_A / NEX1MDO[2] — PORT G I/O Pin, FlexCAN_A and Nexus Primary

PG[4] is a configurable general purpose input or output pin. The pin can be configured to provide either high or reduced output drive, and also to enable or disable either a pull-up or pull-down resistors on the pin. It is multiplexed with the transmit pin for the CAN controller A (FlexCAN_A). It can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality. When the Nexus Primary Port is selected and active, this pin is used as a Message Data Out output.

4.3.64 PG[5] / CNRX_A — PORT G I/O Pin and FlexCAN_A

PG[5] is a configurable general purpose input or output pin. The pin can be configured to provide either high or reduced output drive, and also to enable or disable either a pull-up or pull-down resistors on the pin. It is multiplexed with the receive pin for the CAN controller A (FlexCAN_A). It can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.65 PG[6] / CNTX_B — PORT G I/O Pin and FlexCAN_B

PG[6] is a configurable general purpose input or output pin. The pin can be configured to provide either high or reduced output drive, and also to enable or disable either a pull-up or pull-down resistors on the pin. It is multiplexed with the transmit pin for the CAN controller B (FlexCAN_B). It can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.66 PG[7] / CNRX_B — PORT G I/O Pin and FlexCAN_B

PG[7] is a configurable general purpose input or output pin. The pin can be configured to provide either high or reduced output drive, and also to enable or disable either a pull-up or pull-down resistors on the pin. It is multiplexed with the receive pin for the CAN controller B (FlexCAN_B). It can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.67 PG[8] — PORT G I/O Pin

PG[8] is a configurable general purpose input or output pin. The pin can be configured to provide either high or reduced output drive, and also to enable or disable either a pull-up or pull-down resistors on the pin. It can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.68 PG[9] — PORT G I/O Pin

PG[9] is a configurable general purpose input or output pin. The pin can be configured to provide either high or reduced output drive, and also to enable or disable either a pull-up or pull-down resistors on the pin. It can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.69 PG[10] — PORT G I/O Pin

PG[10] is a configurable general purpose input or output pin. The pin can be configured to provide either high or reduced output drive, and also to enable or disable either a pull-up or pull-down resistors on the pin. It can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.70 PG[11] — PORT G I/O Pin

PG[8] is a configurable general purpose input or output pin. The pin can be configured to provide either high or reduced output drive, and also to enable or disable either a pull-up or pull-down resistors on the pin. It can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.71 PG[12] / PCS[4] — Port G I/O Pin and DSPI_A

PG[12] is a configurable general purpose input or output pin. It can be configured to provide either high or reduced output drives, to enable or disable either a pull-up or pull-down resistors on the pin or to provide either an open drain or push-pull pad. The pins can be configured as a chip select pin PCS[4] when in Master mode for the Serial Peripheral Interface A (DSPI_A). It can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.72 PG[13] / PCS[3] — Port G I/O Pin and DSPI_A

PG[13] is a configurable general purpose input or output pin. It can be configured to provide either high or reduced output drives, to enable or disable either a pull-up or pull-down resistors on the pin or to provide either an open drain or push-pull pad. The pins can be configured as a chip select pin PCS[3] when in Master mode for the Serial Peripheral Interface A (DSPI_A). It can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.73 PG[14] / PCS[4] — Port G I/O Pin and DSPI_B

PG[14] is a configurable general purpose input or output pin. It can be configured to provide either high or reduced output drives, to enable or disable either a pull-up or pull-down resistors on the pin or to provide either an open drain or push-pull pad. The pins can be configured as a chip select pin PCS[4] when in Master mode for the Serial Peripheral Interface B (DSPI_B). It can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.3.74 PG[15] / PCS[3] — Port G I/O Pin and DSPI_B

PG[15] is a configurable general purpose input or output pin. It can be configured to provide either high or reduced output drives, to enable or disable either a pull-up or pull-down resistors on the pin or to provide either an open drain or push-pull pad. The pins can be configured as a chip select pin PCS[3] when in Master mode for the Serial Peripheral Interface B (DSPI_B). It can be configured to wake the system out of low-power mode (DOZE) when an external signal is applied using the interrupt functionality.

4.4 Power Supply Pins

MAC72xx power and ground pins are described below.

NOTE

All VSS pins must be connected together in the application.

4.4.1 VPP — Power For Flash Program and Erase

VPP is the power supply and ground input pins for the program and erase circuitry inside the Flash. If this pin is grounded or left open, programming and erasing of the Flash will be unavailable, and the system will have unpredictable behavior in the case where Flash programming or erasing is attempted.

4.4.2 VDDX1-4,6-11, VSSX1-11 (except VDDX5) — Power and Ground Pins for I/O Drivers

External power and ground for I/O drivers. Because fast signal transitions place high, short-duration current demands on the power supply, use bypass capacitors with high-frequency characteristics and place them as close to the MCU as possible. Bypass requirements depend on how heavily the MCU pins are loaded. If one or more of these pins are left open, the ability of the I/O to drive loads could be compromised, resulting in slower transition times on outputs.

4.4.3 VDDX5 /VDDAPASS — Power Pin for I/O Drivers and Control Voltage for Internal Pass Transistors

External power for I/O drivers and control voltage for internal pass transistors. Because fast signal transitions place high, short-duration current demands on the power supply, use bypass capacitors with high-frequency characteristics and place them as close to the MCU as possible. Bypass requirements depend on how heavily the MCU pins are loaded. If this pin is left open, the internal voltage regulator may not be able to control the internal pass transistors, resulting in a completely non-functional device.

4.4.4 VDDR/VREGEN — Power Pin for the Internal Voltage Regulator

This 5.0V supply pin supplies all current for the 1.5V and 3.3V on-chip regulated supplies. This pin will supply most of the current consumed by the device, except for the I/O, which is supplied by the VDDx/VSSx pad pairs. If this pin is grounded, the internal voltage regulation is disabled, and all 1.5V and 3.3V supplies must be supplied from external sources through the VDD15x/VSS15x, VDD33/VSS33a and VDDPLL/VSSPLL pin pairs.

4.4.5 VDD15a, VSS15a — Core Power Pins

These pins provide the operating voltage and ground for the MCU core logic. Because fast signal transitions place high, short-duration current demands on the power supply, use bypass capacitors with high-frequency characteristics and place them as close to the MCU as possible. This 1.5V supply is derived from the internal voltage regulator when the voltage regulator is enabled. Otherwise, it must be supplied externally. There is no static load on these pins allowed. The internal voltage regulator is turned off, if VREGEN is tied to ground. If the VDD15a pin is grounded or left open, the device will be completely non-functional.

NOTE

No load allowed except for bypass capacitors.

4.4.6 VDD15c/VDDF, VSS15c/VSSF — Core and Flash Logic Power Pins

These pins provide the operating voltage and ground for the MCU and Flash core logic. Because fast signal transitions place high, short-duration current demands on the power supply, use bypass capacitors with high-frequency characteristics and place them as close to the MCU as possible. This 1.5V supply is derived from the internal voltage regulator when the voltage regulator is enabled. Otherwise, it must be supplied externally. There is no static load on these pins allowed. The internal voltage regulator is turned off, if VREGEN is tied to ground. If the VDD15c pin is grounded or left open, the MCU will be completely non-functional.

NOTE

No load allowed except for bypass capacitors.

4.4.7 VDD33/VFLASH, VSS33 — Flash and I/O Pre-Driver Power Pins

These pins provide the operating voltage and ground for the Flash and I/O pre-drivers. Because fast signal transitions place high, short-duration current demands on the power supply, use bypass capacitors with high-frequency characteristics and place them as close to the MCU as possible. This 3.3V supply is derived from the internal voltage regulator when the voltage regulator is enabled. Otherwise, it must be supplied externally. There is no static load on these pins allowed. The internal voltage regulator is turned off, if VREGEN is tied to ground. If the VDD33 pin is grounded or left open, the Flash and all I/O outputs will be completely non-functional.

NOTE

No load allowed except for bypass capacitors.

4.4.8 VDDA, VSSA — Power Supply Pins for ATD and Voltage Regulator Control

These pins provide the operating voltage and ground for the ATD and the Voltage Regulator control block. This allows the supply voltage to the ATD to be bypassed independently of the main I/O supply VDDx and VSSx. If the VDDA pin is grounded or left open, the ATD will be completely non-functional. If the VDDA pin is grounded or left open, and the internal regulator is not bypassed, then the complete MCU

will be non-functional. It is critical to the operation of the ATD and the MCU that this supply be kept as noise-free as possible.

4.4.9 VRH, VRL — ATD Reference Voltage Input Pins

VRH and VRL are the reference voltage input pins for the analog to digital converter.

4.4.10 REFBYPC — ATD Reference Voltage Bypass Capacitor

REFBYPC is used to filter the 3/4 VRH reference voltage. A 100nF external capacitor should be placed between this pin and the VRL pin.

4.4.11 VDDPLL, VSSPLL — Power Supply Pins for PLL

These pins provide the operating voltage and ground for the Oscillator and the Phased-Locked Loop. This allows the supply voltage to the Oscillator and PLL to be bypassed independently.This 3.3V voltage is generated by the internal voltage regulator when the voltage regulator is enabled. Otherwise, it must be supplied externally.

NOTE

No load allowed except for bypass capacitors.

4.4.12 VSS-TEST — Power Supply Pin

This input only pin is reserved for test. The pin must be tied to VSS in all applications. If this pin is shorted to a VDD supply the operation of the device will be unpredictable. If the pin is left open, an internal pulldown will protect the device operation.

Table 4-2. MAC72xx Power and Ground Connection Summary

Mnemonic	Pin Number		Nominal Voltage	Description
	144 QFP	100 QFP		
V_{DD15a}	127	92	1.5 V	Provides operating voltage and ground for the MCU core logic, including the MCU core, all peripherals, and all RAMs. This allows the supply voltage to the core to be bypassed independently. When not bypassed, the core supply is generated by the internal voltage regulators, and these pins are used for an external bypass capacitor.
V_{SS15a}	126	91	0V	

Table 4-2. MAC72xx Power and Ground Connection Summary

Mnemonic	Pin Number		Nominal Voltage	Description
	144 QFP	100 QFP		
V_{DD15c}	52	37	1.5 V	Provides operating voltage and ground for the MCU core logic, including the MCU core, all peripherals, and all RAMs. This allows the supply voltage to the core to be bypassed independently. When not bypassed, the core supply is generated by the internal voltage regulators, and these pins are used for an external bypass capacitor.
V_{SS15c}	53	38	0V	
V_{DD33}	124	89	3.3 V	Provides operating voltage and ground for the Flash and I/O pre-drivers. This allows the supply voltage to the Flash and I/O pre-drivers to be bypassed independently. When not bypassed, the Flash and I/O pre-driver supply is generated by the internal voltage regulators, and these pins are used for an external bypass capacitor.
V_{SS33}	125	90	0V	
V_{DDR}	55	40	5.0 V	Supplies external power for all internal voltage regulation.
$V_{DDX1-4,6-11}$	10,20,49,63, 67,80,120	14,34, 58,87	5.0 V	Supplies external power and ground for the I/O drivers.
$V_{SSX1-11}$	9,19,30,48, 62,79,95,119	13,33,57, 63,86	0 V	
V_{DDX5}	96	64	5.0 V	Supplies external power and ground for the I/O drivers, as well as the control voltage for all internal regulator pass transistors.
V_{DDA}	109	76	5.0 V	Operating voltage and ground for the analog-to-digital converters and the reference for the internal voltage regulator, allows the supply voltage to the A/D to be bypassed independently.
V_{SSA}	112	79	0 V	
V_{RH}	110	77	5.0 V	Supplies external reference voltages for the analog-to-digital converter.
V_{RL}	111	78	0 V	
V_{DDPLL}	56	41	3.3 V	Provides operating voltage and ground for the Phased-Locked Loop and Oscillator. This allows the supply voltage to the PLL and Oscillator to be bypassed independently. When not bypassed, the PLL and Oscillator supply is generated by the internal voltage regulators, and these pins are used for an external bypass capacitor.
V_{SSPLL}	58	43	0 V	
V_{PP}	54	39	5.0V	Supplies external power for program and erase of the Flash.
$V_{SS-TEST}$	61	46	0V	Pin must be tied to Ground on all applications.

Chapter 5
System Clock Description

5.1 Clocks Introduction

The Clock and Reset Generator (CRG) provides the internal clock signals for the core and all peripheral modules. Figure 5-1 shows the clock connections from the CRG to all modules. There are six clocks available on the MAC72xx:

Table 5-1. Clocks Summary

PLLCLK	This is the clock output of the on-chip PLL. When the PLL is enabled, the System Clock is driven directly by this clock. When the system is in Self Clock Mode (SCM), this clock drives the Real Time Interrupt (RTI) and Software Watchdog Timer (SWT).
System Clock	This clock is used throughout the MCU to drive the core, the memories and select peripherals. It may be driven either by the PLLCLK (when the PLL is enabled) or by the OSCCLK (when the PLL is bypassed). For a list of which modules are driven by this clock, please see Table 5-3.
Peripheral Bus Clock	This clock is used throughout the MCU to drive the remainder of the peripherals that are not driven by the System Clock. It is always running at 1/2 the frequency of the System Clock. For a list of which modules are driven by this clock, please see Table 5-3.
Oscillator Clock	This clock is driven directly from the output of the on-chip oscillator. Programming the **CTRL** register inside the FlexCAN will allow the FlexCAN to use either the Oscillator Clock or Peripheral Bus Clock.
RTI Clock	This clock is driven by OSCCLK in normal operation, and by PLLCLK when the system is in Self Clock Mode. It is used to clock the on-chip Real Time Interrupt (RTI) inside the PIT module.
SWT Clock	This clock is driven by OSCCLK in normal operation, and by PLLCLK when the system is in Self Clock Mode. It is used to clock the on-chip Software Watchdog Timer (SWT) inside the MCM module.

NOTE

It is possible to configure the PLL to generate a System Clock frequency higher than that supported by the design of the device. It is the responsibility of the user to ensure that the device is operated within its specified limits at all times.

Please refer to Chapter 23, “Clock and Reset Generator (CRG)” for details on clock generation.

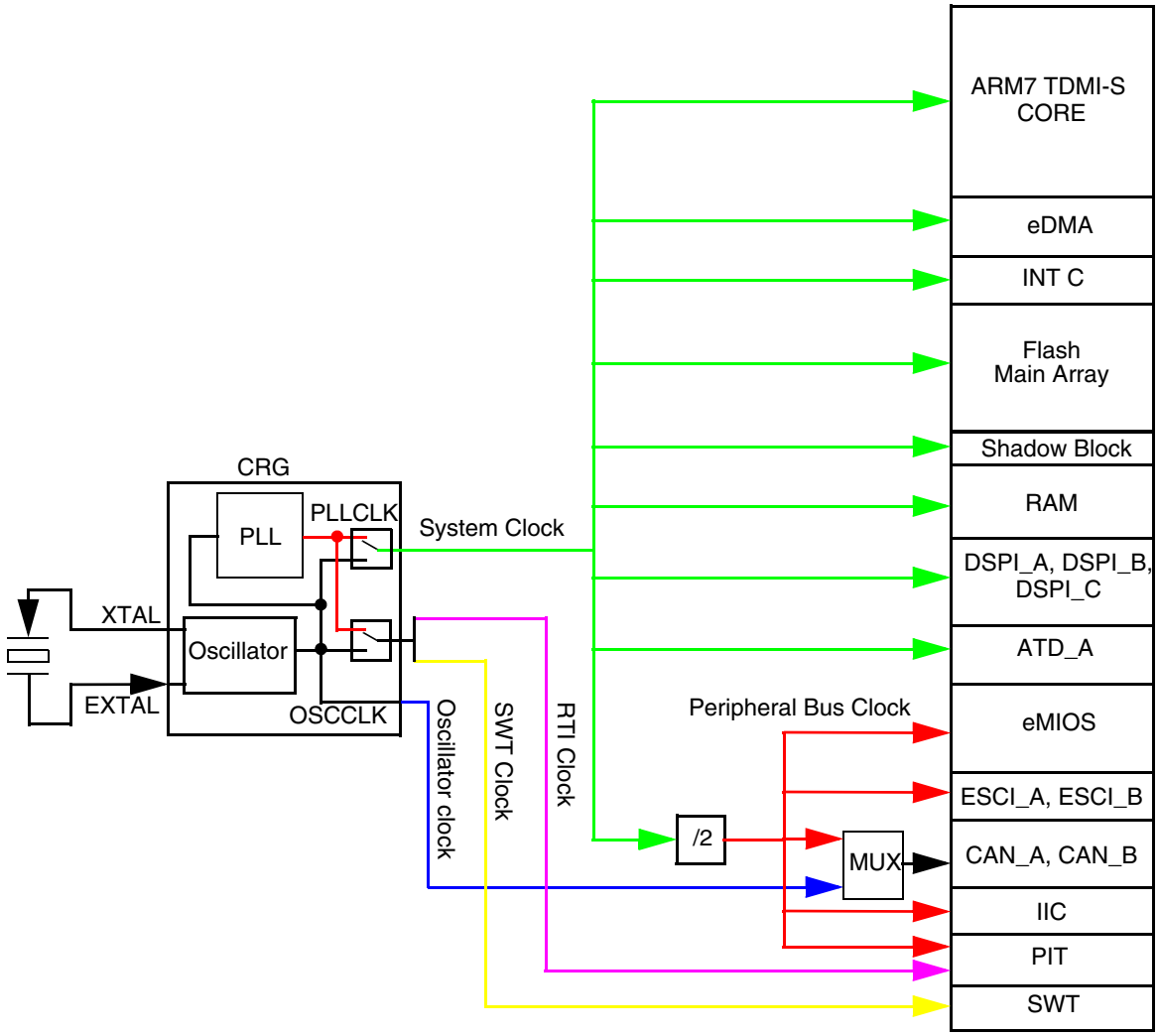


Figure 5-1. MAC72xx Clock Tree

The MCU's clocks can be supplied in a variety of ways, enabling a range of system operating frequencies to be supported:

- From the on-board Phase Locked Loop (PLL) in normal functional mode
- From the PLL in Self Clock Mode (SCM)
- Directly from the Oscillator

The FlexCAN modules may be configured to have their clock sources derived either from the Peripheral Bus Clock or directly from the Oscillator Clock. This allows the user to select the CAN clock based on the required jitter performance. Consult Chapter 28, "FlexCAN2" for more details on the operation and configuration of the CAN modules.

The Real Time Interrupt (RTI) (in the Periodic Interrupt Timer module) and the Software Watchdog Timer (SWT) (in the MCM module) can also be configured to run from either the Oscillator Clock or the PLL

generated clock (in Self Clock Mode only). This allows these functions to continue to run during low power operating modes if required. Refer to Chapter 26, “Periodic Interrupt Timer (PIT_RTI)” and Chapter 13, “Miscellaneous Control Module (MCM)” for more information on these modules.

In order to ensure the presence of the clock, the MCU includes an on-chip hardware Crystal Monitor connected to the output of the Oscillator, OSCCLK. The Crystal Monitor can be configured to invoke the PLL self clocking mode or to generate a System Reset to time out as a result of no OSCCLK being present.

In addition to the Crystal Monitor, the MCU also provides a Clock Quality checker which performs a more accurate check of the clock. The Clock Quality Checker counts a predetermined number of clock edges within a defined time window to ensure that the clock is running. The checker can be invoked following specific events such as wake-up or a Crystal Monitor failure.

5.2 Clock Generation

Figure 5-2 and Figure 5-3 shows the clock outputs and gating in the CRG. The effect of the different Low Power modes on the various clocks is discussed in Chapter 3, “Low Power Modes”.

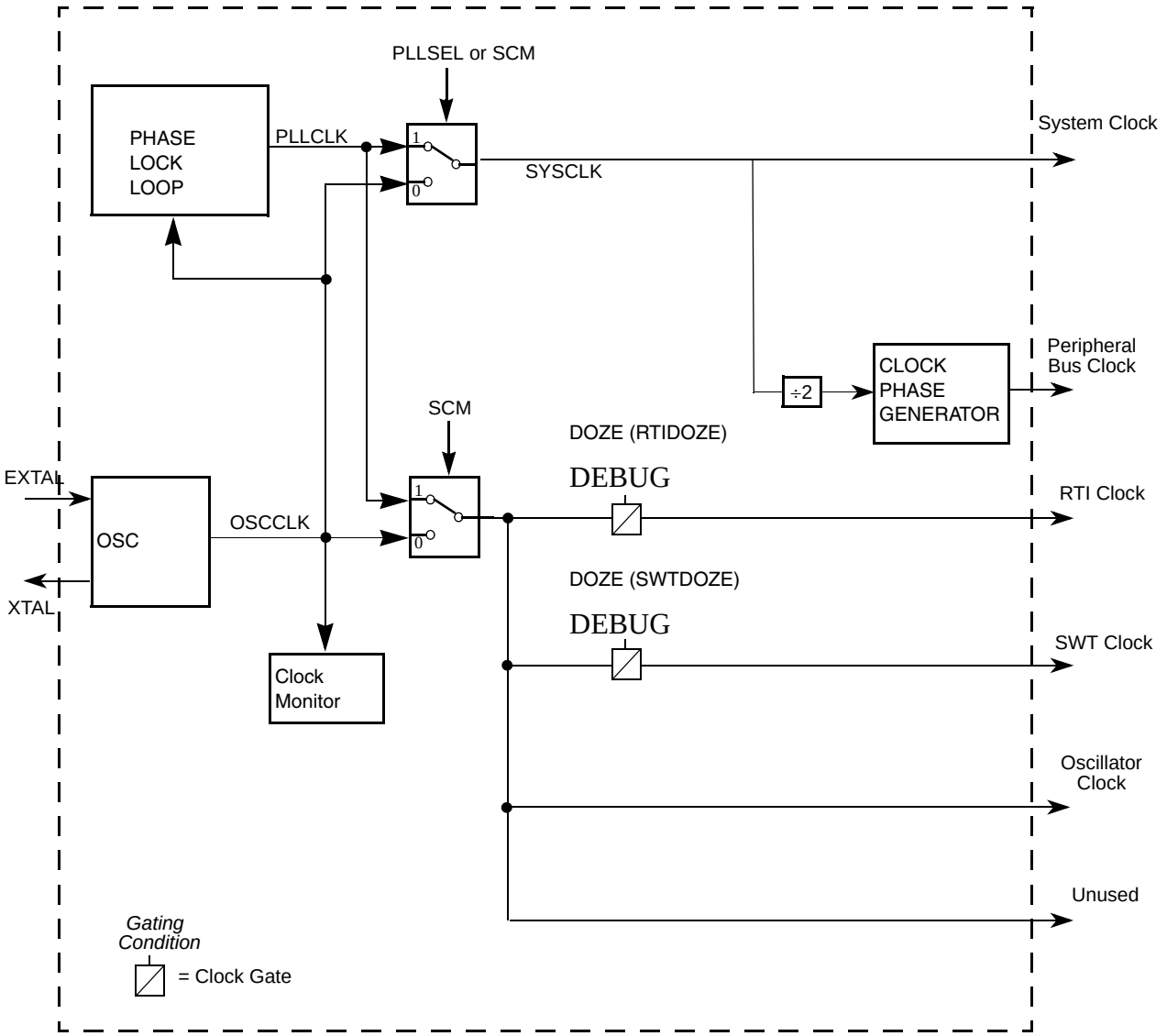


Figure 5-2. CRG Generated Clocks

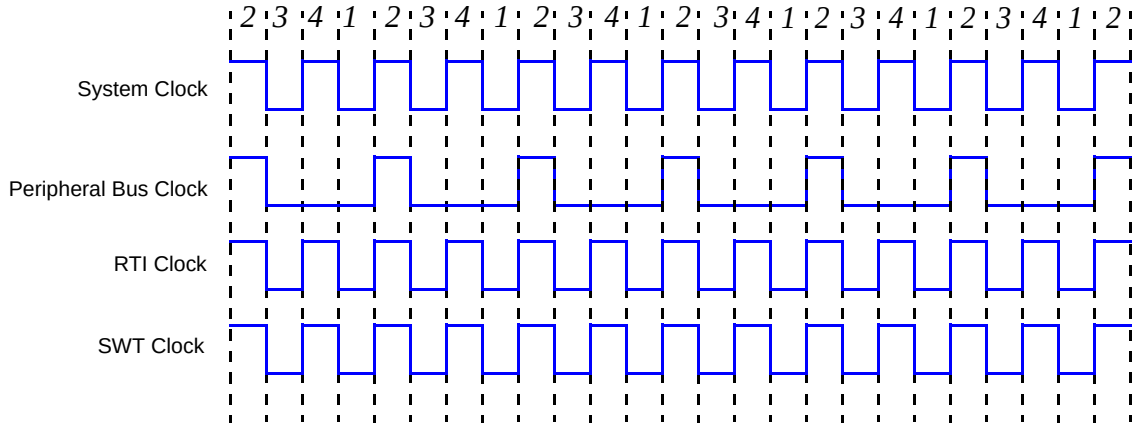


Figure 5-3. Timing of CRG Generated Clocks

5.2.1 Clock Source Selection

With two oscillator modes, and the ability to bypass the PLL, there are four main clock selection modes that the system may be run in:

- Automatic Level Control (ALC) 1:1 Mode - In this mode, the oscillator is enabled, and the PLL is bypassed.
- Automatic Level Control (ALC) PLL Mode - In this mode, the oscillator is enabled, and the PLL is enabled.
- External Clock 1:1 Mode - In this mode, the oscillator is bypassed, and the PLL is bypassed.
- External Clock PLL Mode - In this mode, the oscillator is bypassed, and the PLL is enabled.

For information on selecting the oscillator mode, please refer to Section 24.8.1, “OSC Mode Selection”. The enabling or bypassing of the PLL is done with the PLLSEL bit in the CLKSEL register in the CRG module.

The oscillator mode is chosen by a combination of the JTAG SC4 register and the $\overline{\text{XCLKS}}$ hardware configuration pin, as follows:

Table 5-2. Clock Source Selection

XCLKS	PLLSEL	Mode
1	0	ALC 1:1 Mode
1	1	ALC PLL Mode
0	0	External Clock 1:1 Mode
0	1	External Clock PLL Mode

Note that the PLL always comes up disabled (PLLSEL = 0) after a System or Power On Reset, and must be explicitly enabled by the boot software.

In addition to the above modes, the FlexCAN, SWT and RTI all have some level of configurability with respect to clock source.

5.2.1.1 ALC 1:1 Mode

In this mode, the system clock is fed from the Oscillator, and the PLL is bypassed. Refer to Figure 5-4.

$\overline{\text{XCLKS}}$ = 1

PLLSEL = 0 (PLL is bypassed)

$$\text{SystemClock} = \text{OSCCLOCK} \quad \textit{Eqn. 5-1}$$

$$\text{PeripheralBusClock} = \frac{\text{SystemClock}}{2} = \frac{\text{OSCCLK}}{2} \quad \textit{Eqn. 5-2}$$

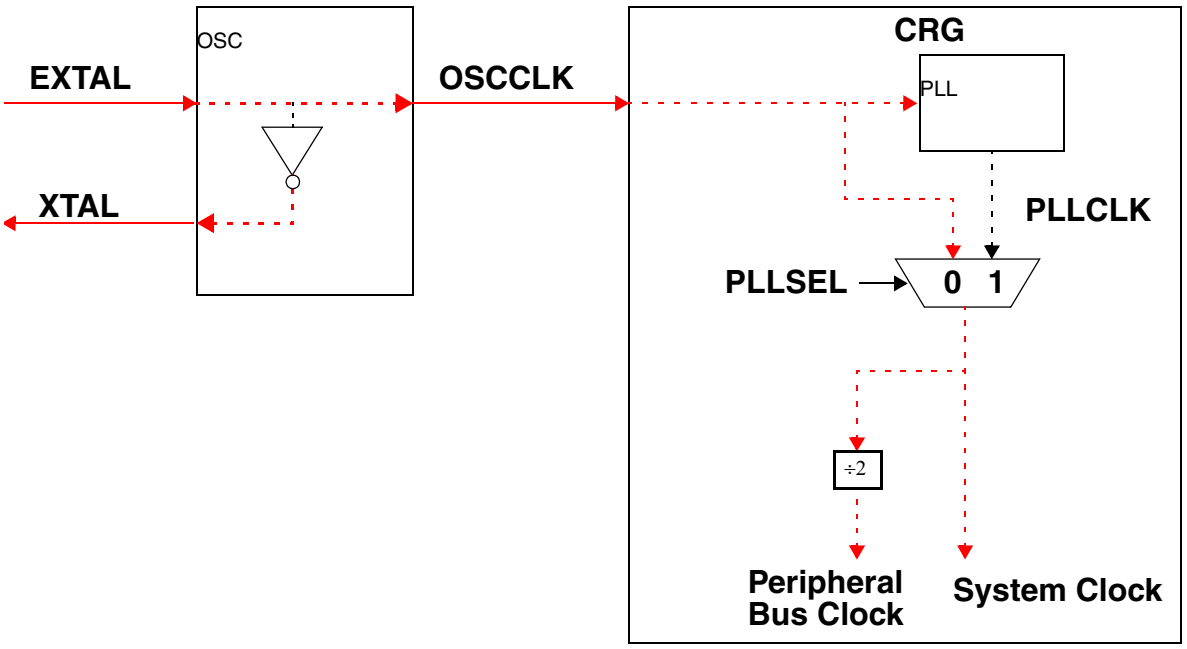


Figure 5-4. ALC 1:1 Mode Clock Path

5.2.1.2 ALC PLL Mode

In this mode, the system clock is fed from the Oscillator, and the PLL is enabled. Refer to Figure 5-5.

$\overline{\text{XCLKS}}$ = 1

PLLSEL = 1 (PLL is not bypassed)

$$\text{SystemClock} = \text{PLLCLK} = 2 \times \text{OSCCLK} \times \frac{\text{SYNR} + 1}{\text{REFDV} + 1} \quad \textit{Eqn. 5-3}$$

$$\text{PeripheralBusClock} = \frac{\text{SystemClock}}{2} = \frac{\text{PLLCLK}}{2} \quad \textit{Eqn. 5-4}$$

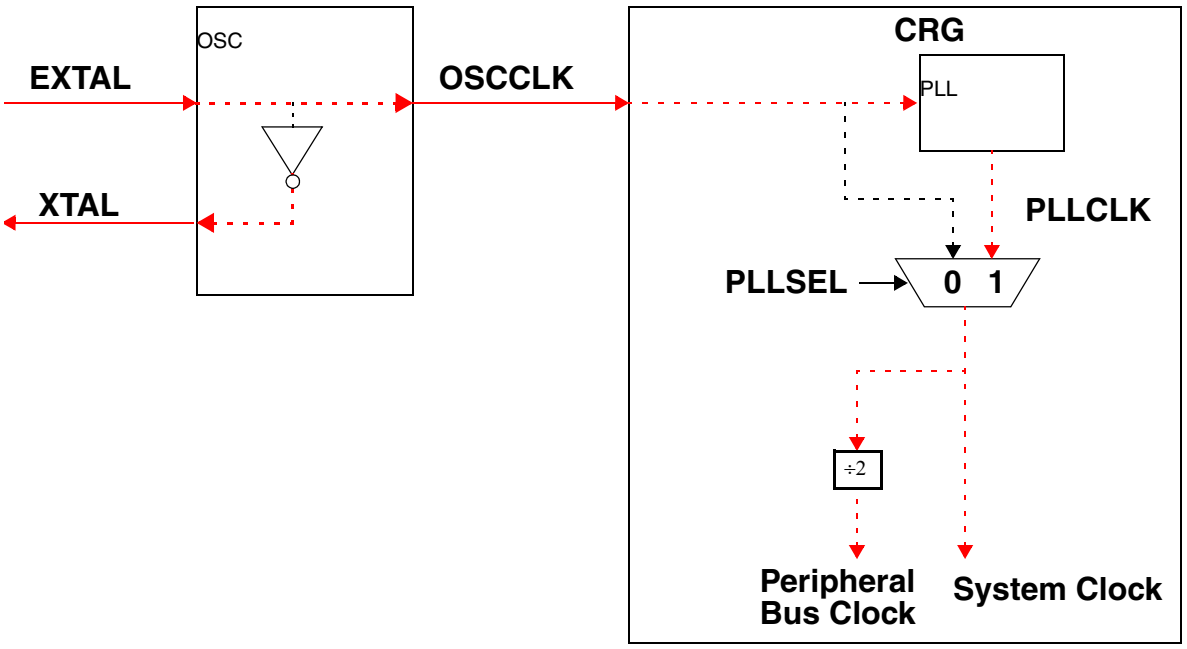


Figure 5-5. ALC PLL Mode Clock Path

5.2.1.3 External Clock 1:1 Mode

In this mode, the system clock is fed from an external 3.3V clock, and the PLL is bypassed. Refer to Figure 5-6.

$\overline{\text{XCLKS}} = 0$

PLLSEL = 0 (PLL is bypassed)

$$\text{SystemClock} = \text{OSCCLOCK} \qquad \textit{Eqn. 5-5}$$

$$\text{PeripheralBusClock} = \frac{\text{SystemClock}}{2} = \frac{\text{OSCCLK}}{2} \qquad \textit{Eqn. 5-6}$$

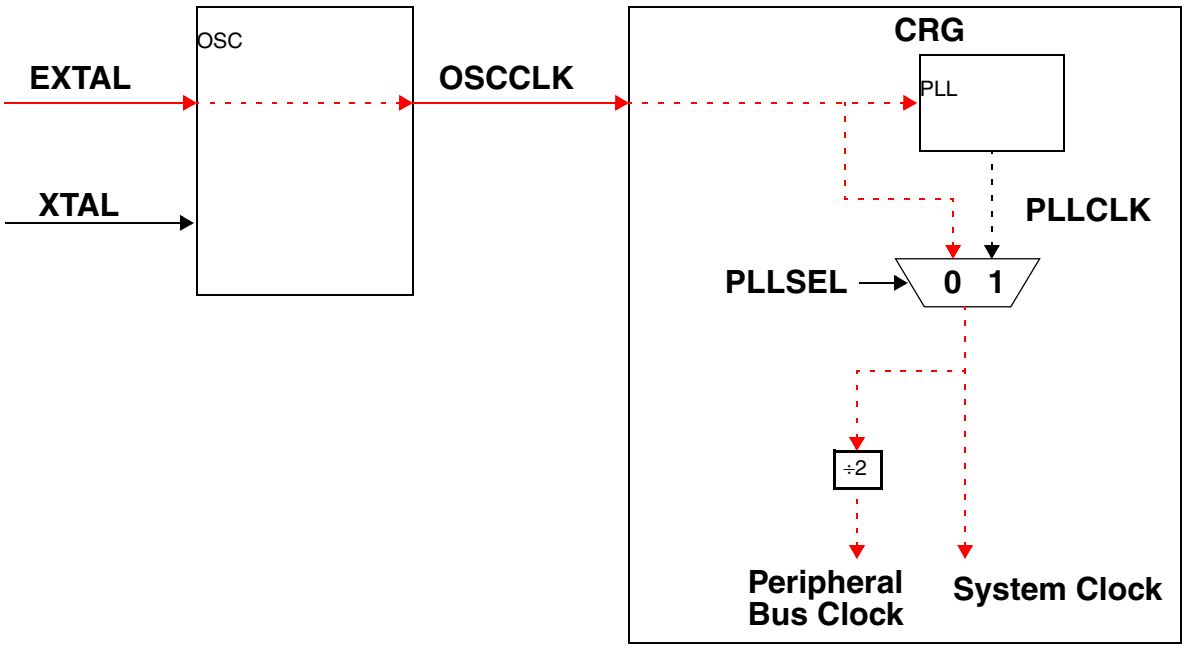


Figure 5-6. External Clock 1:1 Mode Clock Path

5.2.1.4 External Clock PLL Mode

In this mode, the system clock is fed from an external 3.3V clock, and the PLL is enabled. Refer to Figure 5-7.

$\overline{\text{XCLKS}}$ = 0

PLLSEL = 1 (PLL is not bypassed)

$$\text{SystemClock} = \text{PLLCLK} = 2 \times \text{OSCCLK} \times \frac{\text{SYNR} + 1}{\text{REFDV} + 1} \qquad \textit{Eqn. 5-7}$$

$$\text{PeripheralBusClock} = \frac{\text{SystemClock}}{2} = \frac{\text{PLLCLK}}{2} \qquad \textit{Eqn. 5-8}$$

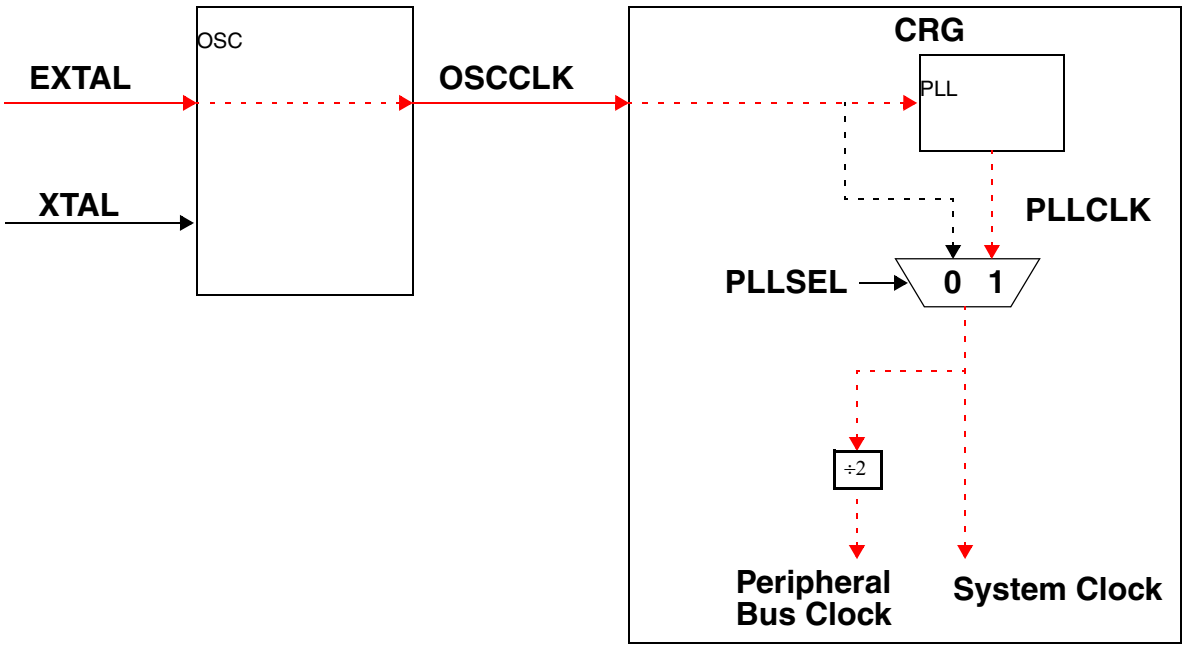


Figure 5-7. External Clock PLL Mode Clock Path

5.2.2 Self Clock Mode (SCM)

If the external clock frequency is not available due to a failure or due to long crystal start-up time, the System Clock is derived from the VCO in the PLL, running at minimum operating frequency. This mode of operation is called Self Clock Mode. Self Clock Mode may be entered at any time, upon failing the Clock Quality Check. Self Clock mode may only be exited by passing a Clock Quality Check after a system reset. When the system is in Self Clock Mode, the RTI, SWT and Oscillator Clock are all automatically switched to PLLCLK.

5.2.3 Crystal Monitor

The MAC72xx provides an on-chip Crystal Monitor that detects a loss of Oscillator Clock. Please refer to Section 23.4.2.3, “Clock Monitor (CM)” for more details. Note that the terms “Crystal Monitor” and “Clock Monitor” are used interchangeably.

5.2.4 Clock Quality Checker

In addition to the Crystal Monitor, the MCU also provides a Clock Quality checker which performs a more accurate check of the Oscillator Clock. Please refer to Section 23.4.2.4, “Clock Quality Checker” for more details.

5.3 Clock Usage

Table 5-3 summarizes the usage of clocks on all MAC72xx modules.

Table 5-3. Module Clock Usage Overview

Module/Pin	System Clock (Fast Clock)	Peripheral Bus Clock (Slow Clock)	Oscillator Clock
ARM7 Core	X		
AXBS	X		
FlexBus	X		
CLKOUT		X	
Nexus	X		
SRAM	X		
INTC	X		
MCM (except SWT)	X		
SWT (in MCM)	X[1]		X[1]
DMA2	X		
AIPS	X		
Flash (array interface)	X		
Flash (registers)		X	
DMA_CH_MUX	X		
IIC		X	
eSCI_A/B		X	
SPI_A/B/C	X		
SSM		X	
PIT (except RTI)		X	
RTI (in PIT)	X[1]		X[1]
FlexCAN_A/B		X[2]	X[2]
ATD_A	X		
eMIOS		X	
PIM		X	
CRG		X	
PTI		X	
BAM	X		

1. PLLCLK is selected when the device is in Self Clock Mode (SCM).
2. Can be switched between the Peripheral Bus Clock and Oscillator Clock via the CLK_SRC bit in the CTRL register in the corresponding FlexCAN.

5.4 Clock Gating

On the MAC72xx, there are two levels of clock gating implemented:

- Global clock gating refers to clock gating inside the CRG module which gates entire clock domains. Figure 5-2 illustrates the global clock gating on the MAC72xx. More information on this can be found in the CRG documentation.
- Local clock gating is implemented inside the CGL, and gates a particular clock for a particular module. Please refer to the individual chapters for information on gating conditions.

A low power STOP mode is not supported on the MAC72xx.

Since STOP mode is not implemented on the MAC72xx, the user may only control the granularity of power consumption in DOZE mode, or via the MDIS bit in each module.This topic is discussed in more detail in Section Chapter 3, “Low Power Modes”.

Figure 5-8 shows the overview of the clock gating and distribution on the MAC72xx.

5.5 Oscillator

Refer to Chapter 24, “Oscillator (OSC) for details on the Oscillator.

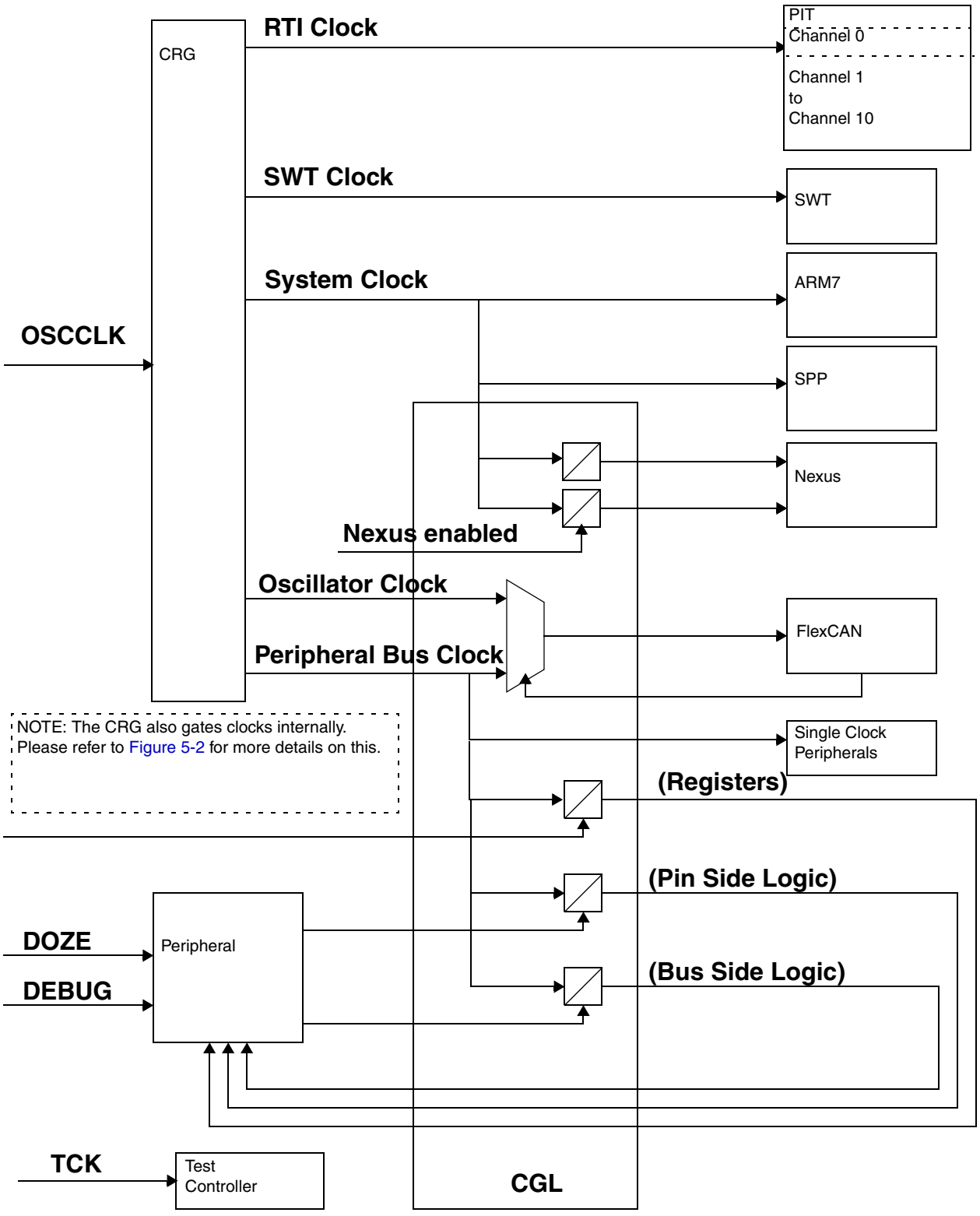


Figure 5-8. System Clock Gating

Chapter 6
Resets

6.1 Resets Introduction

On the MAC72xx, there are three classes of resets:

- Power On Reset (POR)
- System Reset
- Debug Reset

Figure 6-1 illustrates the overall reset distribution on the MAC72xx.

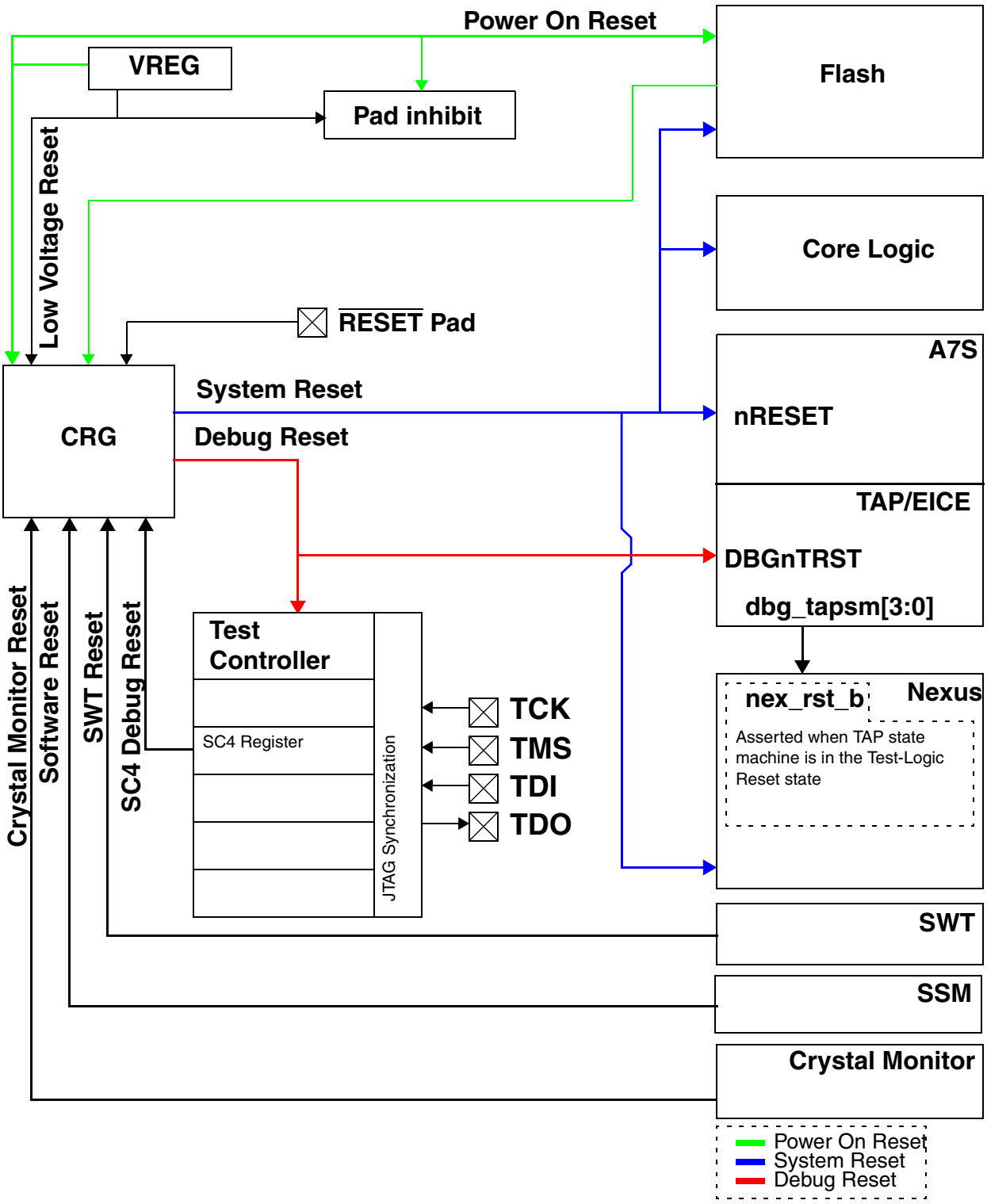


Figure 6-1. MAC72xx Resets

6.2 Power On Reset (POR)

Power On Reset is the reset initiated when the device is being powered up. It is generated by the Voltage Regulator when the voltage supply of the device is below a certain threshold (called the POR trip point). The POR is used to reset or control the following:

- Flash internal circuits
- CRG system reset generation logic
- CRG debug reset generation logic
- Pad/Level shifter inhibitors

6.3 System Reset

The System Reset is the main reset in the system that is used to reset the majority of the logic in the system. It can be caused by any of the following events:

Table 6-1. System Reset Sources

Event	Source	Maskable ?
Power On Reset (POR)	Voltage Regulator	No
Low Voltage Reset (LVR)	Voltage Regulator	No
RESET	External	No
Software Watchdog Timer (SWT)	MCM	Yes (In the MCM)
Crystal Monitor Reset	Crystal Monitor	Yes (In the CRG)
Software Reset	SSM	Yes
Illegal Address	Not implemented	Not implemented

All of the above sources may be detected by software (after the reset) by reading the appropriate register(s) in the CRG module. Note that the use of the MRSR register in the MCM is deprecated.

6.4 Debug Reset

The Debug Reset is generated from the Power-On Reset (POR) of the system, and is intended to reset the debug logic, which should not be reset during a general system reset (i.e.-when the user presses a reset button).

On the MAC72xx, the Debug Reset is initiated by either of the following two conditions:

- a Power-On Reset (POR)
- Setting of bit 5 in the SC4 Test Register (See Section 35.1.1, “JTAG Test Register (SC4)”)

Asserting the debug reset will reset the following logic on the MAC72xx:

- EICE in the ARM7 core, including the TAP state machine
- JTAG synchronization logic in the Test Controller
- SC4 register in the Test Controller

- SC5 register in the Test Controller
- SC6 register in the Test Controller
- SC7 register in the Test Controller
- BIST muxing in the Test Controller
- Scan testpoints in the Test Controller
- The debug reset assertion logic in the Test Controller (i.e.-self clearing reset)
- Nexus registers

What is not explicitly reset are the Nexus registers, which are reset solely by taking the ARM7 TAP state machine into the Test-Logic Reset State ($F). Since the ARM7 core TAP state machine is driven by the debug reset, and resets into the Test-Logic Reset State, this effectively forces the reset of the Nexus registers as well.

6.5 Software Reset

A software initiated reset of the system may be performed by performing a series of writes to the Software Reset register in the SSM module. Please refer to Chapter 25, “System Service Module (SSM_MAC7202)” for details on this. Performing a software reset in this manner is virtually identical to resetting the device via the RESET pin.

6.6 Reset Implementation

Figure 6-1 illustrates the resets in the MAC72xx. Some points to note are:

- The Nexus is reset by cycling the TAP controller in the A7 through the Test-Logic Reset state ($F). This is done inside the Nexus block.
- Since the Debug Reset is asserted during a Power On Reset, and the TAP State Machine is reset to the Test-Logic Reset state on a Debug Reset, the Nexus logic will automatically be reset on a POR event without the need to explicitly use the POR signal in the Nexus.
- Bringing up the MAC72xx in debug mode will consist of the following steps:
  - Power on the MAC72xx. This will reset the ARM7 core, EICE and Nexus.
  - A breakpoint at address $0000 0000 is programmed into the EICE
  - The external reset is asserted, resetting the system except for the EICE and Nexus
  - The processor boots, and fetches its first instruction from $0000 00000, which hits the previously hit breakpoint, and the system enters DEBUG mode.

  Alternatively, the following series of steps may also be used:
  - Power on the MAC72xx, while holding the RESET pin low (asserted). This will reset the EICE and Nexus, but hold the core in a reset state
  - A breakpoint at address $0000 0000 is programmed into the EICE
  - The external reset is released, starting the execution of the core
  - The processor boots, and fetches its first instruction from $0000 00000, which hits the previously hit breakpoint, and the system enters DEBUG mode.
- Switching Nexus ports will consist of the following steps:

— Power on the MAC72xx. This will reset the ARM7 core, EICE and Nexus.
— While RESET is asserted, drive the Nexus Present (PF1) and Nexus Port Select (PF0) lines to enable/disable a particular Nexus Port
— The Nexus is reset by cycling through the Test Reset state in the TAP controller
— Once the Nexus resets, it latches in the EVTI bit from the selected Nexus port.

6.7 Effects of Reset

6.7.1 Hardware Configuration

When a POR or System Reset occurs, the following pins are sampled while the RESET pin is asserted (low) in order to determine the configuration of the system:

- MODA (PD1)
- MODB (PD0)
- NEXPORTSEL (PF0)
- NEXPRESENT (PF1)
- AUTOACK (PF2)
- PORTSIZE (PF3)

In addition, on a POR or System Reset, the security state of the system is determined from the System Censor word in the Flash (See Section 18.7.9, “Flash Security”). This means that a currently secured system may come out of reset as secured, and vice-versa.

The oscillator mode selection is done via the $\overline{\text{XCLKS}}$ hardware configuration pin (PD2), but only during the following conditions:

- POR, *or*
- System Reset after a loss of clock has occurred.

6.7.2 Register States

When a POR or System Reset occurs, MCU registers and control bits are changed to known start-up states. Refer to the respective module chapters for register reset states.

When a Debug Reset occurs, the following registers are reset:

- EICE registers. Please refer to the ARM7 documentation for more information.
- Nexus registers. Please refer to Chapter 11, “A7S Nexus3 Module” for more information.
- TAP State Machine. Please refer to the ARM7 documentation for more information.

6.7.3 Peripheral Disabled State

Following a Power on Reset or a System Reset, the following peripheral modules are disabled, and must be enabled before they can be used:

- eSCI

- Periodic Interrupt Timer (PIT)
- eMIOS (MTS)
- IIC
- FlexCAN
- DSPI

Following a Debug Reset, the Nexus is disabled, and must be explicitly enabled. Please refer to Section 11.1.3, “Modes of Operation” for more information.

6.7.4 I/O pins

Refer to Chapter 34, “Port Integration Module (PIM_MAC7202)” for mode dependent pin configuration of port A, C and D out of Reset. All other pins are configured as General Purpose Inputs out after a POR or System Reset. Debug Reset has no effect on the configuration of GPIO pins.

NOTE

All non-bonded out pins should be configured as outputs or pull-ups or pull-downs enabled after reset in order to avoid current drawn from floating inputs. Refer to Table 4-1 for affected pins.

6.7.5 Memories

Refer to Chapter 9, “Device Memory Map” for locations of the memories depending on the operating mode after Reset.

The RAM array is not automatically initialized out of Reset. Because of the ECC functionality built into the RAM, the array must be initialized using 32-bit writes. After initialization, the array may be accessed using 8, 16 and 32-bit reads and writes.

6.8 System Configuration at Reset

The reset state of the system will be:

- All pads on Ports A-G are placed into GPI mode, except in Expanded Mode, where all external bus pins are placed in Primary Peripheral Mode.
  - Pin A0 to A15 - DATA0 to DATA15
  - Pin C0 to C15 - ADDR0 to ADDR15
  - Pin D0 to D2 - $\overline{\text{BWE0}}$, $\overline{\text{BWE1}}$, $\overline{\text{CLKOUT}}$
  - Pin D5 to D15 - ADDR16 to ADDR21, $\overline{\text{BURST}}$, $\overline{\text{TA}}$, $\overline{\text{CS0}}$, R/$\overline{\text{W}}$
- TDI pad is an input with pull-up enabled
- TDO pad is an output with fastest slew rate selected
- TCK pad is an input with pull-down enabled
- TMS pad is an input with pull-up enabled
- $\overline{\text{RESET}}$ initially driven by the CRG, but switched to an input after the initial power-on reset setup by the CRG. This can be thought of as an open-drain RESET pin.

- All peripherals with a Module Enable bit are disabled.

6.9 Resets Differences from MAC71xx

- Added a software reset
- Consolidated all reset source registers into the CRG
- Additional POR time due to Flash POR handshaking
- Added an internal weak pull-down on the RESET pad
- Overall startup time of the device has changed

Chapter 7
Exceptions

7.1 Introduction

The ARM7 core on the MAC72xx supports the following exceptions:

Table 7-1. ARM Exception Table

Vector Address	Exception	Mode on entry	I State on entry	F State on entry	Priority (1=highest)
$0000 0000	Reset	Supervisor	Disabled	Disabled	1
$0000 0004	Undefined Instruction	Undefined	I	F	6
$0000 0008	Software Interrupt (SWI)	Supervisor	Disabled	F	7
$0000 000C	Abort (Prefetch)	Abort	I	F	5
$0000 0010	Abort (Data)	Abort	I	F	2
$0000 0018	Reserved	-	-	-	-
$0000 0018	IRQ	IRQ	Disabled	F	4
$0000 001C	FIQ	FIRQ	Disabled	Disabled	3

The ARM7 core does not support a relocatable exception table. Therefore, relocation of exception vectors is done by relocating the base address of a particular system resource. This is done primarily through the programming of the AXBS.

7.2 Exception Handling

This section lists any special features that have been added to the MAC72xx to better support the standard ARM7 exceptions. It does not list the features of the Interrupt Controller, except where those features have an impact on exception handling in the system.

Besides the requirements below for each exception type, there is a general requirement that the entire exception table be relocated. This requirement comes from the Primary Boot Loader concept described in Section 9.18, “System Boot Sequence”. In summary, the exception table must be relocatable into a non-FLASH, writable space during the boot process. One way to implement this is to implement an interrupt controller that returns an address for each exception. However, the MAC72xx interrupt controller only returns a vector number (or index), which can be used as an address offset/multiplier. This scheme is of limited use in the Primary/Secondary Boot Loader, where the final exception vector address should be programmable by the Secondary Boot Loader. Another alternative is to provide relocation of the exception table via a software configurable bit in the core itself (like the ColdFire architecture). However, the ARM7TDMI-S does not support this. The final alternative is to fix the exception table at a particular address (both in the core and in the Primary Boot Loader), and then to re-map particular memories to this

fixed address. For purposes of explanation in this document, that address will be assumed to be $0000_0000.

7.2.1 Reset

Address: $0000 0000

As discussed in Chapter 9, “Device Memory Map”, the physical memory or bus to which this address is mapped can be re-located both in hardware and in software.

7.2.2 Undefined Instruction

Address: $0000 0004

There are no additional co-processors in the system.

7.2.3 Software Interrupt

Address: $0000 0008

There is no special functionality on the MAC72xx to support SWIs.

7.2.4 Prefetch (Instruction) Abort

Address: $0000 000C

A special set of registers has been added to the MCM to support easier debugging of both Instruction and Data Aborts. These “core fault” registers will store the *last* address and transfer modifiers (size, type, etc.) that caused an Instruction or Data Abort (i.e.-the last address that caused a Transfer Error on the ARM7 native bus). The associated read/write data will not be stored.

On the MAC72xx, there are several sources of aborts that can be generated throughout the bus hierarchy, including the following:

- AXBS
  - Access to an unavailable slave port on the AXBS (S2, S4 and S6 on the MAC72xx). See Section 15.7.2, “Master/Slave Interface” for more details.
  - Writing to an AXBS slave address when the RO bit in the slave’s SGPCR register is set. See Section 15.7.2, “Master/Slave Interface” for more details.
- FlexBus
  - Access to the FlexBus hits no chip-select address ranges
- Flash
  - Access to Flash blocked by the WACC/SACC/DACC access protection in the Flash Controller and MCM
  - Flash access produces an ECC error
  - Flash access produces a Read-while-Write error
- SRAM

  - SRAM access produces an ECC error
  - Any SRAM access from $5000 to $7FFF (mirrored on a 32K boundary) — only on MAC72x2.
- AIPS (See Section 9.9, “Peripheral Bus Memory Map” for a list of PACR/OPACR registers)
  - User mode access to any AIPS slave with the corresponding SP bit set in the PACR/OPACR register.
  - Write to any AIPS slave with the corresponding WP bit set in the PACR/OPACR register.
  - Any access from an untrusted master to any AIPS slave with the corresponding TP bit set in the PACR/OPACR register.
  - Any access to an AIPS slave that is not implemented (i.e.-marked as “Reserved” in the documentation)
- Access to a Peripheral Bus peripheral is aborted by the peripheral itself.
  - Access to reserved addresses
  - Access to locked registers (See AXBS registers for an example)
  - Access to Supervisor only register when in User mode

The list is similar for the DMA, although the reporting mechanism is different (i.e.-no fault information is available, only an error is reported). However, on the DMA, the contents of the TCD may be examined to determine the cause of an aborted transfer.

Note that Instructions Aborts are only taken when the instruction enters the execution phase of the ARM7 pipeline.

In addition to flagging aborted instruction fetches, this exception is also used to implement the Non Maskable Interrupt (NMI). Please refer to Section 7.3.4, “Non-Maskable Interrupt (NMI) for more details. This means that the Instruction Abort ISR should first check to see if an NMI is pending before attempting to service an aborted instruction fetch.

7.2.5 Data Abort

Address: $0000 0010

See Section 7.2.4, “Prefetch (Instruction) Abort” for details of special features. The Data Abort exception is not used for the implementation of the NMI, and behaves as normal.

7.2.6 IRQ

Address: $0000 0018

7.2.7 FIQ

Address: $0000 001C

7.3 Interrupts

The MAC72xx supports up to 64 programmable interrupt sources, with Request $00 having the highest priority. Table 7-2 lists all 64 interrupts sources. Note that unused interrupt sources are reserved for future use, and are not available to the user.

Table 7-2. Interrupt Sources

Signal	Interrupt Request[a]	Interrupt Source	Priority (1=highest)	Includes
ipi_int[0]	$00	DMA	64	ch0 - DMA Channel #0
ipi_int[1]	$01	DMA	63	ch1 - DMA Channel #1
ipi_int[2]	$02	DMA	62	ch2 - DMA Channel #2
ipi_int[3]	$03	DMA	61	ch3 - DMA Channel #3
ipi_int[4]	$04	DMA	60	ch4 - DMA Channel #4
ipi_int[5]	$05	DMA	59	ch5 - DMA Channel #5
ipi_int[6]	$06	DMA	58	ch6 - DMA Channel #6
ipi_int[7]	$07	DMA	57	ch7 - DMA Channel #7
ipi_int[8]	$08	DMA	56	ch8 - DMA Channel #8
ipi_int[9]	$09	DMA	55	ch9 - DMA Channel #9
ipi_int[10]	$0A	DMA	54	ch10 - DMA Channel #10
ipi_int[11]	$0B	DMA	53	ch11 - DMA Channel #11
ipi_int[12]	$0C	DMA	52	ch12 - DMA Channel #12
ipi_int[13]	$0D	DMA	51	ch13 - DMA Channel #13
ipi_int[14]	$0E	DMA	50	ch14 - DMA Channel #14
ipi_int[15]	$0F	DMA	49	ch15 - DMA Channel #15
ipi_int[16]	$10	DMA	48	err - DMA Channel Execution Error
ipi_int[17]	$11	MCM	47	swt - Software Watchdog Timer
ipi_int[18]	$12	CRG	46	lock - PLL locked scm - Self Clock Mode entered
ipi_int[19]	$13	PIT	45	timer1 - Timer Channel 1
ipi_int[20]	$14	PIT	44	timer2 - Timer Channel 1
ipi_int[21]	$15	PIT	43	timer3 - Timer Channel 1
ipi_int[22]	$16	PIT	42	timer4 - Timer Channel 1 rti - Real Time Interrupt
ipi_int[23]	$17	MCM	41	ECC Error
ipi_int[24]	$18	CAN_A	40	mb0 to mb13 - Message Buffer mb15 to mb31 - Message Buffer
ipi_int[25]	$19	CAN_A	39	mb14 - Message Buffer

Table 7-2. Interrupt Sources

Signal	Interrupt Request[a]	Interrupt Source	Priority (1=highest)	Includes
ipi_int[26]	$1A	CAN_A	38	busoff - FlexCAN has entered the “Bus Off” state error - Any of a number of FlexCAN errors wakein - Asserted when the FlexCAN wakes up txwarn/rxwarn - Repetitive errors detected
ipi_int[27]	$1B	CAN_B	37	mb0 to mb13 - Message Buffer mb15 to mb31 - Message Buffer
ipi_int[28]	$1C	CAN_B	36	mb14 - Message Buffer
ipi_int[29]	$1D	CAN_B	35	busoff - FlexCAN has entered the “Bus Off” state error - Any of a number of FlexCAN errors wakein - Asserted when the FlexCAN wakes up txwarn/rxwarn - Repetitive errors detected
ipi_int[30]	$1E	Unused	34	
ipi_int[31]	$1F	Unused	33	
ipi_int[32]	$20	Unused	32	
ipi_int[33]	$21	Unused	31	
ipi_int[34]	$22	Unused	30	
ipi_int[35]	$23	Unused	29	
ipi_int[36]	$24	IIC	28	ibal - Arbitration Lost condition tcf - Byte Transfer condition iaas - Address Detect condition
ipi_int[37]	$25	SPI_A	27	eoqf - End of Queue tfff - TX FIFO Fill tcf - Transfer Complete tfuf - TX FIFO Underflow rfdf - RX FIFO Drain rfof - RX FIFO Overflow overrun
ipi_int[38]	$26	SPI_B	26	eoqf - End of Queue tfff - TX FIFO Fill tcf - Transfer Complete tfuf - TX FIFO Underflow rfdf - RX FIFO Drain rfof - RX FIFO Overflow overrun

Table 7-2. Interrupt Sources

Signal	Interrupt Request[a]	Interrupt Source	Priority (1=highest)	Includes
ipi_int[39]	$27	SCI_A	25	TDRE - Byte transferred to the transmit shift register TC - Transmit complete RDRF - Received data available in the SCI data register IDLE - Receiver input has become idle OR - Overrun condition NF - Noise has been detected FE - Frame error PF - Parity error BERR - Bit Error (only valid in LIN mode) RXRDY - LIN hardware has received a data byte TXRDY - LIN hardware can accept a control or data byte LWAKE - A Wakeup Character has been received from a LIN frame STO - Slave TimeOut PBERR - Physical Bus Error detected CERR - CRC Error detected CKERR - Checksum Error detected FRC - LIN Frame completed OVFL - LINRX Register Overflow
ipi_int[40]	$28	SCI_B	24	TDRE - Byte transferred to the transmit shift register TC - Transmit complete RDRF - Received data available in the SCI data register IDLE - Receiver input has become idle OR - Overrun condition NF - Noise has been detected FE - Frame error PF - Parity error BERR - Bit Error (only valid in LIN mode) RXRDY - LIN hardware has received a data byte TXRDY - LIN hardware can accept a control or data byte LWAKE - A Wakeup Character has been received from a LIN frame STO - Slave TimeOut PBERR - Physical Bus Error detected CERR - CRC Error detected CKERR - Checksum Error detected FRC - LIN Frame completed OVFL - LINRX Register Overflow
ipi_int[41]	$29	Unused	23	
ipi_int[42]	$2A	Unused	22	
ipi_int[43]	$2B	eMIOS	21	ch0 - eMIOS Channel 0
ipi_int[44]	$2C	eMIOS	20	ch0 - eMIOS Channel 1
ipi_int[45]	$2D	eMIOS	19	ch0 - eMIOS Channel 2
ipi_int[46]	$2E	eMIOS	18	ch0 - eMIOS Channel 3
ipi_int[47]	$2F	eMIOS	17	ch0 - eMIOS Channel 4
ipi_int[48]	$30	eMIOS	16	ch0 - eMIOS Channel 5

Table 7-2. Interrupt Sources

Signal	Interrupt Request[a]	Interrupt Source	Priority (1=highest)	Includes
ipi_int[49]	$31	eMIOS	15	ch0 - eMIOS Channel 6
ipi_int[50]	$32	eMIOS	14	ch0 - eMIOS Channel 7
ipi_int[51]	$33	Unused	13	ch0 - eMIOS Channel 8 (not on MAC72x2)
ipi_int[52]	$34	Unused	12	ch0 - eMIOS Channel 9 (not on MAC72x2)
ipi_int[53]	$35	Unused	11	ch0 - eMIOS Channel 10 (not on MAC72x2)
ipi_int[54]	$36	Unused	10	ch0 - eMIOS Channel 11 (not on MAC72x2)
ipi_int[55]	$37	Unused	9	ch0 - eMIOS Channel 12 (not on MAC72x2)
ipi_int[56]	$38	Unused	8	ch0 - eMIOS Channel 13 (not on MAC72x2)
ipi_int[57]	$39	Unused	7	ch0 - eMIOS Channel 14 (not on MAC72x2)
ipi_int[58]	$3A	Unused	6	ch0 - eMIOS Channel 15 (not on MAC72x2)
ipi_int[59]	$3B	ATD_A	5	cqnf - Command Queue Not Full cqf - Command Queue Full cqe - Command Queue Empty crl - Conversion Result Lost eto - external Trigger Overrun cp - Conversion Paused cc - Conversion Complete
ipi_int[60]	$3C	SPI_C	4	eoqf - End of Queue tfff - TX FIFO Fill tcf - Transfer Complete tfuf - TX FIFO Underflow rfdf - RX FIFO Drain rfof - RX FIFO Overflow overrun
ipi_int[61]	$3D	PIM	3	ipi_int_pim[0] - Port A ipi_int_pim[1] - Port B ipi_int_pim[2] - Port C ipi_int_pim[3] - Port D ipi_int_pim[4] - Port E ipi_int_pim[5] - Port F ipi_int_pim[6] - Port G
ipi_int[62]	$3E	IRQ (PD4)	2	irq - IRQ pin asserted
ipi_int[63]	$3F	XIRQ (PD3)	1	xirq - XIRQ pin asserted

a. The Interrupt ‘Request is not the same as the Interrupt Vector Number as returned by the Interrupt Controller. The relationship is defined in Equation 7-1.

$$\text{Vector Number} = \text{Interrupt Request} + 0x40 \qquad \textit{Eqn. 7-1}$$

7.3.1 Interrupt Clearing

By definition, all interrupt flags that can be cleared by accessing an interrupt status bit should be cleared by writing ‘1’. Clear on read, or clear on write 0 are not allowed on the MAC72xx device. All interrupts sources have been checked, and meet this requirement. The obvious exception are the external interrupts XIRQ, IRQ and NMI.

7.3.2 XIRQ and IRQ

Please also see Section 7.3.4, “Non-Maskable Interrupt (NMI)” for details on the NMI, and the behaviour of the XIRQ interrupt pin when in NMI mode.

7.3.3 PIT RTI and Timer 4

The PIT Timer 4 and RTI interrupts are OR’d together in order to allow maximum flexibility in PIT interrupt generation. Combined with the fact that the Timer 4 and RTI interrupts can be masked independently, this enables the following use cases:

- Mask RTI and unmask Timer 4: Timer 4 will generate the interrupt
- Mask Timer 4 and unmask RTI: RTI will generate the interrupt
- Unmask both: Timer 4 or RTI will generate the interrupt
- Mask both: no interrupts generated

7.3.4 Non-Maskable Interrupt (NMI)

The actual implementation of the NMI on the MAC72xx is done almost completely within the MCM module. The NMI has the following features:

- Write-once enabling of NMI mode in order to improve fault tolerance
- Write-once selectable NMI polarity in order to improve fault tolerance
- NMI Pending status

Because the IRQ and FIRQ interrupts may be masked by software in the ARM7 core, and modification to the ARM7 core was not deemed acceptable, an alternative method for implementing the NMI was conceived. The ARM7 core exceptions look like:

Table 7-3. ARM7 Core Exception

Priority		Exception	Software Maskable ?
Highest	1	RESET	No
	2	Data Abort	No
	3	FIRQ	Yes
	4	IRQ	Yes
	5	Pre-fetch Abort	No

Table 7-3. ARM7 Core Exception (Continued)

Priority		Exception	Software Maskable ?
	6	Undefined Instruction	No
Lowest	7	SWI	Yes

There are several requirements for the NMI on the MAC72xx:

- NMI is re-entrant
- Minimize NMI latency
- Automatic disabling of IRQ/FIRQ during NMI handling with the possibility of explicit software re-enable
- NMI has priority in the case of simultaneous NMI/Pre-fetch abort

The Data Abort exception has a higher priority in the core than the Instruction Abort, but since the NMI must be associated with a bus access, the interrupt latency can be improved by associating it with instruction fetches, which are far more frequent. This means that a normal Data Abort will have higher priority than an NMI, which may not be desirable.

To implement a "truer" NMI, the abort could be forced, regardless of the access type (instruction or data). This would place more burden on the customer, in that they would have to check for NMIs in both the Instruction and Data Abort exception routines, but this would give us a statistically better latency at least.

However, even with this scheme, there is no guaranteed latency. Due to the nature of the ARM instruction set (i.e.-almost all instructions can be made conditional), a handshaking scheme should be implemented that tags every bus access with an NMI-caused abort until the user services the NMI. It may be tricky to avoid re-entrancy of the Data Abort routine with this scheme.

The sequence of events in the system looks like the following:

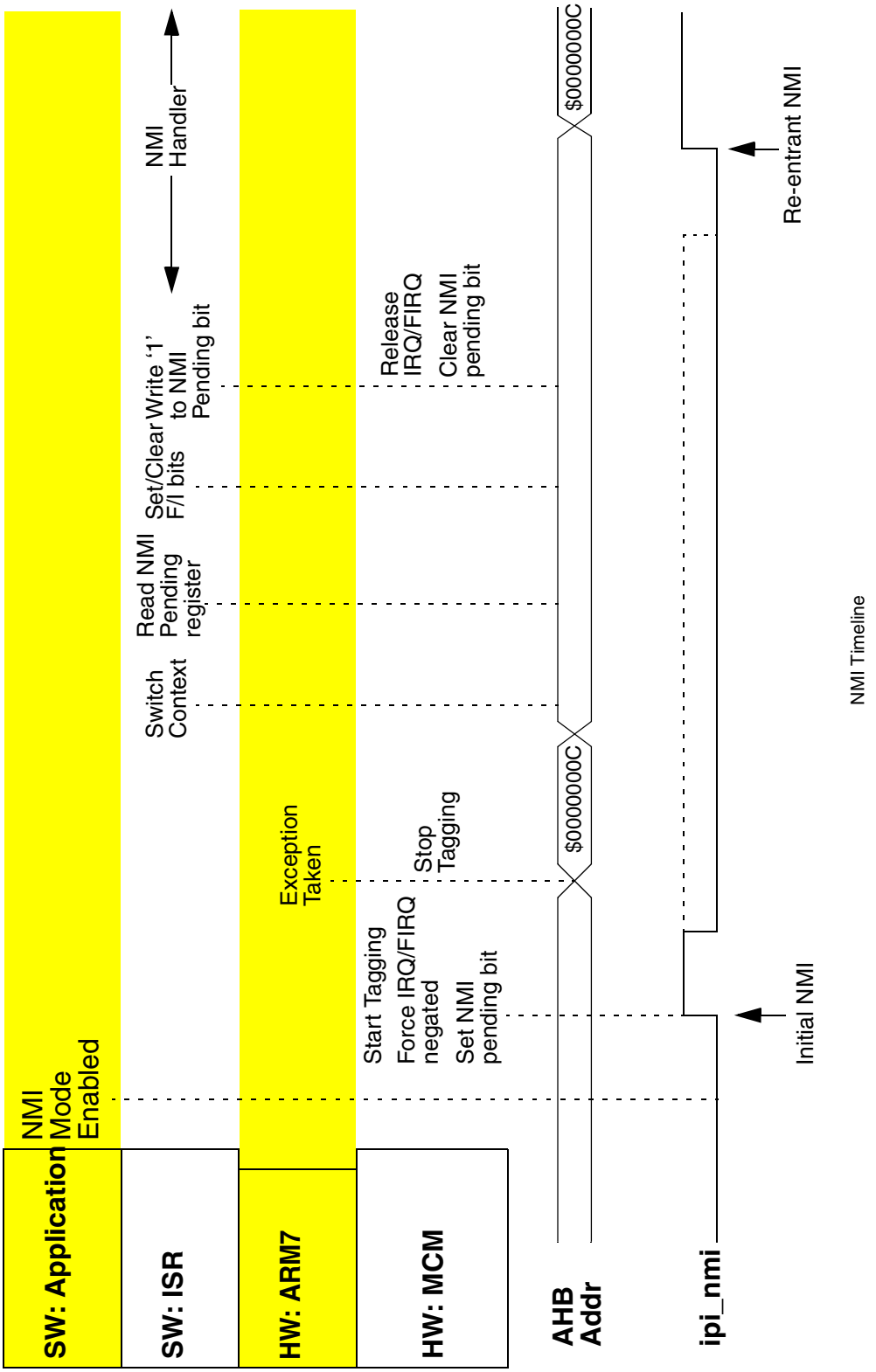
1. System resets in non-NMI mode
2. Software puts the SoC put into NMI mode and sets NMI polarity by writing a write-once register in the MCM. Until this happens the NMI interrupts are disabled, regardless of the value of the **ipi_nmi** input.
3. Once NMI mode is entered, the SoC forces proper sensitization of the path from **ipp_ind** (pad PD3) to the **ipp_nmi** input on the platform.
4. When a change from passive to active is seen on **ipi_nmi** (i.e.-edge sensitive), the MCM performs the following actions:
   - Set the NMI Pending bit in the MCM
   - Drive the FIRQ/IRQ inputs to the core negated
   - Begin tagging all instruction fetches as aborted
5. When one of the tagged instructions enters the execute phase in the ARM7 pipeline, a Pre-fetch abort exception is taken (assuming no higher exception occurs at the same time).
6. Upon seeing the instruction fetch to $0000_000C (Pre-fetch abort), the MCM stops tagging the instructions.
7. The ISR saves off all registers (i.e.-switch context, PC, PSR, etc.)

8. The ISR reads the NMI pending bit to determine if the exception was due to an NMI or to a real Pre-fetch abort. This provides the simplest (i.e.-lowest latency) NMI recognition, as well as providing for the case where a simultaneous NMI and Pre-fetch abort occur. In this case, the NMI should be serviced first (because the ISR sees a pending NMI). Once the NMI ISR is completed, the ARM core will try to re-fetch the "offending" instruction, so any "real" Pre-fetch aborts will be taken again.
9. Once the NMI branch of the ISR is taken, the ISR can set or clear the F and I bits in the CPSR to determine if a FIRQ/IRQ exception should be allowed during execution of the NMI handler. Remember, during this time, the FIRQ/IRQ signals to the ARM core are being forced negated by the MCM
10. Next, the ISR should write ’1’ to the NMI Pending bit. This will cause the MCM to do 2 things:
    — Stop forcing the FIRQ/IRQ negated, thereby enabling/disabling the FIRQ/ IRQ based solely on the F/I bits in the CPSR
    — Enable another NMI (i.e.-re-entrant NMI). This is automatic, since the NMI input is edge sensitive (i.e.-you must negate it by clearing the source before it can be asserted again).
11. Once the NMI input is negated, another active edge of the NMI will trigger a new NMI exception, *if* the NMI pending bit was written ‘1’ (i.e.-Step #10). If NMI re-entrancy is not desired, the NMI pending bit can be written at the end of the exception handler.

Since the Data Abort exception has higher priority than the Pre-fetch abort, the Data Abort exception handler may be entered while an NMI is pending. Since the Data Abort is not tagged for a pending NMI, and Data Aborts received are true Data Aborts. Upon entering the Data Abort exception, the first instruction fetch is tagged by the NMI hardware, and the Pre-fetch Abort exception handler will be called, thus guaranteeing higher NMI priority with a small latency penalty.

When the device is in NMI mode, pin PD[3] is used exclusively for the NMI. As such, the following special behavior is in effect:

- The XIRQ interrupt input to the platform is driven low (disabled)
- The port control signals from the PIM are ignored, so that the pad is always configured to be an input.


SW: Application
SW: ISR
HW: ARM7
HW: MCM
AHB Addr
ipi_nmi
NMI Mode Enabled
Switch Context
Read NMI Pending register
Set/Clear F/I bits
Write '1' to NMI Pending bit
NMI Handler
Exception Taken
Start Tagging
Force IRQ/FIRQ negated
Set NMI pending bit
Stop Tagging
Release IRQ/FIRQ
Clear NMI pending bit
$0000000C
$0000000C
Initial NMI
Re-entrant NMI
NMI Timeline

7.4 Exceptions Differences from the MAC71xx

- NMI functionality added, which uses the Prefetch Abort exception to signal an NMI
- Interrupt sources re-mapped. Please refer to Section 14.5, “INTC Differences from MAC71xx”

Chapter 8
Debug

8.1 Debug Introduction

The MAC7200 family of devices offers debugging with the ARM Core's EmbeddedICE and a NEXUS 3 interface. EmbeddedICE offers debug features such as setting Breakpoints or Watchpoints, modifying and reading memory contents. EmbeddedICE uses the standard JTAG serial interface and Test Access Port (TAP), and is compatible with existing ARM tool chains. The NEXUS interface provides real time program and data trace capability and also uses the JTAG port, as well as providing an auxiliary port. The auxiliary port can be provided in two pin positions depending on the device and package. Please refer to Chapter 11, "A7S Nexus3 Module" for more information on configuring and using the Nexus interface.

For details on the E-ICE interface please refer to Debugging Your System in the ARM7TDMI-S Users Manual, or ARM Application Note 31' Using Embedded'.

8.2 Debug Features

- Customer visible SC4 Debug Register with the following functionality:
  - JTAG Lockout Recovery
  - Device ID
  - Debug Reset
  - Chip Status
  - Core Run Control
- Please refer to Section 10.2, "ARM7 Features" for a list of debug related ARM7 features.
- Please refer to Section 11.1.2, "Nexus Feature List" for a list of debug related Nexus features.

8.3 Debug Protocol

The JTAG interface and TAP state machine follow the IEEE 1149.1-1990 protocol. Note that the MAC72xx does not support JTAG, but simply uses the JTAG protocol as a standard debug interface.

8.4 Debug Implementation

8.4.1 JTAG Interface

The JTAG interface on the MAC72xx is accessed using the following pins:

- **TCK** - Input Clock
- **TMS** - Input Mode Select

- **TDI** - Serial Data In
- **TDO** - Serial Data Out

Through the JTAG interface, the following may be accessed (See also Table 11-8):

- ARM7 EICE registers
- Nexus registers
- Debug register (SC4)

Although there are several blocks of logic that can be accessed through the JTAG interface, the MAC72xx contains only a single TAP state machine, located in the ARM7 complex. This TAP State Machine is used to select the various JTAG-accessible registers in the system. This is illustrated in Figure 8-1.

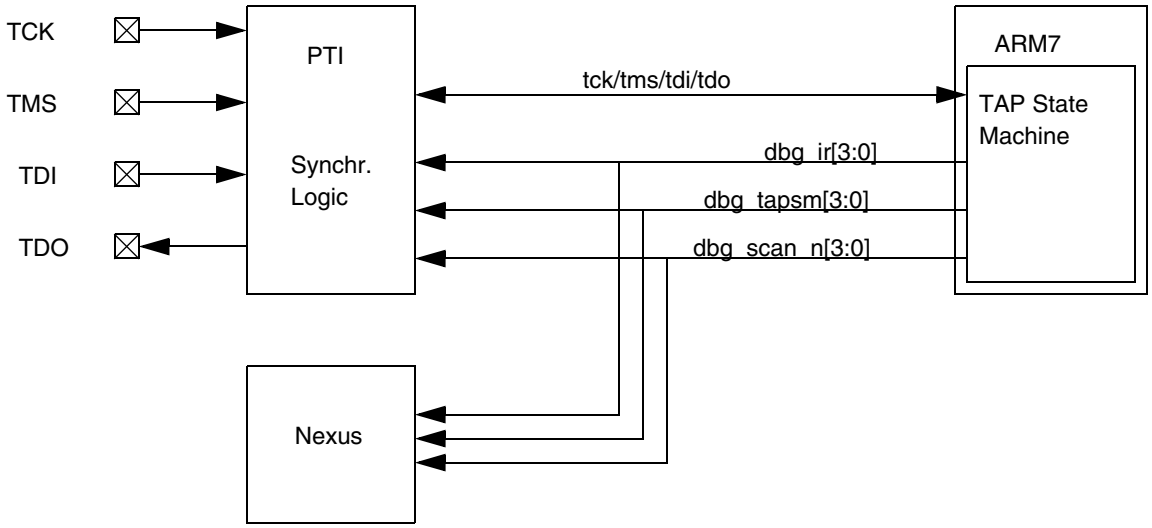


Figure 8-1. JTAG Interface Overview

Selection between the different pieces of Test Logic on the MAC72xx is done by decoding the **dbg_ir[3:0]** and **dbg_scan_n[3:0]** signals from the TAP state machine. Table 8-1 shows the decoding for the MAC72xx. The **dbg_tapsm[3:0]** signal is the current state of the TAP state machine, and is used by the PTI and Nexus to determine which JTAG state the TAP is currently in.

Table 8-1. JTAG Test Logic Selection

dbg_ir[3:0]	dbg_scan_n[3:0]	Function
$2 (SCAN_N)	$4	ARM7 Scan Chain 0 Selected
$C (INTEST)	$4	Debug Register SC4 Selected
$8 (NEXUS_ACCESS)	n/a	Nexus Selected

Based on which test logic is selected, the TCK, TMS, TDI and TDO inputs/outputs are routed to the correct destination. The following figures illustrate this.

8.4.1.1 TCK Routing

The TCK signal from the pad is synchronized, and then drives the TCK inputs of the PTI logic, Nexus logic and the ARM7. See Section 8.4.2, “Synchronization” for more details on how TCKEN is used to synchronize TMS and TDI.

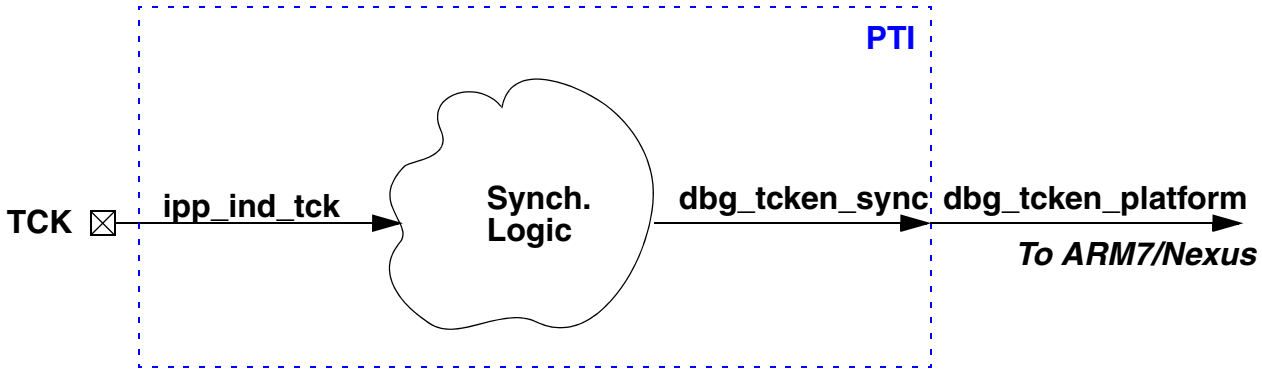


Figure 8-2. JTAG TCK Routing

8.4.1.2 TMS Routing

The TMS signal from the pad is synchronized, and then drives the TMS inputs of the PTI logic, Nexus logic and the ARM7. The TMS input of the SoC must be valid on the rising edge of the TCK clock, with the appropriate setup and hold requirements. If the TMS signal is asserted based on the falling clock edge of TCK, and the maximum TCK frequency is not exceeded, then the setup and hold requirements for TMS are guaranteed to be met.

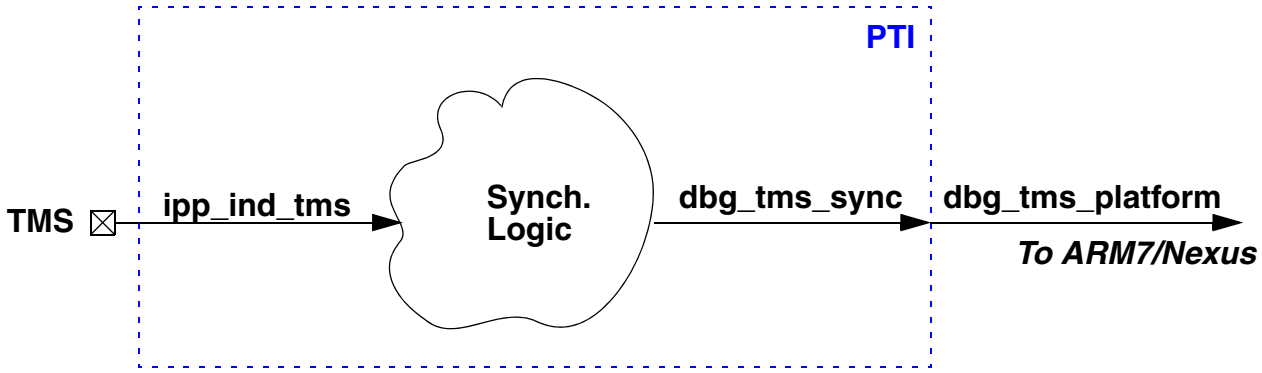


Figure 8-3. JTAG TMS Routing

8.4.1.3 TDI Routing

The TDI signal from the pad is synchronized, and then muxed in to drive the TDI inputs of the PTI logic, Nexus logic and the ARM7. The synchronized signal is muxed in order to provide additional security when the debug is disabled in the system for Flash security. When the debug is disabled, the INTEST instruction is selected, and the TAP state machine is in the Shift-DR state, the TDI input *to the platform* is tied low. The TDI input to the PTI logic (SC4/5/6/7) is always driven by **dbg_tdi_sync**, thus allowing the writing and reading of SC4/5/6/7 even when debug is disabled. This is to allow the user to run the JTAG Lockout

Recovery, which can be used to unsecure the device and thereby enable all debug features. The TDI input of the SoC must be valid on the rising edge of the TCK clock, with the appropriate setup and hold requirements. If the TDI signal is asserted based on the falling clock edge of TCK, and the maximum TCK frequency is not exceeded, then the setup and hold requirements for TDI are guaranteed to be met.

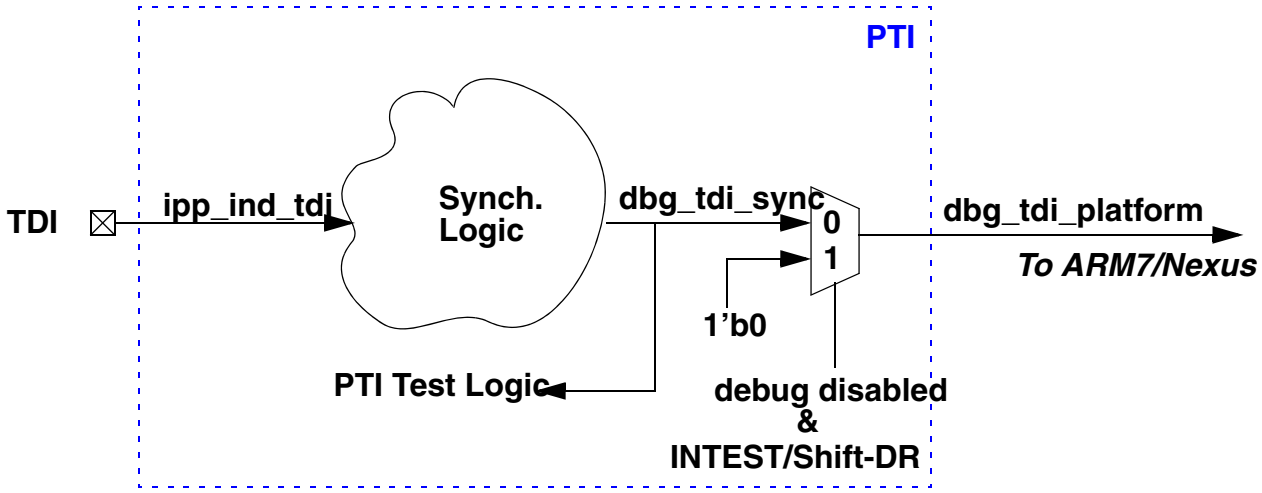


Figure 8-4. JTAG TDI Routing

8.4.1.4 TDO Routing

The TDO signal is driven by one of many test logic sources (See Table 8-1). If no test logic source is selected, the ARM7 will drive the TDO signal low.

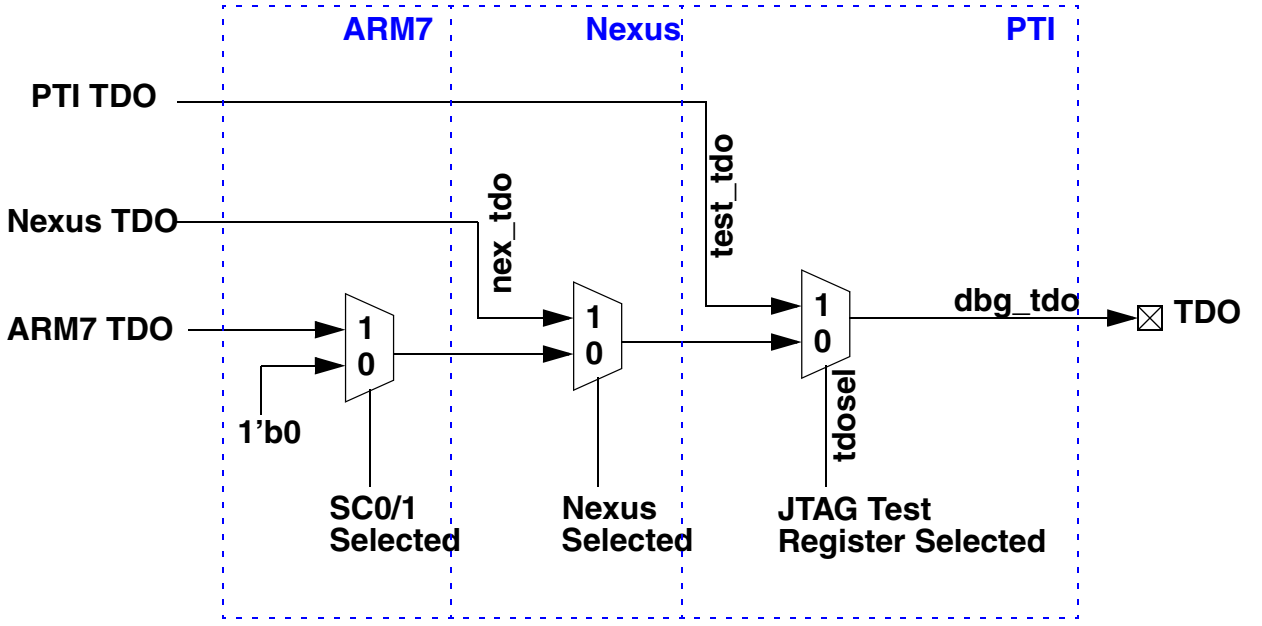


Figure 8-5. JTAG TDO Routing

8.4.2 Synchronization

As with any external input, it is necessary to synchronize the TCK, TMS and TDI inputs to the internal clock of the TAP state machine (**ipg_clk**). However, in the case of these signals, it is not necessary to synchronize all three signals, as TMS and TDI are driven externally with respect to TCK. Therefore, the synchronization can be performed by synchronizing the TCK input, and then generating an enable (**tap_enable**) signal from this synchronization logic that is used to strobe the TMS and TDI inputs. The actual logic is illustrated in Figure 8-6, and the timing is described in Figure 8-7.

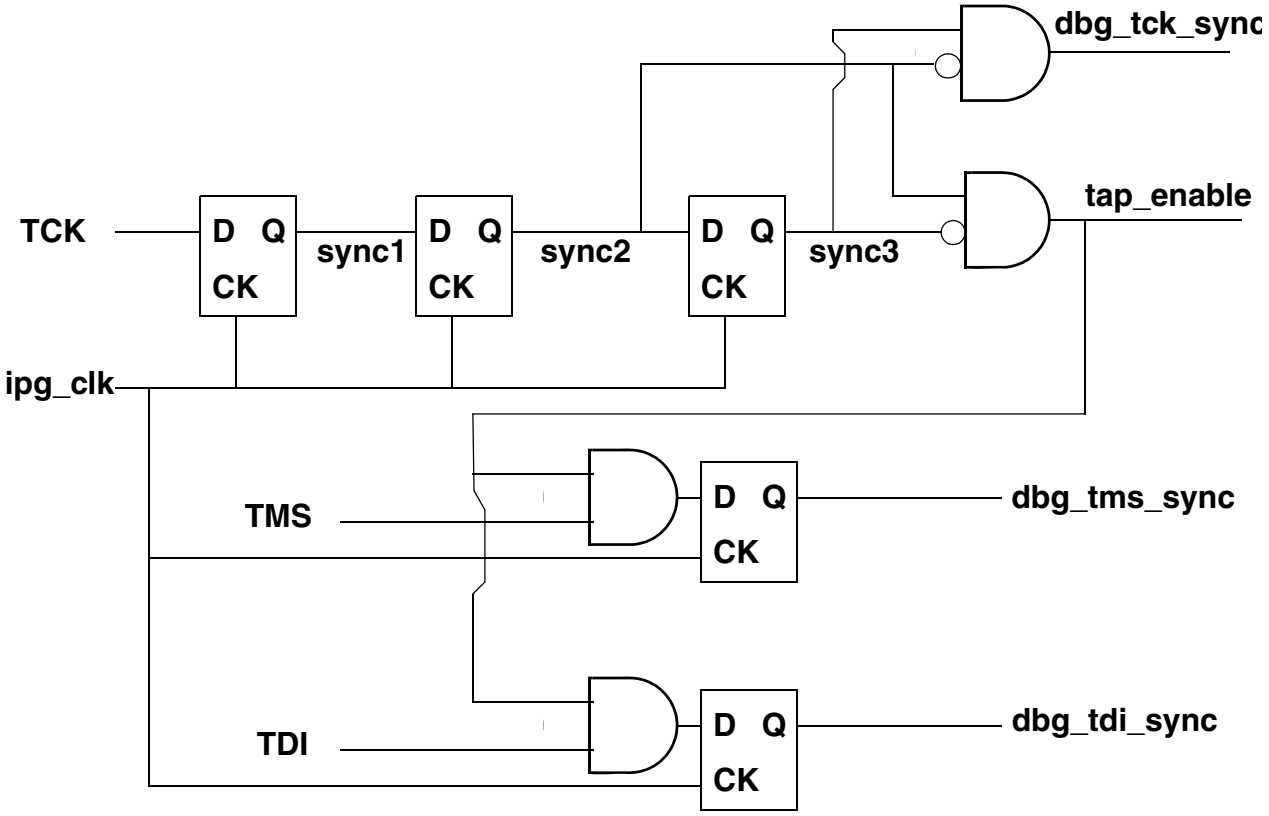


Figure 8-6. JTAG Synchronization Circuit

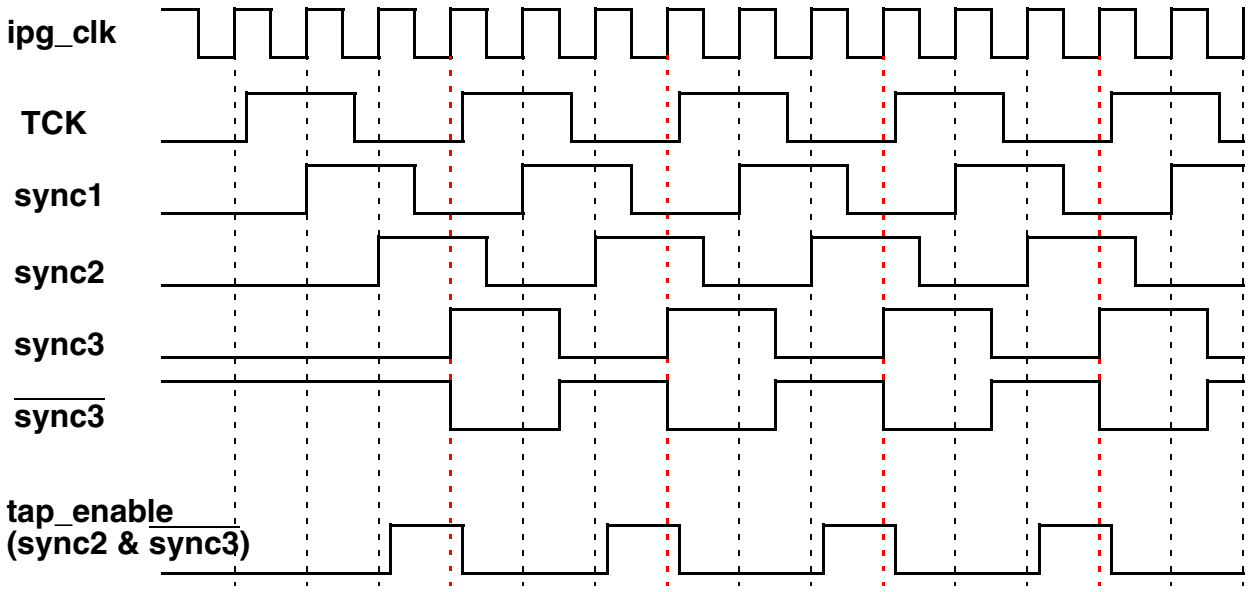


Figure 8-7. JTAG Synchronization Timing (TCK = 1/3 frequency of ipg_clk)

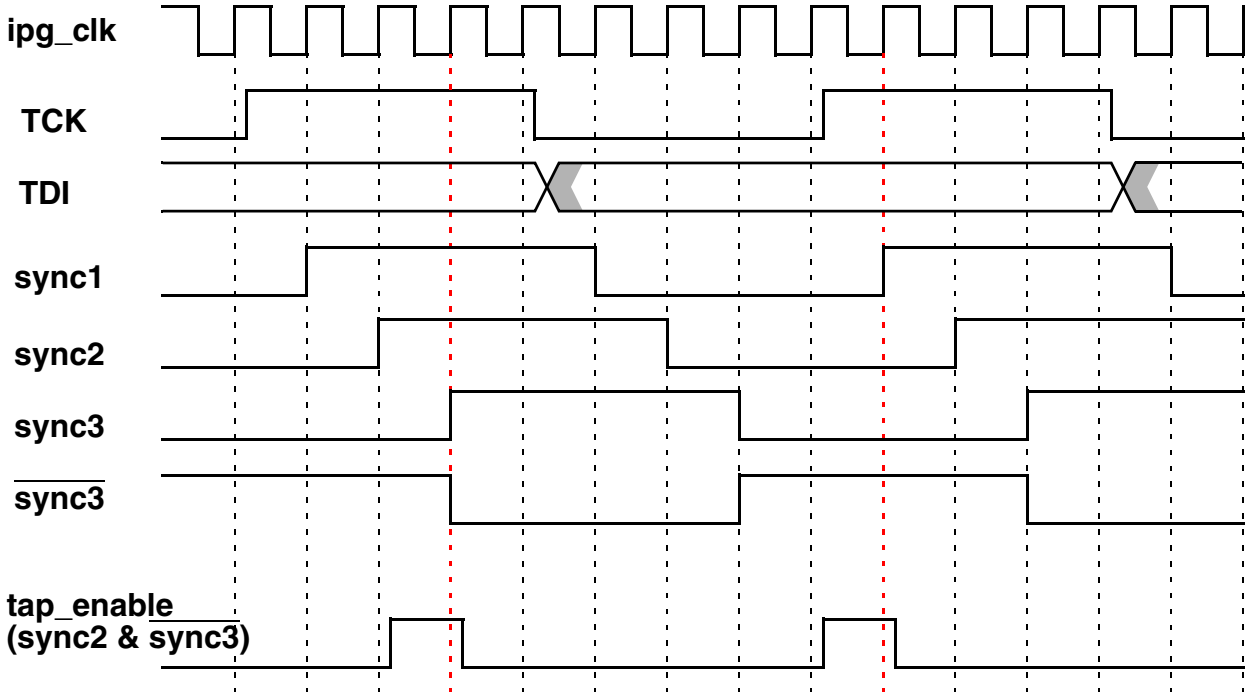


Figure 8-8. JTAG Synchronization Timing (TCK = 1/8 frequency of ipg_clk)

Even though Figure 8-7 illustrates the case where TCK is 1/3 the frequency of the system clock, the actual maximum operating speed of the TCK input is 1/8 the frequency of the system clock. This is done to account for differences in insertion delay between TCK, TMS and TDI without having to constrain the timing of these pins.

8.4.3 Debug Reset

See Section 6.4, “Debug Reset” for a complete description of debug logic resets.

8.5 Debug External Pins

Table 8-2. Debug External Pins

Pin	Description
TCK	JTAG Input Clock. This pin provides the test clock input to synchronize the device’s test logic. TCK is independent of the processor clock, and is used only as the reference clock for the TMS, TDI and TDO pins. During and after a system reset, this pin will be configured as an input with an internal pull-down active on this signal. This pull-down is implemented using active elements, and is approximately equivalent to a 20KOhm resistor under typical operating conditions.
TMS	JTAG Input Mode Select. This pin is used to input the test mode to sequence the TAP controller’s state machine. The pin is sampled on the rising edge of the TCK pin. During and after a system reset, this pin will be configured as an input with an internal pull-up active on this signal. This pull-up is implemented using active elements, and is approximately equivalent to a 20KOhm resistor under typical operating conditions.
TDI	JTAG Serial Data In. This serial pin is sampled in the device on the rising edge of the TCK pin. During and after a system reset, this pin will be configured as an input with an internal pull-up active on this signal. This pull-up is implemented using active elements, and is approximately equivalent to a 20KOhm resistor under typical operating conditions.
TDO	JTAG Serial Data Out. This pin is a three-state test data output pin that is actively driven in the shift-IR and shift-DR controller states. The pin state changes on the falling edge of the TCK pin. During and after a system reset, this pin will be configured as an output with maximum slew rate enabled.

8.6 Debug Bus Aborts

There is no bus interface to any of the debug logic.

8.7 Debug Differences from MAC71xx

See Section 11.7, “Nexus Differences from MAC71xx”.

8.8 Debug Application Usage

The Debug Mode of the system is completely determined by the Debug state of the ARM7 core. Therefore, please refer to the ARM7 documentation for all of the possible ways for the ARM7 core to enter the debug state.

8.8.1 ARM Debug Overview

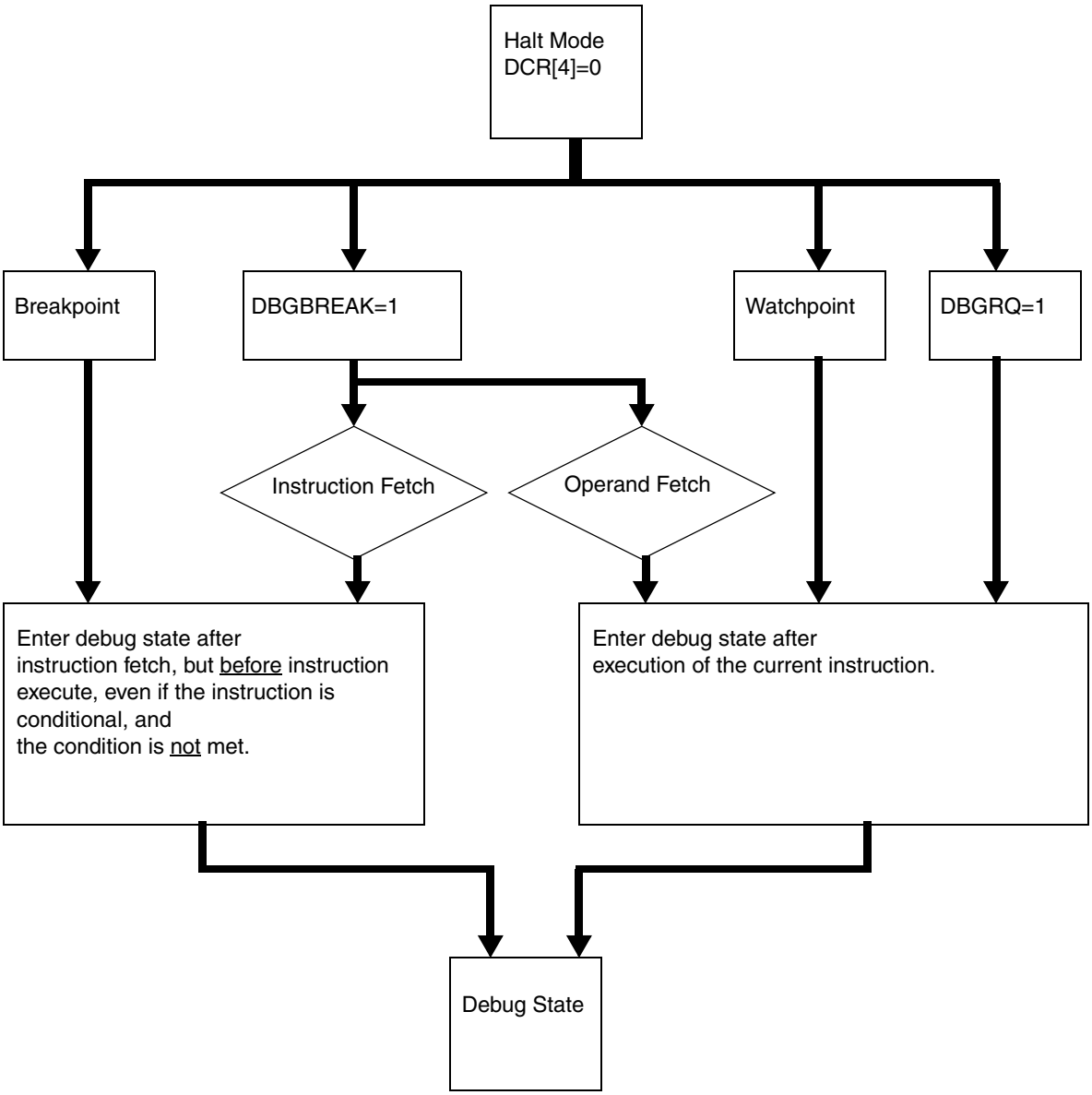


Figure 8-9. Halt Mode Overview

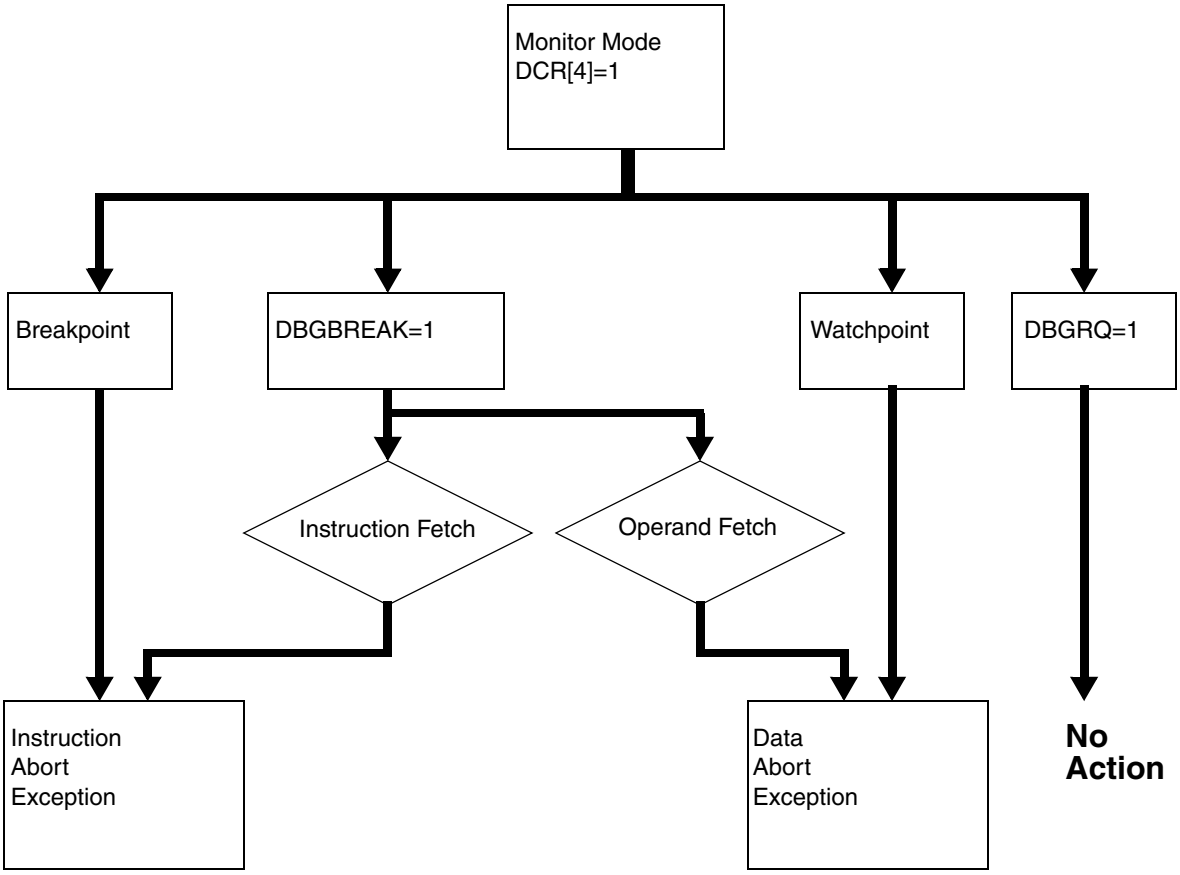


Figure 8-10. Monitor Mode Overview

Table 8-3. ARM7TDMI-S Debug State Overview

	Debug State	Monitor Mode
Processor	External bus indicates “internal cycles”	Processor executes normally
Interrupts	Disabled	Enabled
Aborts	Disabled	Enabled. Instruction/Data Abort also triggered by Breakpoint/Watchpoint, respectively.
JTAG Interface	Active Can insert instructions into the ARM instruction pipeline to examine registers and system state	Active Can communicate with a software monitor executing on the core

8.8.2 Entering Debug mode

- The ARM7 core is forced into a debug state

- The ARM7 core is placed into Halt Mode or Monitor Mode, depending on the state of the **DCR[4]** bit.
- The ARM7 core asserts the **DBGACK** output
- The CRG asserts the **ipg_debug** signal
- The Real Time Interrupt (RTI) clock **rti_osc_ps_clk** is shut off by the CRG
- The Software Watchdog Timer (SWT) clock **swt_osc_ps_clk** is shut off by the CRG
- Debug Mode is optional for all peripherals except the SWT and RTI
- All peripheral level debug features are enabled (Please refer to the individual chapters for a complete description of functionality in Debug Mode).
  - DMA2
  - FlexCAN2
  - DSPI
  - eMIOS
  - ATD
  - MCM (Software Watchdog Timer)

8.8.3 Exiting Debug mode

- The ARM7 core exits the debug state
- The ARM7 core exits Halt/Monitor Mode
- The ARM7 core negates the **DBGACK** output
- The CRG negates the **ipg_debug** signal
- The Real Time Interrupt (RTI) clock **rti_osc_ps_clk** is turned on by the CRG
- The Software Watchdog Timer (SWT) clock **swt_osc_ps_clk** is turned on by the CRG
- All peripheral level debug features are disabled (Please refer to the individual chapters for a complete description of functionality in Debug Mode).
  - DMA2
  - FlexCAN2
  - DSPI
  - eMIOS
  - ATD
  - MCM (Software Watchdog Timer)

8.8.4 Nexus Low Power State

Both the Debug and Doze mode states may be determined by reading out the Nexus LPS bits. Please refer to Table 11-9 for the exact encodings.

8.8.5 Debug Shift Register SC4

Please refer to Section 35.1.1, “JTAG Test Register (SC4)”.

8.8.6 Using the JTAG Interface

Because the JTAG interface requires internal clocks for synchronization, it is generally not safe to assume that the interface is available all of the time. In particular, there are several situations where the JTAG interface may not be available:

- Immediately after a Power On Reset (POR)
- Immediately after a loss of clock condition

In general, under normal operating conditions, the JTAG interface may be used while the RESET is asserted for a normal system reset. The safest approach is to reset the device, program the debug registers, and then reset the device again. Since the debug logic is not reset by a normal system reset, this procedure is guaranteed to work.

In order to meet the IEEE 1149.1-1990 JTAG specification, it is required that the JTAG interface be run at no faster than 1/8 the system clock frequency. As an example, for an 80MHz system, the maximum TCK frequency is 10MHz. There may be further limitations introduced by the development system or PCB. If the debug system does not seem to be functioning correctly, try reducing the JTAG speed and resetting the system.

8.8.7 JTAG Pad Control

Control of the JTAG related pads is done purely from the PIM registers, as follows:

TCK

- Can configure pull-up or pull-down
- Can disable the TDI functionality. In this case, the pad becomes an output (driven low), and the internal TDI signal is driven low (0).

TMS

- Can configure pull-up or pull-down
- Can disable the TDI functionality. In this case, the pad becomes an output (driven low), and the internal TDI signal is driven low (0).

TDI

- Can configure pull-up or pull-down
- Can disable the TDI functionality. In this case, the pad becomes an output (driven low), and the internal TDI signal is driven low (0).

TDO

- Can configure slew rate
- Can disable the TDO functionality. In this case, the pad becomes an input (All changes on the pad are ignored by the system).

8.8.8 Resetting Debug Logic

Please refer to Section 6.4, “Debug Reset” for a complete description of debug logic resets.

Chapter 9
Device Memory Map

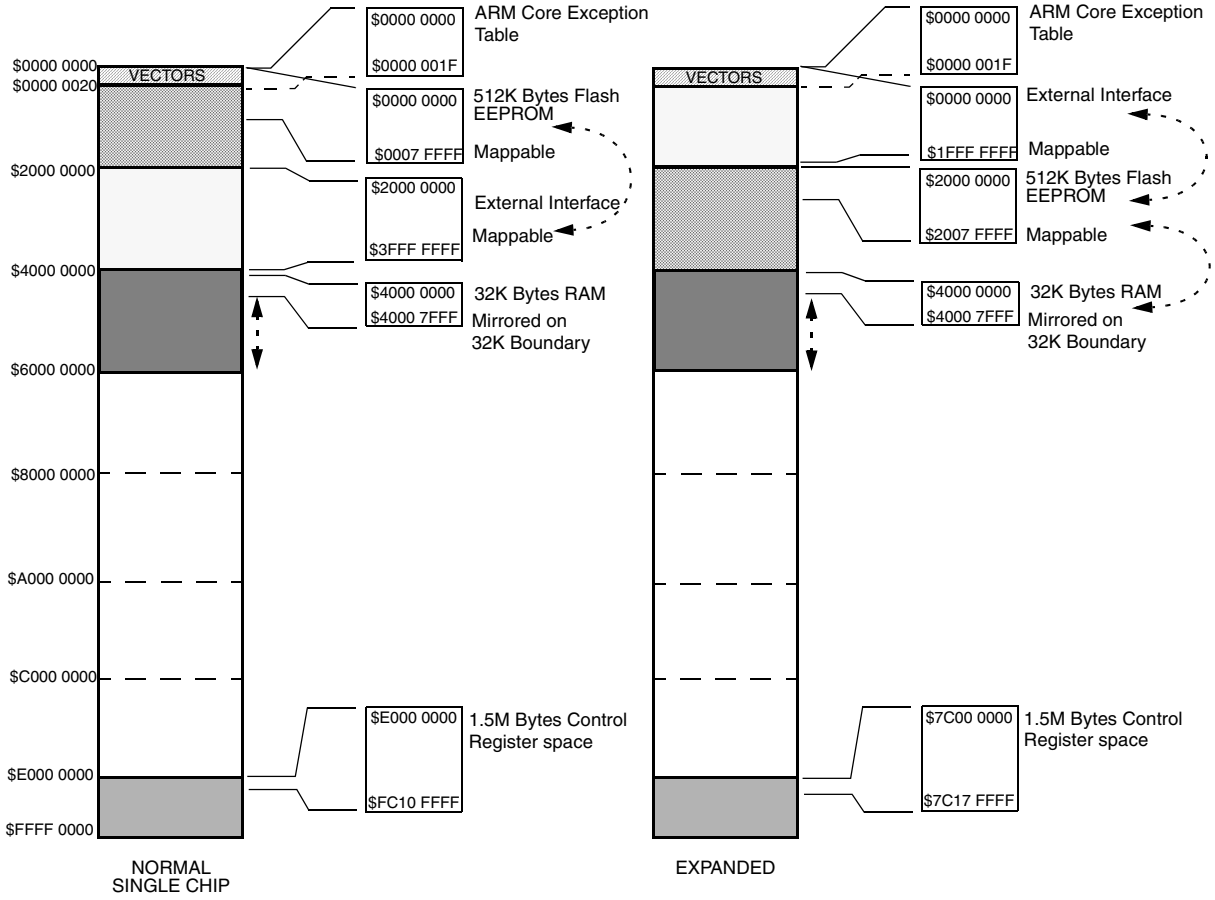
This section describes the memory map of the MAC72xx after reset, in the various possible chip modes. For more details on selecting chip modes, please refer to Chapter 2, “Modes of Operation”. The memory map of the device immediately following reset is different based on the mode of operation that has been selected, but is largely re-mappable following reset. It is possible to re-map the flash main array and shadow block, the SRAM and the external bus interface using the AAMR register in the MCM module, but not the ARM7 exception table, which is fixed at $0000 0000 to $0000 001F, Figure 9-1 shows the device memory map at reset in each of the various device modes.

	Normal Single Chip	Secured Single Chip	Normal Expanded	Secured Expanded	Normal Bootload	Secured Bootload	Lockout Recovery
$0000 0000	Flash Main Array	Flash Main Array	FlexBus	FlexBus	Shadow Block	Shadow Block	BAM
$00F0 0000	Shadow Block	Shadow Block					
$2000 0000	FlexBus		Flash Main Array		Flash Main Array	Flash Main Array	Flash Main Array
$20F0 0000			Shadow Block		Shadow Block	Shadow Block	Shadow Block
$4000 0000	SRAM	SRAM	SRAM	SRAM	SRAM	SRAM	SRAM
$6000 0000							
$8000 0000							
$A000 0000	BAM	BAM	BAM	BAM	BAM	BAM	BAM
$C000 0000							
$E000 0000	Peripheral Bus	Peripheral Bus	Peripheral Bus	Peripheral Bus	Peripheral Bus	Peripheral Bus	Peripheral Bus

Figure 9-1. MAC72xx Memory Map Overview

9.1 Memory Map Example

Figure 9-2. MAC72x1 Memory Map Example

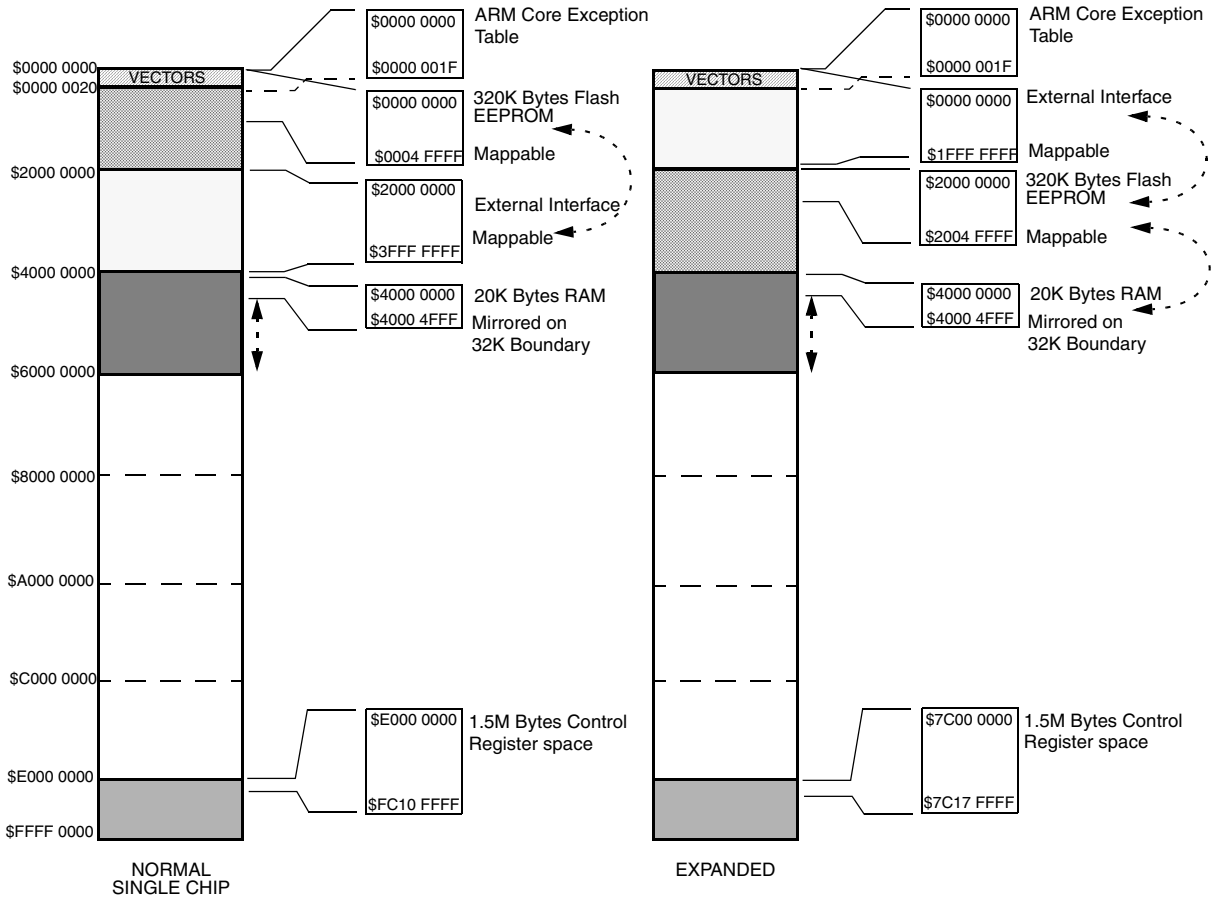


After reset the map is:

Normal Single Chip Mode	Normal Expanded Chip Mode
$0000 0000 - $0007 FFFF: 512K Flash	$2000 0000 - $2007 FFFF: 512K Flash
$00F0 0000 - $00F0 7FFF: 32K Shadow Block	$20F0 0000 - $20F0 7FFF: 32K Shadow Block
$2000 0000 - $3FFF FFFF: External Bus	$0000 0000 - $1FFF FFFF: External Bus
$4000 0000 - $4000 7FFF: 32K RAM	$4000 0000 - $4000 7FFF: 32K RAM
$FC00 0000 - $FC03 FFFF: Register Space	$FC00 0000 - $FC03 FFFF: Register Space

Flash, External Interface and RAM space can all be remapped in software on 512M byte boundaries after Reset. The ARM core exception table can not be remapped.

Figure 9-3. MAC72x2 Memory Map Example



After reset the map is:

Normal Single Chip Mode	Normal Expanded Chip Mode
$0000 0000 - $0004 FFFF: 320K Flash	$2000 0000 - $2004 FFFF: 320K Flash
$00F0 0000 - $00F0 7FFF: 32K Shadow Block	$20F0 0000 - $20F0 7FFF: 32K Shadow Block
$2000 0000 - $3FFF FFFF: External Bus	$0000 0000 - $1FFF FFFF: External Bus
$4000 0000 - $4000 4FFF: 20K RAM	$4000 0000 - $4000 4FFF: 20K RAM
$FC00 0000 - $FC03 FFFF: Register Space	$FC00 0000 - $FC03 FFFF: Register Space

Flash, External Interface and RAM space can all be remapped in software on 512M byte boundaries after Reset. The ARM core exception table can not be remapped.

9.2 Normal Single Chip Mode

Normal Single Chip Mode is typically used to debug and develop application code, where a software bootloader is not required. Booting is performed from the Flash main array, and all functionality is available. In this mode, the following functionality is available:

- Debug functionality (EICE, Nexus and the JTAG interface) is enabled

- External Bus Interface is enabled
- Flash main array access is enabled
- Shadow Block access is enabled
- Boot from the Flash main array

Table 9-1 shows the device memory map of the MAC72xx in Normal Single Chip Mode after Reset.

Table 9-1. Device Memory Map in Normal Single Chip Mode (After Reset)[1]

Address	Module	Size (Bytes)
$0000 0000 - $0007 FFFF $0000 0000 - $0004 FFFF	Flash main array[2] (MAC72x1) Flash main array[2] (MAC72x2)	512K 320K
$00F0 0000 - $00F0 7FFF	Shadow Block	32K
$0008 0000 - $1FFF FFFF	Reserved	~511M
$2000 0000 - $3FFF FFFF	External Bus Interface	512M
$4000 0000 - $4000 7FFF $4000 0000 - $4000 4FFF	SRAM[3] (MAC72x1) SRAM[3] (MAC72x2)	32K 20K
$4000 8000 - $FBFF FFFF	Reserved	~3008M
$FC00 0000 - $FFFF FFFF	Peripheral Bus modules	64M

1. This reset memory map equates to a reset value of $f0d0 0b98 in the AAMR register in the MCM module.
2. The Exception Vector Table is located at $0000 0000 - $0000 001F in the Flash main array.
3. The SRAM is mirrored across the entire 512MByte address range on a 32K boundary. Please refer to Section 9.10, “SRAM Memory Map”.

After Reset, the memory map may be re-mapped using the AAMR register in the MCM module. The allowed combinations are shown in Table 9-2. The Peripheral Bus space is always fixed at $FC00 0000, and is not re-mappable.

Table 9-2. Allowed Memory Maps in Normal Single Chip Mode

$0000 0000	$2000 0000	$4000 0000	AAMR Register
External Bus	Flash Main Array	SRAM	$f0d0 0b89
Unused	Flash Main Array	SRAM	$f0d0 0b80
SRAM	Flash Main Array	SRAM	$f0d0 0b8b
Flash Main Array	External Bus	SRAM	$f0d0 0b98
External Bus	External Bus	SRAM	$f0d0 0b99
Unused	External Bus	SRAM	$f0d0 0b90
SRAM	External Bus	SRAM	$f0d0 0b9b
Flash Main Array	Unused	SRAM	$f0d0 0b08
External Bus	Unused	SRAM	$f0d0 0b09
Unused	Unused	SRAM	$f0d0 0b00
SRAM	Unused	SRAM	$f0d0 0b0b
Flash Main Array	SRAM	SRAM	$f0d0 0bb8
External Bus	SRAM	SRAM	$f0d0 0bb9
Unused	SRAM	SRAM	$f0d0 0bb0
SRAM	SRAM	SRAM	$f0d0 0bbb

9.3 Normal Primary Bootloader Mode

Normal Primary Bootloader Mode is typically used to debug and develop application code, where a software bootloader is required. Booting is performed from the Shadow Block, and all functionality, except the External Bus, is available. In this mode, the following functionality is available:

- Debug functionality (EICE, Nexus and the JTAG interface) is enabled
- External Bus Interface is disabled
- Flash main array access is enabled
- Shadow Block access is enabled
- Boot from the Shadow Block

Table 9-3 shows the device memory map of the system in both Normal and Secured Primary Bootloader Mode after Reset. The memory map for this mode differs from the memory map in Normal Single Chip Mode as follows:

- The Shadow Block is mirrored to address $0000 0000. Note that the mirroring (rather than relocating) of the Shadow Block means that it is available both at address $0000 0000 and at address $20F0 0000.
- The Flash Main Array is relocated to address $2000 0000
- The External Bus Interface is not available

Table 9-3. Device Memory Map in Normal/Secured Primary Bootloader Mode (After Reset)[1]

Address	Module	Size (Bytes)
$0000 0000 - $0000 7FFF	Shadow Block[2]	32K
$0000 8000 - $1FFF FFFF	Reserved	~511M
$2000 0000 - $2007 FFFF $2000 0000 - $2004 FFFF	Flash Main Array (MAC72x1) Flash Main Array (MAC72x2)	512K 320K
$20F0 0000 - $20F0 7FFF	Shadow Block[2]	32K
$2008 0000 - $3FFF FFFF	Reserved	~511M
$4000 0000 - $4000 7FFF $4000 0000 - $4000 4FFF	SRAM[3] (MAC72x1) SRAM[3] (MAC72x2)	32K 20K
$4000 8000 - $FBFF FFFF	Reserved	~3008M
$FC00 0000 - $FFFF FFFF	Peripheral Bus modules	64M

1. This reset memory map equates to a reset value of $f0d0 0b80 in the AAMR register in the MCM module.
2. The Shadow Block is available at both $0000 0000 and $20F0 0000.
3. The SRAM is mirrored across the entire 512MByte address range on a 32K boundary. Therefore, the first word in SRAM may be read from or written to using address $4000 0000 or address $4000 8000 or address $4001 0000, etc.

After Reset, the memory map may be re-mapped using the AAMR register in the MCM module. The allowed combinations are shown in Table 9-4. The Peripheral Bus space is always fixed at $FC00 0000, and is not re-mappable.

Table 9-4. Allowed Memory Maps in Normal Primary Bootloader Mode

$0000 0000	$2000 0000	$4000 0000	AAMR Register
Unused	Flash Main Array	SRAM	$f0d0 0b80
SRAM	Flash Main Array	SRAM	$f0d0 0b8b
Flash Main Array	Unused	SRAM	$f0d0 0b08
Unused	Unused	SRAM	$f0d0 0b00
SRAM	Unused	SRAM	$f0d0 0b0b
Flash Main Array	SRAM	SRAM	$f0d0 0bb8
Unused	SRAM	SRAM	$f0d0 0bb0
SRAM	SRAM	SRAM	$f0d0 0bbb

9.4 Normal Expanded Mode

Normal Expanded Mode is typically used to debug and develop application code from an external memory. Booting is done from the external bus, and all functionality is available. The advantage of debugging code from an external memory (versus the internal Flash Main Array), is that software breakpoints may be inserted without the added burden of re-programming the Flash Main Array. In this mode, the following functionality is available:

- Debug functionality (EICE, Nexus and the JTAG interface) is enabled
- External Bus Interface is enabled
- Flash main array access is enabled
- Shadow Block access is enabled
- Boot from the External Bus

Table 9-5 shows the device memory map of the device in Normal Expanded Mode after Reset. The memory map for this mode differs from the memory map in Normal Single Chip Mode as follows:

- The External Bus Interface is relocated to address $0000 0000
- The Flash Main Array is relocated to address $2000 0000

Table 9-5. Device Memory Map in Normal Expanded Mode (After Reset)[1]

Address	Module	Size (Bytes)
$0000 0000 - $1FFF FFFF	External Bus Interface	512M
$2000 0000 - $2007 FFFF $2000 0000 - $2004 FFFF	Flash Main Array (MAC72x1) Flash Main Array (MAC72x2)	512K 320K
$20F0 0000 - $20F0 7FFF	Shadow Block	32K
$2008 0000 - $3FFF FFFF	Reserved	~511M
$4000 0000 - $4000 7FFF $4000 0000 - $4000 4FFF	SRAM[2] SRAM[2]	32K 20K
$4000 8000 - $FBFF FFFF	Reserved	3008M
$FC00 0000 - $FFFF FFFF	Peripheral Bus modules	64M

1. This reset memory map equates to a reset value of $f0d0 0b89 in the AAMR register in the MCM module.

2. The SRAM is mirrored across the entire 512MByte address range on a 32K boundary. Therefore, the first word in SRAM may be read from or written to using address $4000 0000 or address $4000 8000 or address $4001 0000, etc.

After Reset, the memory map may be re-mapped using the AAMR register in the MCM module. The allowed combinations are shown in Table 9-6. The Peripheral Bus space is always fixed at $FC00 0000, and is not re-mappable.

Table 9-6. Allowed Memory Maps in Normal Expanded Mode

$0000 0000	$2000 0000	$4000 0000	AAMR Register
External Bus	Flash Main Array	SRAM	$f0d0 0b89
Unused	Flash Main Array	SRAM	$f0d0 0b80
SRAM	Flash Main Array	SRAM	$f0d0 0b8b
Flash Main Array	External Bus	SRAM	$f0d0 0b98
External Bus	External Bus	SRAM	$f0d0 0b99
Unused	External Bus	SRAM	$f0d0 0b90
SRAM	External Bus	SRAM	$f0d0 0b9b
Flash Main Array	Unused	SRAM	$f0d0 0b08
External Bus	Unused	SRAM	$f0d0 0b09
Unused	Unused	SRAM	$f0d0 0b00
SRAM	Unused	SRAM	$f0d0 0b0b
Flash Main Array	SRAM	SRAM	$f0d0 0bb8
External Bus	SRAM	SRAM	$f0d0 0bb9
Unused	SRAM	SRAM	$f0d0 0bb0
SRAM	SRAM	SRAM	$f0d0 0bbb

9.5 Secured Single Chip Mode

Secured Single Chip Mode is typically used to execute application code in the final application, where a software bootloader is not required. Booting is done from the Flash Main Array, with limited device functionality. In this mode, the following functionality is available:

- Debug functionality (EICE, Nexus and the JTAG interface) is disabled
  - SC4 chain is still enabled
- External Bus Interface is disabled
- Flash main array access is enabled
- Shadow Block access is enabled
- Boot from the Flash main array

Table 9-7 shows the device memory map of the device in Secured Single Chip Mode after Reset. The memory map for this mode differs from the memory map in Normal Single Chip Mode as follows:

- The External Bus Interface is not available

Table 9-7. Device Memory Map in Secured Single Chip Mode (After Reset)[1]

Address	Module	Size (Bytes)
$2000 0000 - $2007 FFFF $2000 0000 - $2004 FFFF	Flash Main Array (MAC72x1) Flash Main Array (MAC72x2)	512K 320K
$00F0 0000 - $00F0 7FFF	Shadow Block	32K
$0008 0000 - $3FFF FFFF	Reserved	~1023M
$4000 0000 - $4000 7FFF $4000 0000 - $4000 4FFF	SRAM[2] SRAM[2]	32K 20K
$4000 8000 - $FBFF FFFF	Reserved	3008M
$FC00 0000 - $FFFF FFFF	Peripheral Bus modules	64M

1. This reset memory map equates to a reset value of $f0d0 0b08 in the AAMR register in the MCM module.
2. The SRAM is mirrored across the entire 512MByte address range on a 32K boundary. Therefore, the first word in SRAM may be read from or written to using address $4000 0000 or address $4000 8000 or address $4001 0000, etc.

After Reset, the memory map may be re-mapped using the AAMR register in the MCM module. The allowed combinations are shown in Table 9-8. The Peripheral space is always fixed at $FC00 0000, and is not re-mappable.

Table 9-8. Allowed Memory Maps in Secured Single Chip Mode

$0000 0000	$2000 0000	$4000 0000	AAMR Register
Unused	Flash Main Array	SRAM	$f0d0 0b80
SRAM	Flash Main Array	SRAM	$f0d0 0b8b
Flash Main Array	Unused	SRAM	$f0d0 0b08
Unused	Unused	SRAM	$f0d0 0b00
SRAM	Unused	SRAM	$f0d0 0b0b
Flash Main Array	SRAM	SRAM	$f0d0 0bb8
Unused	SRAM	SRAM	$f0d0 0bb0
SRAM	SRAM	SRAM	$f0d0 0bbb

9.6 Secured Primary Bootloader Mode

Secured Primary Bootloader Mode is typically used to execute application code in the final application, where a software bootloader is required. Booting is done from the Shadow Block, with limited device functionality. Secured Primary Bootloader Mode is identical to Normal Primary Bootloader Mode, except that debug functionality (EICE, Nexus and the JTAG interface) is disabled and the FlexBus is turned off. Please refer to Section 9.3, “Normal Primary Bootloader Mode” for the functionality available and the memory map.

9.7 Secured Expanded Mode

Secured Single Expanded Mode is typically used to execute application code in the final application, where on-chip Flash is not required. Alternatively, it is also used to unsecure secured devices. Booting is

done from the external bus, with limited device functionality. In this mode, the following functionality is available:

- Debug functionality (EICE, Nexus and the JTAG interface) is enabled
- External Bus Interface is enabled
- Flash main array access is disabled
- Shadow Block access is disabled
- Boot from the External Bus

Table 9-9 shows the device memory map of the device in secured expanded mode after Reset. The memory map for this mode differs from the memory map in Normal Single Chip Mode as follows:

- The External Bus Interface is relocated to address $0000 0000
- The Flash Main Array is not available
- The Shadow Block is not available

Table 9-9. Device Memory Map in Secured Expanded Mode (After Reset)[1]

Address	Module	Size (Bytes)
$0000 0000 - $3FFF FFFF	External Bus Interface	512M
$4000 0000 - $4000 7FFF $4000 0000 - $4000 4FFF	SRAM[2] SRAM[2]	32K 20K
$4000 8000 - $FBFF FFFF	Reserved	3008M
$FC00 0000 - $FFFF FFFF	Peripheral Bus modules	64M

1. This reset memory map equates to a reset value of $f0d0 0b09 in the AAMR register in the MCM module.
2. The SRAM is mirrored across the entire 512MByte address range on a 32K boundary. Therefore, the first word in SRAM may be read from or written to using address $4000 0000 or address $4000 8000 or address $4001 0000, etc.

After Reset, the memory map may be re-mapped using the AAMR register in the MCM module. The allowed combinations are shown in Table 9-10. The Peripheral space is always fixed at $FC00 0000, and is not re-mappable.

Table 9-10. Allowed Memory Maps in Secured Expanded Mode

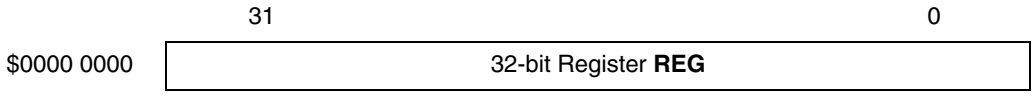
$0000 0000	$2000 0000	$4000 0000	AAMR Register
External Bus	External Bus	SRAM	$f0d0 0b99
Unused	External Bus	SRAM	$f0d0 0b90
SRAM	External Bus	SRAM	$f0d0 0b9b
External Bus	Unused	SRAM	$f0d0 0b09
Unused	Unused	SRAM	$f0d0 0b00
SRAM	Unused	SRAM	$f0d0 0b0b
External Bus	SRAM	SRAM	$f0d0 0bb9
Unused	SRAM	SRAM	$f0d0 0bb0
SRAM	SRAM	SRAM	$f0d0 0bbb

9.8 Accessing registers

All registers use bit 31 to represent the most significant bit (msb), and bit 0 to represent the least significant bit (lsb). All memory-mapped registers may be accessed using properly aligned 8-bit, 16-bit or 32-bit accesses, except where indicated. This means that all addresses may be accessed using 8-bit accesses, all even addresses may be accessed using 16-bit accesses, and every fourth address may be accessed using 32-bit accesses. On the MAC7200 family, which are big-endian devices, this results in the following register mapping:

9.8.1 32-bit Register Accesses

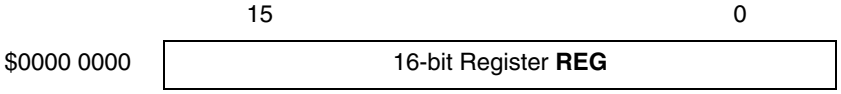
A read to the above register will result in the following values:



All other accesses (i.e., 16-bit READ from $0000 0001) are not allowed.

8-bit READ from $0000 0000	returns REG[31:24]
8-bit READ from $0000 0001	returns REG[23:16]
8-bit READ from $0000 0002	returns REG[15:8]
8-bit READ from $0000 0003	returns REG[7:0]
16-bit READ from $0000 0000	returns REG[31:16]
16-bit READ from $0000 0002	returns REG[15:0]
32-bit READ from $0000 0000	returns REG[31:0]

9.8.2 16-bit Register Accesses

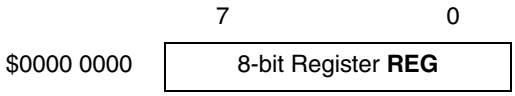


A read to the above register will result in the following values:

8-bit READ from $0000 0000	returns REG[15:8]
8-bit READ from $0000 0001	returns REG[7:0]
16-bit READ from $0000 0000	returns REG[15:0]
32-bit READ from $0000 0000 returns	REG[15:0] + the next 16 bits in the memory map

All other accesses (i.e.-16-bit READ from $0000 0001) are not allowed.

9.8.3 8-bit register accesses



A read to the above register will result in the following values:

8-bit READ from $0000 0000	returns REG[7:0]
16-bit READ from $0000 0000	returns REG[7:0] + the next 8 bits in the memory map
32-bit READ from $0000 0000	returns REG[7:0] + the next 24 bits in the memory map

All other accesses (i.e.-16-bit READ from $0000 0001) are not allowed.

It is possible to enable bus aborts on illegal register accesses. Please refer to Chapter 25, "System Service Module (SSM_MAC7202)" for more information on this.

9.9 Peripheral Bus Memory Map

The Peripheral Bus bridge (AIPS) is located on a slave port of the AXBS crossbar switch with a reset value of $E000 0000, and is not relocatable. However, the Peripheral Bus modules are located above this boundary, at $FC00 0000. Table 9-11 shows the address of all Peripheral Bus modules.

Table 9-11. Peripheral Bus Memory Map

Address	Module	Size (Bytes)	(O)PACR
$FC00 0000 - $FC00 3FFF	AMBA to IP Bus Bridge (AIPS) Configuration Registers	16K	PACR0
$FC00 4000 - $FC00 7FFF	ARM Crossbar Switch (AXBS) Configuration Registers	16K	PACR1
$FC00 8000 - $FC00 BFFF	External Interface Module (EIM) Configuration Registers	16K	PACR2
$FC00 C000 - $FC03 FFFF	Reserved	208K	
$FC04 0000 - $FC04 3FFF	Miscellaneous Control Module (MCM)	16K	PACR16
$FC04 4000 - $FC04 7FFF	Enhanced Direct Memory Access Controller (eDMAC)	16K	PACR17
$FC04 8000 - $FC04 BFFF	Interrupt Controller (INTC)	16K	PACR18
$FC04 C000 - $FC07 FFFF	Reserved	212K	
$FC08 0000 - $FC08 3FFF	System services Module (SSM)	16K	OPACR0
$FC08 4000 - $FC08 7FFF	DMA Channel Multiplexer (DMA_CH_MUX)	16K	OPACR1
$FC08 8000 - $FC08 BFFF	Clock and Reset Generator (CRG)	16K	OPACR2
$FC08 C000 - $FC08 FFFF	Programmable Interval Timer (PIT)	16K	OPACR3
$FC09 0000 - $FC09 3FFF	Reserved	16K	
$FC09 4000 - $FC09 7FFF	CAN controller 0 (FlexCAN_A)	16K	OPACR5
$FC09 8000 - $FC09 BFFF	CAN controller 1 (FlexCAN_B)	16K	OPACR6
$FC09 C000 - $FC09 FFFF	Reserved	16K	
$FC0A 0000 - $FC0A 3FFF	Reserved	16K	
$FC0A 4000 - $FC0A 7FFF	Reserved	16K	
$FC0A 8000 - $FC0A BFFF	Reserved	16K	
$FC0A C000 - $FC0A FFFF	Inter-IC bus (IIC)	16K	OPACR11
$FC0B 0000 - $FC0B 3FFF	Reserved	16K	
$FC0B 4000 - $FC0B 7FFF	Serial Peripheral Interface 0 (DSPI_A)	16K	OPACR13

Table 9-11. Peripheral Bus Memory Map (Continued)

Address	Module	Size (Bytes)	(O)PACR
$FC0B 8000 - $FC0B BFFF	Serial Peripheral Interface 1 (DSPI_B)	16K	OPACR14
$FC0B C000 - $FC0B FFFF	Serial Peripheral Interface 2 (DSPI_C)	16K	OPACR15
$FC0C 0000 - $FC0C 3FFF	Reserved	16K	
$FC0C 4000 - $FC0C 7FFF	Enhanced Serial Communication Interface 0 (ESCI_A)	16K	OPACR17
$FC0C 8000 - $FC0C BFFF	Enhanced Serial Communication Interface 1 (ESCI_B)	16K	OPACR18
$FC0C C000 - $FC0C FFFF	Reserved	16K	
$FC0D 0000 - $FC0D 3FFF	Reserved	16K	
$FC0D 4000 - $FC0D 7FFF	Reserved	16K	
$FC0D 8000 - $FC0D BFFF	Reserved	16K	
$FC0D C000 - $FC0D FFFF	Enhanced Modular I/O Subsystem (eMIOS)	16K	OPACR23
$FC0E 0000 - $FC0E 3FFF	Analog to Digital Converter (ATD_A)	16K	OPACR24
$FC0E 4000 - $FC0E 7FFF	Reserved	16K	
$FC0E 8000 - $FC0E BFFF	Port Integration Module (PIM)	16K	OPACR26
$FC0E C000 - $FC0E FFFF	Reserved	16K	
$FC0F 0000 - $FC0F 3FFF	Flash registers	16K	OPACR28fs
$FC0F 4000 - $FFFF FFFF	Reserved	~63M	

9.10 SRAM Memory Map

Like the Peripheral Bus modules, the SRAM resides on a slave port of the AXBS, and therefore has a movable base address on a 512MByte boundary. However, unlike the Peripheral Bus, the SRAM is “mirrored” in the 512MByte space on a 32KB boundary. This gives us the following memory map for the SRAM:

Table 9-12. SRAM Memory Map

Address Offset	Function
$0000 0000 $0000 7FFF	SRAM
$0000 8000 $0000 FFFF	SRAM
$0001 0000 $0001 7FFF	SRAM
: : : :	: : : :

Table 9-12. SRAM Memory Map (Continued)

Address Offset	Function
$1FFF 0000 $1FFF 7FFF	SRAM
$1FFF 8000 $1FFF FFFF	SRAM

Because of the odd size of the SRAM (20KB) and the mirroring on a 32KB boundary, there will be a “hole” in the upper 16KB of each mirrored 32KB where no reads or writes are allowed. Any write or read to this hole will produce a bus abort. Please refer to Section 19.1.6.2, “SRAM Address Mirroring” for more details on the mirroring in the SRAM controller.

The default base address for the SRAM is $4000 0000 and is not selectable during reset, but is software selectable after reset.

9.11 FlexBus Memory Map

Like the Peripheral Bus modules, the FlexBus resides on a slave port of the AXBS, and therefore has a movable base address on a 512MByte boundary. In addition, the FlexBus has 3 chip-selects, where each chip select can select from 64KBytes to 512MBytes, and may overlap.

Table 9-13. FlexBus Memory Map

Address Offset	Function
$0000 0000 $1FFF FFFF	FlexBus

The default base address for the FlexBus depends on the chip mode selected (See Figure 9-1) and is software selectable after reset. Note that in some modes, the Flexbus is not available in the memory map (for security reasons).

9.12 Flash Main Array Memory Map

The Flash main array is relocatable on any 512 MByte boundary, including $0000 0000 (via the AAMR register in the MCM). The reset location is $0000 0000 in Single Chip Mode, and $2000 0000 in Expanded Chip and PBL Mode. The base address is software selectable in any chip mode. Note that in some modes, the Flash Main Array is not available in the memory map (for security reasons).

Table 9-14. MAC72x1 Flash Main Array Memory Map

Function	Partition (Size)	Block (Size)	Address Offset
LAS	Partition 2 (128K)	LAS Block 8 (64K)	$0000 0000 $0000 FFFF
		LAS Block 9 (64K)	$0001 0000 $0001 FFFF
	Partition 0 (64K)	LAS Block 0 (16K)	$0002 0000 $0002 3FFF
		LAS Block 1 (16K)	$0002 4000 $0002 7FFF
		LAS Block 2 (16K)	$0002 8000 $0002 BFFF
		LAS Block 3 (16K)	$0002 C000 $0002 FFFF
	Partition 1 (64K)	LAS Block 4 (16K)	$0003 0000 $0003 FFFF
		LAS Block 5 (16K)	$0003 4000 $0003 7FFF
		LAS Block 6 (16K)	$0003 8000 $0003 BFFF
		LAS Block 7 (16K)	$0003 C000 $0003 FFFF
MAS	Partition 3 (256K)	MAS Block 1 (128K)	$0004 0000 $0005 FFFF
		MAS Block 0 (128K)	$0006 0000 $0007 FFFF
HAS	—	Not Mapped on MAC72x1	$0008 0000 $000F FFFF
—	—	Not Mapped	$0010 0000 $00EF FFFF
—	—	Shadow Block (32K)	$00F0 0000 $00F0 7FFF

Table 9-15. MAC72x2 Flash Main Array Memory Map

Function	Partition (Size)	Block (Size)	Address Offset
LAS	Partition 0 (256K)	LAS Block 0 (128K)	$0000 0000 $0001 FFFF
		LAS Block 1 (128K)	$0002 0000 $0003 FFFF
MAS	Partition 1 (32K)	MAS Block 0 (16K)	$0004 0000 $0004 3FFF
		MAS Block 1 (16K)	$0004 4000 $0004 7FFF
	Partition 2 (32K)	MAS Block 2 (16K)	$0004 8000 $0004 BFFF
		MAS Block 3 (16K)	$0004 C000 $0004 FFFF
MAS	—	Not Mapped on MAC72x2	$0005 0000 $0007 FFFF
HAS	—	Not Mapped on MAC72x2	$0008 0000 $000F FFFF
—	—	Not Mapped	$0010 0000 $00EF FFFF
—	—	Shadow Block (32K)	$00F0 0000 $00F0 7FFF

NOTE

Accesses to areas not mapped will produce a Prefetch or Data Abort exception (depending on the access type).

9.13 Shadow Block Memory Map

The 32K Shadow Block is always available at FLASH_BASE + $00F0 0000 (when the Flash is available in the memory map). In addition, when resetting into PBL mode, it is mirrored at address $0000 0000. When the Flash Main Array is not available in the memory map (for security reasons), the Shadow Block is also unavailable.

Table 9-16. Shadow Block Memory Map[1]

Address Offset	Function
$0000 0000 $0000 7DDF	Shadow Block (User Area)
$0000 7DE0	System Censor Word
$0000 7DE8	LML Register Shadow

Table 9-16. Shadow Block Memory Map[1] (Continued)

Address Offset	Function
$0000 7DF0	HBL Register Shadow
$0000 7DF8	SLL Register Shadow
$0000 7E00	PFCR2 (BIU2) Register Shadow
$0000 7E08	PFSACC (BIU3) Register Shadow
$0000 7E10	PFDACC (BIU4) Register Shadow
$0000 7E18 $0000 7FFF	Shadow Block (User Area)

1. This memory map is assuming a 32K Shadow Block. If the size of the Shadow Block changes, the registers will move accordingly (i.e.-the registers are always placed at the end of the Shadow Block in order to allow the maximum amount of contiguous user area possible).

9.14 Boot Assist Module (BAM) Memory Map

The BAM, unlike other resources, is not software relocatable in the system, and is always present at $A000 0000, regardless of chip mode. When JTAG Lockout Recovery is triggered, the BAM is mirrored to address $0000 0000, and the Lockout Recovery firmware is executed after reset. Although it is generally not used by the application, it is accessible in software at address $A000 0000.

Table 9-17. Boot Assist Module (BAM) Memory Map

Address Offset	Function
$0000 0000 $1FFF FFFF	BAM

9.15 Exception Table Memory Map

The ARM7 exception vector table is located at $0000 0000 to $0000 001F, and may not be relocated either at reset or after reset. Please refer to Chapter 7, “Exceptions” for more details on exceptions.

Table 9-18. Exception Table Memory Map

Address Offset	Function
$0000 0000	Reset exception
$0000 0004	Undefined Instruction exception
$0000 0008	Software Interrupt (SWI) exception
$0000 000c	Prefetch Abort exception
$0000 0010	Data Abort exception
$0000 0014	Reserved

Table 9-18. Exception Table Memory Map (Continued)

Address Offset	Function
$0000 0018	IRQ exception
$0000 001c	FIRQ exception

9.16 Memory Map Relocation

The MAC72xx is designed to have relocatable memory map resources, for the following reasons:

- The ARM7 exception table is not relocatable. Therefore, in order to execute exception handlers from different memories, relocation of these memories is necessary.
- Security of the Flash may dictate that certain combinations of resources should not be valid on a secured device.
- Support for a software bootloader concept
- General system flexibility

As previously discussed, relocation of resources is generally available via the crossbar switch on a 512MB boundary. In addition, certain resources (SRAM and Shadow Block) may also be mirrored to give an extra granularity of relocation. The following sections discuss the details of this relocation.

9.16.1 System Memory Map Combinations

Possible combinations that can be configured via software after reset are listed in Table 9-19. Shaded lines are those combinations that are not allowed in the MCM.

Table 9-19. Possible Memory Map Configurations[1]

$0000 0000	$2000 0000	$4000 0000	$A000 0000	$E000 0000
Unused	Unused	SRAM	BAM	Peripheral Bus
Unused	FLASH	SRAM	BAM	Peripheral Bus
Unused	FlexBus	SRAM	BAM	Peripheral Bus
Unused	SRAM	SRAM	BAM	Peripheral Bus
FLASH	Unused	SRAM	BAM	Peripheral Bus
FLASH	FLASH	SRAM	BAM	Peripheral Bus
FLASH	FlexBus	SRAM	BAM	Peripheral Bus
FLASH	SRAM	SRAM	BAM	Peripheral Bus
FlexBus	Unused	SRAM	BAM	Peripheral Bus
FlexBus	FLASH	SRAM	BAM	Peripheral Bus
FlexBus	FlexBus	SRAM	BAM	Peripheral Bus
FlexBus	SRAM	SRAM	BAM	Peripheral Bus

Table 9-19. Possible Memory Map Configurations[1] (Continued)

$0000 0000	$2000 0000	$4000 0000	$A000 0000	$E000 0000
SRAM	Unused	SRAM	BAM	Peripheral Bus
SRAM	FLASH	SRAM	BAM	Peripheral Bus
SRAM	FlexBus	SRAM	BAM	Peripheral Bus
SRAM	SRAM	SRAM	BAM	Peripheral Bus
BAM	FLASH	SRAM	BAM	Peripheral Bus

1. The Shadow Block is not included in the above table, because it is generally not relocatable. In Normal/Secured Bootloader Modes, the Shadow Block is located at address $0000 0000 at reset, but is not relocatable after reset. In all other modes, if the Flash is available, then the Shadow Block is also available (at FLASH_BASE + $00F0 0000).

9.16.2 Changing Chip Modes

Since changing the security mode of the Flash requires a system reset afterwards, the chip modes can only be switched by resetting the device, and re-applying the correct signals to the MODA/MODB pins.

9.16.3 Resource Relocation Summary

Also refer to Chapter 2, “Modes of Operation”.

9.16.3.1 FlexBus

The FlexBus is generally software relocatable (via the AAMR register in the MCM) to the following addresses:

- $0000 0000
- $2000 0000

9.16.3.2 Flash Main Array

The Flash Main Array is generally software relocatable (via the AAMR register in the MCM) to the following addresses:

- $0000 0000
- $2000 0000

9.16.3.3 Shadow Block

The relocation of the Shadow Block is done by relocating the Flash main array.

In Normal/Secured Bootloader Mode, where the Shadow Block is relocated to address $0000 0000, once another resource (like the SRAM) has been relocated to address $0000 0000, the Shadow Block can not be relocated back to address $0000 0000 without resetting the device. The exact mechanism for this is described in Section 9.16.4, “Programming the AAMR register in the MCM”.

9.16.3.4 SRAM

The SRAM is always software relocatable (via the AAMR register in the MCM) to the following addresses:

- $0000 0000
- $2000 0000
- $4000 0000

9.16.4 Programming the AAMR register in the MCM

Because the programming of this register is initially slightly counter-intuitive, a short explanation of how it works is presented.

Each 512Mbyte address range in the MAC72xx's 4GByte addressable space has a corresponding nibble (called a "slot" hereafter). When an transfer is seen on a Master Port of the AXBS (crossbar switch), the upper three address bits are used to determine which AAMR slot will be used to determine the correct Slave Port to route to. The slots in the AAMR are allocated as follows:

Base Address	$E000 0000	$C000 0000	$A000 0000	$8000 0000
AAMR Bits	31:28	27:24	23:20	19:16

Base Address	$6000 0000	$4000 0000	$2000 0000	$0000 0000
AAMR Bits	15:12	11:8	7:4	3:0

The upper bit of each nibble/slot is the valid bit, 1=valid, 0=invalid. In the case of an invalid address, the transfer is aborted. The lower 3 bits of each nibble/slot determine the Slave Port number that the transfer will be routed to. The MAC72xx has the following valid Slave Ports:

Table 9-20. AXBS Slave Port Definitions

Slave Port	Resource (AAMR Value)
S0	FLASH ($8)
S1	FlexBus ($9)
S3	SRAM ($b)
S5	BAM ($d)
S7	Peripheral Bus ($f)

NOTE

Bits [31:8] of the AAMR are always fixed at value $f0d00b. If any nibble is written with an illegal value, then none of the register nibbles will be updated.

In Normal/Secured Bootloader Mode, when the Shadow Block is located at $0000 0000, the reset value of the AAMR register will be $88.

In Normal/Secured Bootloader mode, once AAMR[3] has been set (i.e.-Nibble 0 is marked as valid), the Shadow Block is no longer available at address $0000 0000. Any write to the AAMR that does not set bit 3 will leave the Shadow Block mirroring at address $0000 0000. As an example, in Normal/Secured Bootloader mode, the initial AAMR register value is $f000 0b88. If the user writes $f000 0bb0 to the AAMR register, the following configuration will be set:

Table 9-21.

Before		After	
$0000 0000	Shadow Block	$0000 0000	Shadow Block
$2000 0000	Main Flash array	$2000 0000	SRAM
$20F0 0000	Shadow Block		

However, if the user writes $f000 0bb8 to the AAMR register, the following configuration will be set:

Table 9-22.

Before		After	
$0000 0000	Shadow Block	$0000 0000	Main Flash array
		$20F0 0000	Shadow Block
$2000 0000	Main Flash array	$2000 0000	SRAM
$20F0 0000	Shadow Block		

9.17 Exception Table

See Section 7.2, “Exception Handling”.

9.18 System Boot Sequence

9.18.1 Programming with a Bootloader

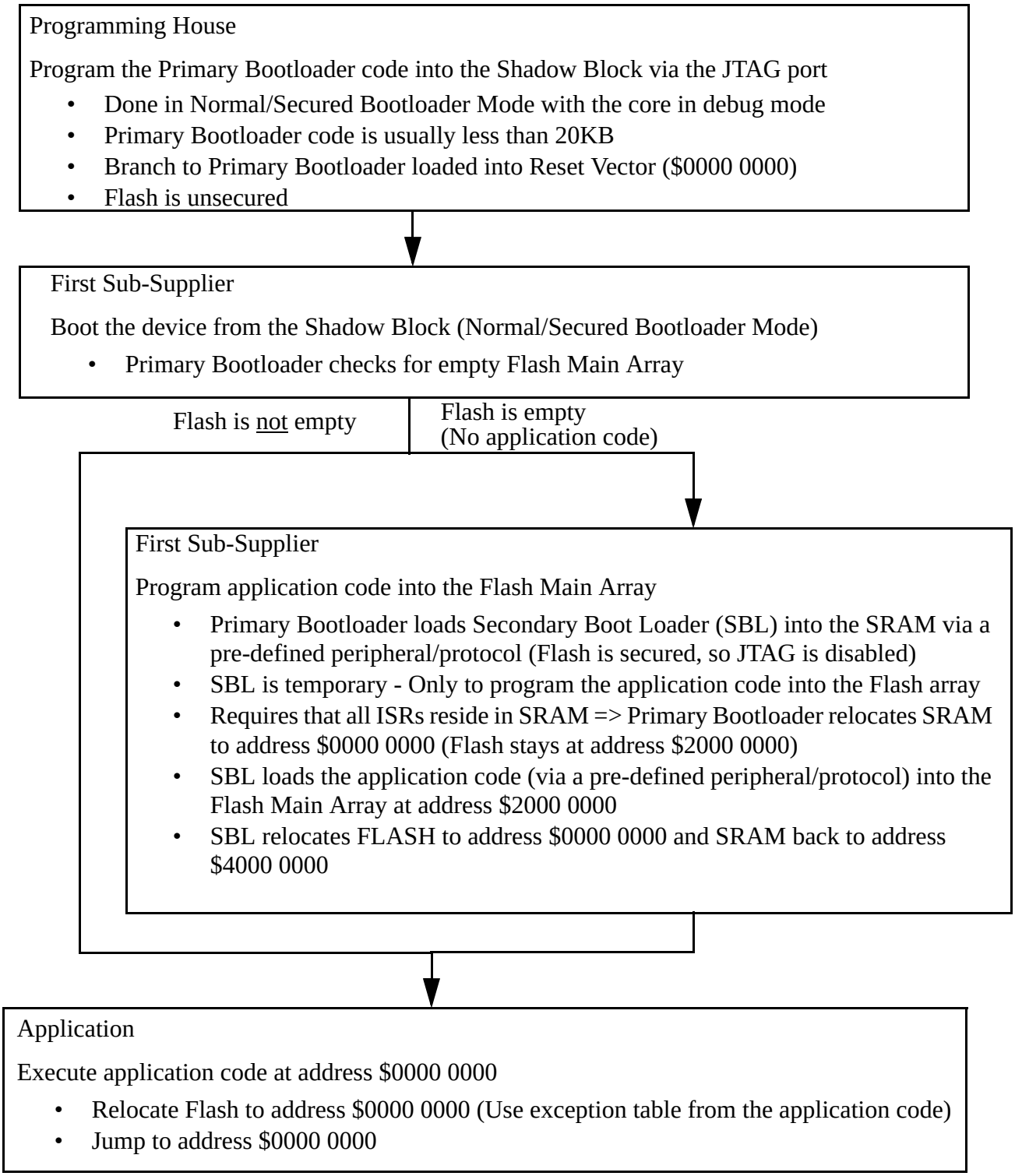


Figure 9-4. Device Programming Sequence

9.18.2 “Normal” Boot with a Bootloader

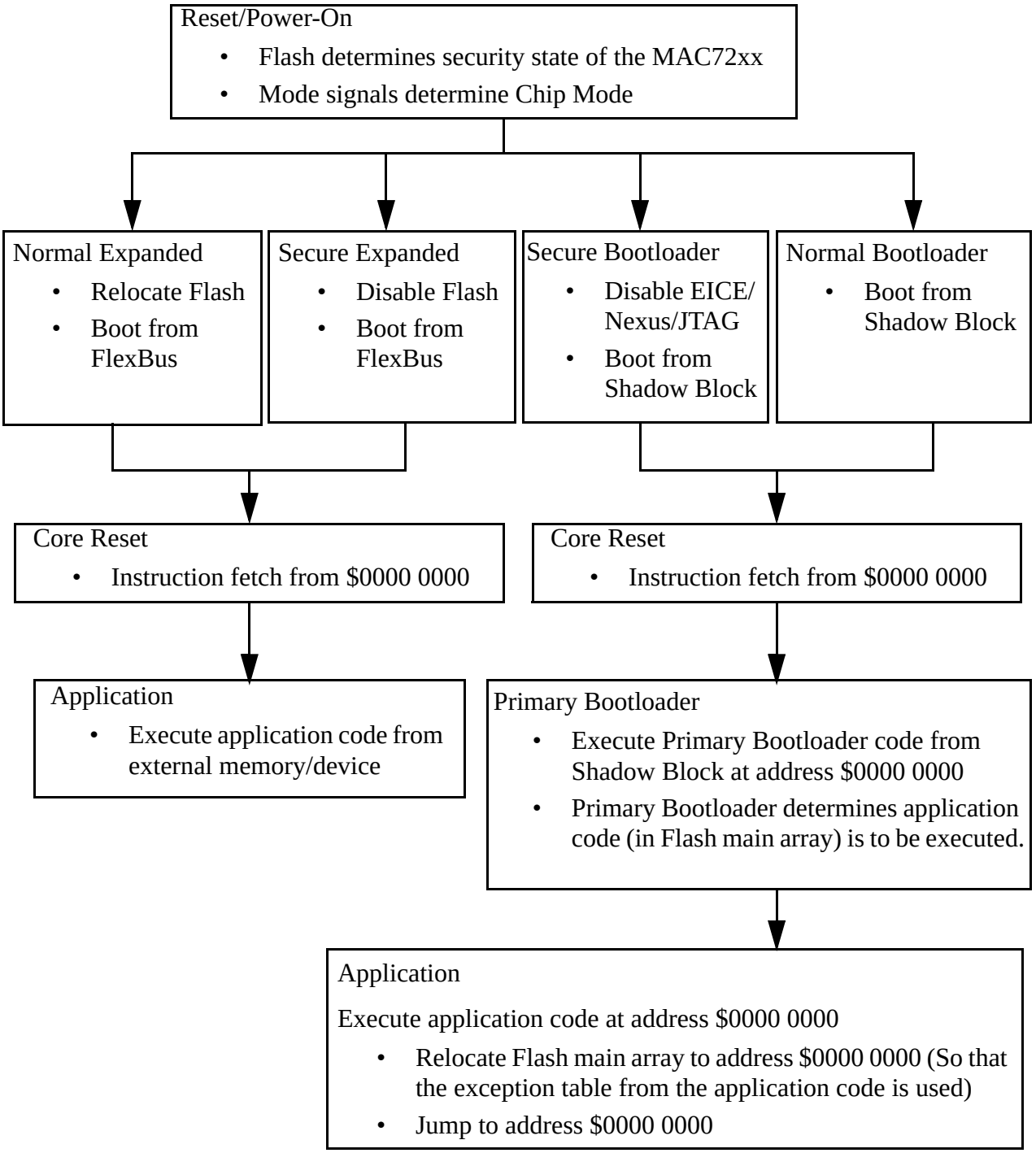


Figure 9-5. Device “Normal” Boot Sequence

Note that in both Programming and “Normal” modes, the system is required to dynamically re-map a memory into address $0000 0000. Because of the pre-fetch/fetch pipeline in the ARM7 core, this requires some “overlap” between the code in the original memory and the newly swapped in memory. The easiest way to do this is to insert NOP opcodes into both sets of code at the right location. The following example illustrates this, using the switch between the bootloader code in the Shadow Block, and the application code in the Instruction Flash. Note that the example uses “pseudo assembly code”, rather than actual ARM7 code.

Address	Original Code (PBL) in the Shadow Block	New Code (App. Code) in the Flash Main Array
$0000 0000	JMP $0000_0000	NOP
$0000 0004	NOP	NOP
$0000 0008	NOP	NOP
$0000 000C	NOP	App. Code starts here...

At least 3 NOPs should be used in both the “original” and “new” code.

Chapter 10
ARM7TDMI-S Core

10.1 Introduction

The MAC7200 family is implemented with a licensed ARM7 CPU. This is a 32-bit RISC core with a three stage pipeline offering high instruction throughput. The core used across the family is the ARM7TDMI-S which supports both a 32-bit instruction set and a 16-bit instruction set (THUMB) to assist with code density. No architectural modifications have been made to this core during implementation, enabling the MAC7200 family to remain compliant to the ARM ISA V4T and existing tool chains. The core has been configured to support Big Endian memory systems.

For detailed information about the ARM core please consult the following documents:

- ARM Architecture Reference Manual (Second Edition)
- ARM7TDMI-S (Rev 4) Technical Reference Manual (Issue A)

10.2 ARM7 Features

- 32-bit ARM7 TDMI-S RISC Core
  - Up to 70MHz operating frequency
  - Efficient code density through 16-bit instructions (THUMB).
  - Alternate general purpose registers.
  - Byte (8-bit), Halfword (16-bit), Word (32-bit) data types supported.
  - Cores and Memory connected using high performance AMBA AHB bus.
  - Integrated E-ICE module for debug
- Version rev4p3.04

T	Thumb architecture extension
D	Debug extensions
M	Enhanced multiplier (32x8) with 64-bit result support
I	EmbeddedICE Macrocell extension
S	Fully Synthesizable

10.3 ARM7 Implementation

The MAC72xx devices use the latest version of the ARM7TDMI-S core available. For the MAC72x2 xM84A masksets and MAC72x1 0M19G maskset, revision rev4p3.04 is used.

10.4 ARM7 External Pins

There are no ARM7 signals that drive or are driven from MCU pins.

10.5 ARM7 Bus Aborts

As the ARM7 core is a bus master, it is not capable of generating bus aborts.

10.6 ARM7 Application Usage

Please refer to ARM7 Application Notes from ARM for more information.

10.6.1 Register Bank Initialization

It is necessary to initialize all ARM7 user registers, as they are not initialized by a POR or System Reset.

Chapter 11
A7S Nexus3 Module

11.1 Introduction

The A7S Nexus3 module provides real-time development capabilities for ARM7™ core based microcontrollers in compliance with the IEEE-ISTO Nexus 5001. This module provides development support capabilities for MCUs without requiring address and data pins for internal visibility.

A portion of the pin interface is also compliant with the IEEE 1149.1 JTAG standard. The IEEE-ISTO 5001 standard defines an extensible auxiliary port which, for ARM7, will be used in conjunction with the JTAG port.

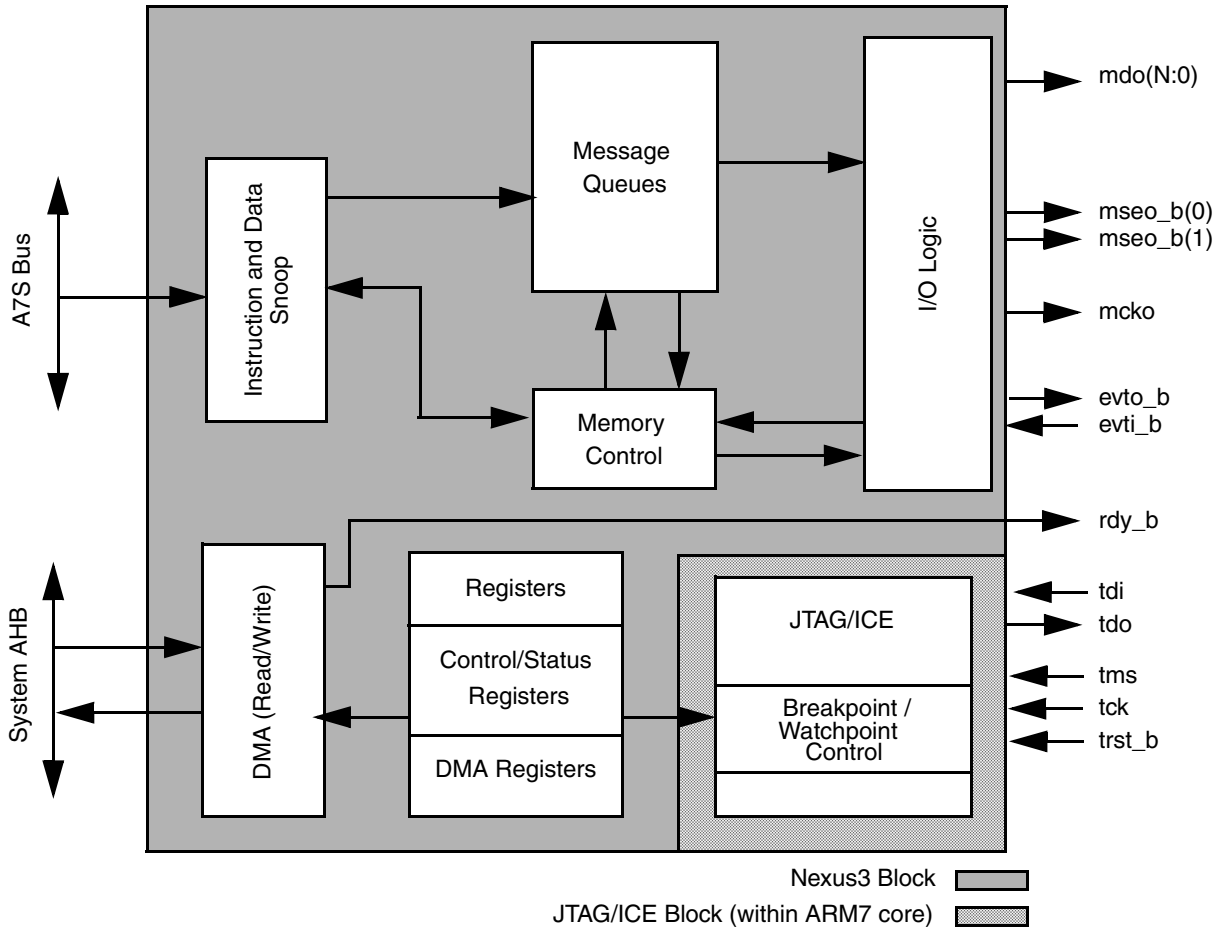


Figure 11-1. A7S Nexus3 Functional Block Diagram

11.1.1 A7S Nexus3 Overview

This specification defines the auxiliary pin functions, transfer protocols and standard development features of a Class 3 device in compliance with the IEEE-ISTO Nexus 5001 standard**.** The development features supported are Program Trace, Data Trace, Watchpoint Messaging, Breakpoints, Ownership Trace and Read/Write Access via the JTAG interface. The A7S Nexus3 module also supports two Class 4 features: Watchpoint Triggering, and processor overrun control.

11.1.2 Nexus Feature List

The A7S Nexus3 module is compliant with Class 3 of the IEEE-ISTO 5001 standard. The following features are implemented:

- 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.
  - Program Trace in ARM mode will use Branch/Predicate History Messaging due to the conditional nature of 32-bit ARM instructions
  - Program Trace in Thumb Mode can be programmed to use the Branch History Messaging method, or use traditional Program Trace – Direct/Indirect Branch Messaging
- 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.
- Breakpoint on instruction address.
- Watchpoint Trigger enable of Program and/or Data Trace Messaging
- Auxiliary interface for higher data input/output
  - Configurable two (2) or eight (8) MDO (Message Data Out) pins
  - One (1) or two (2) $\overline{\text{MSEO}}$ (Message Start/End Out) pins
  - One (1) $\overline{\text{RDY}}$ (Read/Write Ready) pin
  - One (1) $\overline{\text{EVTO}}$ (Watchpoint Event) pin
  - One (1) $\overline{\text{EVTI}}$ (Event In) pin
  - One (1) MCKO (Message Clock Out) pin
- 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

11.1.3 Modes of Operation

There are three (3) basic modes of operation for the A7S Nexus3 block.

- Reset
- Normal
- Disabled

11.1.3.1 Reset

The reset configuration is received via the $\overline{\text{EVTI}}$ pin to enable or disable the A7S Nexus3 module. $\overline{\text{EVTI}}$ is sampled synchronously at the exit from the JTAG Test-Logic-Reset state. Reset configuration information must be valid on $\overline{\text{EVTI}}$ during the JTAG “Test Logic Reset” state (see 11.12, “IEEE 1149.1 State Machine and RD/WR Sequences”).

The A7S Nexus3 module will disable (drive inactive) the output pins during the JTAG Test-Logic-Reset state or when a Power-on-Reset (POR) event occurs.

11.1.3.2 Normal

If $\overline{\text{EVTI}}$ is asserted at the exit from JTAG Test-Logic-Reset, the A7S Nexus3 module will be enabled. The module will be ready to accept control input via the JTAG pins.

The A7S Nexus3 module may also be enabled by loading the NEXUS-ACCESS instruction into the JTAG Instruction Register. Once the A7S Nexus3 module has been enabled, it will remain enabled until entry into the JTAG Test-Logic-Reset state.

11.1.3.3 Disabled

If $\overline{\text{EVTI}}$ is negated at the exit from JTAG Test-Logic-Reset, the A7S Nexus3 module will be disabled. No trace output will be provided, auxiliary port output pins ($\overline{\text{MDO}}$, $\overline{\text{MSEO}}$, MCKO) will be disabled (driven inactive), and Nexus3 registers will not be accessible for reads or writes.

NOTE

If there is no debug/development tool connected to the SoC, the $\overline{\text{EVTI}}$ pin should be held de-asserted in order to keep the Nexus3 module disabled.

The A7S Nexus3 module may be enabled after the exit from JTAG Test-Logic-Reset state by loading the NEXUS-ACCESS instruction into the JTAG Instruction Register.

11.1.4 TCODEs supported

The A7S Nexus3 pins allow for flexible transfer operations via Public Messages. A TCODE defines the transfer format, the number and/or size of the packets to be transferred, and the purpose of each packet. The IEEE-ISTO 5001 standard defines a set of public messages. The A7S Nexus3 block supports the public TCODEs seen in Table 11-1.

Table 11-1. Public TCODEs Supported

Message Name	Minimum Packet Size (bits)	Maximum Packet Size (bits)	Packet Name	Packet Type	Packet Description
Debug Status	6	6	TCODE	fixed	TCODE number = 0
	4	4	SRC	fixed	source processor identifier (multiple Nexus configuration)
	8	8	STATUS	fixed	Debug Status Register (DS[31:24])
Ownership Trace Message	6	6	TCODE	fixed	TCODE number = 2
	4	4	SRC	fixed	source processor identifier (multiple Nexus configuration)
	32	32	PROCESS	fixed	Task/Process ID tag
Program Trace – Direct Branch Message	6	6	TCODE	fixed	TCODE number = 3 (see Note below)
	4	4	SRC	fixed	source processor identifier (multiple Nexus configuration)
	1	8	I-CNT	variable	# sequential instructions executed since last taken branch
Program Trace – Indirect Branch Message	6	6	TCODE	fixed	TCODE number = 4 (see Note below)
	4	4	SRC	fixed	source processor identifier (multiple Nexus configuration)
	1	8	I-CNT	variable	# sequential instructions executed since last taken branch
	1	32	U-ADDR	variable	unique part of target address for taken branches/exceptions
Data Trace – Data Write Message	6	6	TCODE	fixed	TCODE number = 5
	4	4	SRC	fixed	source processor identifier (multiple Nexus configuration)
	3	3	DSZ	fixed	data size (Refer to Table 11-6)
	1	32	U-ADDR	variable	unique portion of the data write address
	1	32	DATA	variable	data write value (see Note)
Data Trace – Data Read Message	6	6	TCODE	fixed	TCODE number = 6
	4	4	SRC	fixed	source processor identifier (multiple Nexus configuration)
	3	3	DSZ	fixed	data size (Refer to Table 11-6)
	1	32	U-ADDR	variable	unique portion of the data read address
	1	32	DATA	variable	data read value (see Note)
Error Message	6	6	TCODE	fixed	TCODE number = 8
	4	4	SRC	fixed	source processor identifier (multiple Nexus configuration)
	5	5	ERROR	fixed	error code (Refer to Table 11-2)
Program Trace – Direct Branch Message w/ Sync	6	6	TCODE	fixed	TCODE number = 11 (see Note below)
	4	4	SRC	fixed	source processor identifier (multiple Nexus configuration)
	1	8	I-CNT	variable	# sequential instructions executed since last taken branch
	1	32	F-ADDR	variable	full target address (leading zero (0) truncated)

Table 11-1. Public TCODEs Supported (Continued)

Message Name	Minimum Packet Size (bits)	Maximum Packet Size (bits)	Packet Name	Packet Type	Packet Description
Program Trace – Indirect Branch Message w/ Sync	6	6	TCODE	fixed	TCODE number = 12 (see Note below)
	4	4	SRC	fixed	source processor identifier (multiple Nexus configuration)
	1	8	I-CNT	variable	# sequential instructions executed since last taken branch
	1	32	F-ADDR	variable	full target address (leading zero (0) truncated)
Data Trace – Data Write Message w/ Sync	6	6	TCODE	fixed	TCODE number = 13
	4	4	SRC	fixed	source processor identifier (multiple Nexus configuration)
	3	3	DSZ	fixed	data size (Refer to Table 11-6)
	1	32	F-ADDR	variable	full access address (leading zero (0) truncated)
	1	32	DATA	variable	data write value (see Note)
Data Trace – Data Read Message w/ Sync	6	6	TCODE	fixed	TCODE number = 14
	4	4	SRC	fixed	source processor identifier (multiple Nexus configuration)
	3	3	DSZ	fixed	data size (Refer to Table 11-6)
	1	32	F-ADDR	variable	full access address (leading zero (0) truncated)
	1	32	DATA	variable	data read value (see Note)
Watchpoint Message	6	6	TCODE	fixed	TCODE number = 15
	4	4	SRC	fixed	source processor identifier (multiple Nexus configuration)
	8	8	WPHIT	fixed	watchpoint source(s) (refer to Table 11-3)
Resource Full Message	6	6	TCODE	fixed	TCODE number = 27
	4	4	SRC	fixed	source processor identifier (multiple Nexus configuration)
	4	4	RCODE	fixed	resource code (Refer to Table 11-4)
	1	32	HIST	variable	branch / predicate instruction history
Program Trace – Indirect Branch History Message	6	6	TCODE	fixed	TCODE number = 28 (see Note below)
	4	4	SRC	fixed	source processor identifier (multiple Nexus configuration)
	1	8	I-CNT	variable	# sequential instructions executed since last taken branch
	1	32	U-ADDR	variable	unique part of target address for taken branches/exceptions
	1	32	HIST	variable	branch / predicate instruction history
Program Trace – Indirect Branch History Message w/ Sync	6	6	TCODE	fixed	TCODE number = 29 (see Note below)
	4	4	SRC	fixed	source processor identifier (multiple Nexus configuration)
	1	8	I-CNT	variable	# sequential instructions executed since last taken branch
	1	32	F-ADDR	variable	full target address (leading zero (0) truncated)
	1	32	HIST	variable	branch / predicate instruction history

Table 11-1. Public TCODEs Supported (Continued)

Message Name	Minimum Packet Size (bits)	Maximum Packet Size (bits)	Packet Name	Packet Type	Packet Description
Program Trace – Program Correlation Message	6	6	TCODE	fixed	TCODE number = 33
	4	4	SRC	fixed	source processor identifier (multiple Nexus configuration)
	4	4	ECODE	fixed	event correlated with program flow (Refer to Table 11-5)
	1	8	I-CNT	variable	# sequential instructions executed since last taken branch
	1	32	HIST	variable	branch / predicate instruction history

NOTE

Due to the conditional nature of 32-bit ARM instructions, when in ARM mode, Program Trace will be implemented using Branch History/Predicate Instruction Messages. When in Thumb mode, the user can select between traditional Program Trace using Direct/Indirect Branch Messages, or Branch History Messages.

If the Branch History method is selected in Thumb mode, the shaded TCODES above will not be messaged out.

Table 11-2. Error Code Encoding (TCODE = 8)

Error Code	Description
00000	Ownership Trace overrun
00001	Program Trace overrun
00010	Data Trace overrrun
00011	Read/write access error
00101	Invalid access opcode (Nexus Register unimplemented)
00110	Watchpoint overrun
00111	(Program Trace or Data Trace) & Ownership Trace overrun
01000	(Program Trace or Data Trace or Ownership Trace) & Watchpoint overrun
01001–10111	Reserved
11000	BTM lost due to collision w/ higher priority message
11001–11111	Reserved

Table 11-3. Watchpoint Source Encoding (TCODE = 15)

Watchpoint Code	Description
xxxx xxx1	ARM7 Watchpoint 0
xxxx xx1x	ARM7 Watchpoint 1
xxxx x1xx	Nexus Watchpoint 1

Table 11-3. Watchpoint Source Encoding (TCODE = 15) (Continued)

Watchpoint Code	Description
xxxx 1xxx	Nexus Watchpoint 2
xxx1 xxxx	Nexus Watchpoint 3
xx1x xxxx	Nexus Watchpoint 4
x1xx xxxx	Nexus Watchpoint 5
1xxx xxxx	Nexus Watchpoint 6

Table 11-4. Resource Code Encoding (TCODE = 27)

Resource Code	Description
0000	Program Trace Instruction Counter Overflow
0001	Branch / Predicate Instruction History Buffer
0010–1111	Reserved for future functionality

Table 11-5. Event Code Encoding (TCODE = 33)

Event Code	Description
0000	Entry into Debug Mode
0001	Entry into Low Power Mode (CPU only)
0010–0011	Reserved for future functionality
0100	Program Trace Disable
0101–1111	Reserved for future functionality

Table 11-6. Data Trace Size (DSZ) Encodings (TCODE = 5, 6, 13, 14)

DTM Size Encoding	Transfer Size
000	8-bit
001	16-bit
010	32-bit
011–111	Reserved

11.2 Nexus Protocol

For a complete description of the Nexus protocol, please refer to the IEEE Nexus specification.

11.3 Nexus Implementation

Nexus 3 was chosen over Nexus 2 because of the additional requirement for Data Trace capabilities. The Nexus module, part of a standard supported by Freescale, has the following advantages.

- No cost
- Already implemented for the ARM9 core
- Supported by Metroworks tools

11.3.1 Nexus Port Replacement

The selection between the Primary and Secondary Nexus ports will be done via an external pin (See Section 2.2, “MCU Hardware Configuration Summary”) latched at RESET in order to alleviate the need for the development system to write into a memory-mapped register. The Primary and Secondary ports are defined as follows:

Table 11-7. Nexus Port Assignments

Nexus Pin	Primary Port	Secondary Port
EVTI	PA9	PC3
MCKO	PB0	PC4
EVTO	PB1	PC5
MSEO	PB2	PC6
RDY	PB3	PC7
MDO[0]	PG2	PC8
MDO[1]	PG3	PC9
MDO[2]	PG4	PC10
MDO[3]	PB9	PC11
MDO[4]	PB11	PC12
MDO[5]	PB13	PC13
MDO[6]	PB14	PC14
MDO[7]	PB15	PC15

11.3.2 TAP Controller Encodings

The following table lists the JTAG TAP controller IR register encodings used on the MAC72xx.

Table 11-8. JTAG TAP Controller IR Register Encodings

IR Encoding	Function	Used By
$0 = 0000	EXTEST	ARM9
$1 = 0001	Unused	
$2 = 0010	SCAN_N	ARM7/ARM9
$3 = 0011	SAMPLE/PRELOAD	ARM9
$4 = 0100	RESTART	ARM7/ARM9
$5 = 0101	CLAMP	ARM920T

Table 11-8. JTAG TAP Controller IR Register Encodings (Continued)

IR Encoding	Function	Used By
$6 = 0110	Unused	
$7 = 0111	HIGHZ	ARM920T
$8 = 1000	NEXUS_ACCESS	Nexus
$9 = 1001	CLAMPZ	ARM920T
$A = 1010	Unused	
$B = 1011	Unused	
$C = 1100	INTEST	ARM9[1]
$D = 1101	Unused	
$E = 1110	IDCODE	ARM7/ARM9
$F = 1111	BYPASS	ARM7/ARM9

1. The INTEST IR encoding is used to load/unload values into the six configuration registers (“Scan Chains”) built into the MAC72xx:

Bit	Definition
0	Not Used
1–2	ARM7/EICE
3	Not Used
4	PTI Scan Chain 4
5	PTI Scan Chain 5
6	PTI Scan Chain 6
7	PTI Scan Chain 7

12. The IDCODE returned on the MAC72xx is hard-coded in the ARM7 core and is $4f1f0f0f.

11.4 Nexus Integration

On the MAC72xx, the following Nexus Low Power State (LPS) encodings are used:

Table 11-9. Nexus LPS Encodings

Mode	ext_lps
Normal	ext_lps[2:0] = 3’b000
Stop	ext_lps[0] = 1
Doze	ext_lps[1] = 1
Debug	ext_lps[2] = 1

NOTE

On the MAC72xx, STOP mode is not implemented, and ext_lps[0] will always read back as a ‘0’.

11.4.1 Nexus Integration and SoC Security

In order to prevent access to internal resources, such as the Flash, when they are secured, all debug features (i.e.-EICE and Nexus) are disabled when the chip is in a secured mode, denoted by the negation of the **debug_enabled** signal. Please refer to Chapter 9, “Device Memory Map” for more details on this. When **debug_enabled** is negated, the following actions are taken:

- The **DBGEN** signal to the ARM7 core is negated.
- All clocks to the Nexus are gated off. See the section “Power Consumption” below for more details.

11.4.2 Nexus Integration and FlexBus Port Sizing

Because the Nexus Secondary Port is muxed in with the address bus of the external bus, it is impossible to use the FlexBus when the Nexus is active, and the Secondary Port is selected. It is only possible to use the Nexus on the Primary Port with an 8-bit external bus.

Table 11-10. FlexBus Port Sizing with Nexus

Nexus Port Select (PF0)	Nexus Present (PF1)	Allowed FlexBus Port Sizes (PF3)
0 (Primary)	1 (Yes)	8-bit (PF3=0)
0 (Primary)	0 (No)	8/16-bit (PF3=0/1)
1 (Secondary)	1 (Yes)	FlexBus not available
1 (Secondary)	0 (No)	8/16-bit (PF3=0/1)

11.4.3 Nexus Integration and Port Control

Unlike other “peripherals” in the system, which require programming of the Port Integration Module (PIM) in order to give control of the pins to the peripheral, the Nexus will automatically be given control of the appropriate ports once enabled (by setting of Nexus Present and Nexus Port Select hardware configuration pins). This is done in order to avoid requiring the development system to write internal memory-mapped registers. On the MAC71xx family, the number of MSEO pins selected was done via an input port on the Nexus module (**ext_mseo_sel**) (0 = 1-pin, 1 = 2-pin). For all MAC71xx devices, this was tied to 0, denoting a single $\overline{\text{MSEO}}$ pin was available. On the MAC72xx, this selection is done via the MSC bit in the Port Control Register (PCR) in the Nexus (0 = 1-pin, 1 = 2-pin). On the MAC72xx, a single $\overline{\text{MSEO}}$ pin is available. Therefore, the software must not set this bit in the PCR register.

11.5 Nexus External Pins

The external pin interface is flexible. The minimum number of pins necessary is 4 pins for a basic Program Trace implementation, or a maximum of 13 for a full Program + Data Trace implementation. All pins total must be made available on either a Primary or Secondary Nexus Port

Table 11-11. Nexus External Pins

Signal	Description
MDO[7:0]	8-bit Message Data Out (2 required, 6 optional)
$\overline{\text{MSEO}}$	Message Start/End Out
$\overline{\text{EVTI}}$	Event In
$\overline{\text{EVTO}}$	Event Out
MCKO	Message Clock Out
$\overline{\text{RDY}}$	Ready (Output)

11.5.1 MDO - Message Data (Output)

These pins provide the packet data for the trace messages. The Nexus module can be programmed to use a minimum number of MDO pins (2) or a maximum number of MDO pins (8). Following are the minimum requirements for the number of MDO pins:

- Program Trace only (Class 2): 2 MDO pins. As a benchmark, the EEMBC suite of automotive benchmarks was run on a M*CORE Nexus3 module using the 2-pin MDO option (only turning on Program Trace), at 100MHz without buffer overflow.
- Program and Data Trace (Class 3): 8 MDO pins

11.5.2 MSEO - Message Start/End (Active low output)

These pins indicate to the development tool when each packet within a message begins and ends. The Nexus module supports two of these pins for higher throughput (bandwidth), but only one is necessary for IEEE compliance. For a Program and Data Trace only module running @ 80MHz, only 1 MSEO pin is necessary.

11.5.3 EVTI - Event In (Active low input)

This required pin is multi-faceted. Initially, it is used to enable Nexus (sampled during a Debug Reset). It is also programmable to create a “synchronization” type message (giving the full address if the tool is lost), or initiate a debug request to the ARM core.

11.5.4 EVTO - Event Out (Active low output)

This optional pin can be used by the development tool to trigger on certain events. EVTO can be programmed to assert based on the occurrence of a watchpoint (defined within the ARM CPU), or upon the entry into Debug Mode.

11.5.5 RDY - DMA Ready (Active low output)

This pin indicates to the development tool that a DMA access has completed successfully.

11.5.6 MCKO - Message Clock (Output)

This pin provides the clock reference for all Nexus inputs and outputs.

11.6 Nexus Bus Aborts

The Nexus is not capable of producing any bus aborts. Any Nexus bus accesses that generate aborts are treated as if the accesses were initiated by the ARM7 core.

11.7 Nexus Differences from MAC71xx

- Changed from Nexus2p to Nexus3p (Includes data trace)
- Added four “address match only” breakpoints (pending approval)
- Added two data watchpoints
- Changed from two MDO pins to eight MDO pins
- Number of $\overline{\text{MSEO}}$ pins is no longer hard-coded in the design, but now selectable with the MSC bit in the Nexus PCR register

11.8 Nexus Application Usage

11.8.1 Nexus Configuration

Enabling and selection of the Nexus port is via the value of the Port F0/F1 pins at Reset (See Table 11-12). On some packages, only one Nexus Port may be available. Once a Nexus Port is enabled and selected, the Nexus itself must still be enabled. Please refer to the Nexus 3 Block Guide or the Nexus web site www.ieee-isto.org/Nexus5001 for more information.

Table 11-12. Nexus Configuration

PF0	PF1	Description
0	0	Nexus Ports are disabled (Ports are used for "normal" functionality)
0	1	Nexus Primary Port is enabled
1	0	Nexus Ports are disabled (Ports are used for "normal" functionality)
1	1	Nexus Secondary Port is enabled

On the MAC71xx family, the number of MSEO pins selected was done via an input port on the Nexus module (**ext_mseo_sel**) (0 = 1-pin, 1 = 2-pin). For all MAC71xx devices, this was tied to 0, denoting a single $\overline{\text{MSEO}}$ pin was available. On the MAC72xx, this selection is done via the MSC bit in the Port

Control Register (PCR) in the Nexus (0 = 1-pin, 1 = 2-pin). On the MAC72xx, a single $\overline{\text{MSEO}}$ pin is available. Therefore, the software must not set this bit in the PCR register.

11.8.2 Programming the PCR Register

In order to match the hardware implementation of the Nexus3p on the MAC72xx, the following **PCR** register bits must always be programmed as indicated:

PCR[31:30]	11
PCR[25]	0

11.8.3 Resetting Nexus

Resetting the Nexus3p is done as described in the Nexus Block Guide, by taking the JTAG TAP State Machine into the Test-Logic-Reset state. There is no minimum required time that this state has to be held. Please also refer to Chapter 6, “Resets” for information on resetting Nexus.

11.8.4 Enabling Nexus

In order to enable Nexus, you must perform one of the following methods:

Method #1

1. At reset, drive the PF1 pin high to enable a Nexus port
2. At reset, drive the PF0 pin low or high to select which Nexus port to use
3. Drive the JTAG TAP State Machine into the Test-Logic-Reset state
4. Drive the $\overline{\textbf{EVTI}}$ pin low (asserted)
5. Drive the JTAG TAP State Machine out of the Test-Logic-Reset state (into the Run-Test/Idle state for instance)

Method #2

1. At reset, drive the PF1 pin high to enable a Nexus port
2. At reset, drive the PF0 pin low or high to select which Nexus port to use
3. Load the NEXUS_ACCESS instruction into the JTAG TAP State Machine

Once the Nexus module has been enabled, it will remain enabled until entry into the JTAG Test-Logic-Reset state or a Power On Reset is performed.

11.8.5 Disabling Nexus

In order to disable Nexus, you must perform the following:

1. At reset, drive the PF1 pin low to disable Nexus ports. The value driven on PF0 does not matter in this case.
2. If a Power On Reset has been performed, and the Nexus was not enabled, then no further action is necessary.

3. Drive the JTAG TAP State Machine into the Test-Logic-Reset state. While the State Machine is in this state, all Nexus output pins are driven to their off value
4. Drive the **EVTI** pin high (negated)
5. Drive the JTAG TAP State Machine out of the Test-Logic-Reset state (into the Run-Test/Idle state for instance)

Note that the Nexus is always in a disabled state after a Power On Reset. Once the Nexus module has been disabled, it will remain disabled until explicitly enabled.

When the Nexus is disabled the Nexus outputs have the following off values:

EVTO	1
MSEO	1
RDY	1
MCKO	0
MDO[n]	0

In the disabled state, all Nexus3 registers are not accessible for reads or writes. In addition, the **EVTI** input is ignored, except when exiting the Test-Logic-Reset state, as described in Section 11.8.3, “Resetting Nexus.

11.8.6 Nexus Development Status (DS) Register

The DS implementation on the MAC72xx is identical to the MAC71xx, except that the MAC72xx does not support STOP mode, so the DS[28] bit should always read back 0.

The DS[29] is for DOZE mode and the DS[30] is for Debug mode, which on the MAC72xx is identical to the ARM7 Debug Mode (DS[31]).

The DS[27:26] bits are tied low (0), and DS[25:24] are unused and read back 0b00. This gives the following possible combinations for DS[31:24]:

$00	Entering Normal Mode
$20	Entering Doze mode from Normal Mode
$C0	Entering Debug Mode from Normal Mode
$E0	Entering Debug Mode from Doze Mode or Doze Mode from Debug Mode

11.8.7 Unintended Activation of Nexus

If the Nexus is not intended to be used, the PF1 pin should be driven low at reset. This will disconnect all Nexus outputs, and force the **EVTI** input to the Nexus high (disabled). It is still possible to activate the Nexus through the JTAG interface in this case. However, there will be no impact to the value of the pins on either Nexus port.

If, however, the Nexus port is accidentally enabled (by shorting the PF1 pin to 5V for instance), then either the Primary or Secondary Nexus port will be active. If the Nexus registers have not been programmed since the last Power On Reset, then the Nexus will come up disabled, and the Nexus pins will have the functionality as described in Section 11.8.5, “Disabling Nexus. In order for the Nexus port to change values, the Nexus must be enabled, as described in Section 11.8.3, “Resetting Nexus.

Note that, if the Nexus is enabled, simply setting the **TM** field in the Nexus **DC** register to 000 (No Trace) will not cause the **MDO[7:0]** pins to become inactive. With the Nexus protocol, there are various non-trace messages that may be issued.

In order for the **MCKO** output to toggle, the **PCR[29]** bit must be set.

If the software wishes to check the status of the Nexus port during boot (highly recommended), then the **NEXUS** field of the **STATUS** register in the SSM module may be read. Note that **STATUS[12]** reflects the latched value of the PF1 pin, and **STATUS[11]** reflects the latched value of the PF0 pin.

11.9 External Signal Description

This section details information regarding the A7S Nexus3 pins and pin protocol.

11.9.1 Functional Description

The A7S Nexus3 pin interface provides the function of transmitting messages from the messages queues to the external tools. It is also responsible for handshaking with the message queues.

11.9.2 Pins Implemented

The A7S Nexus3 module implements one (1) $\overline{\text{EVTI}}$ and one (1) or two (2) $\overline{\text{MSEO}}$. It also implements up to eight (8) MDO pins, (1) $\overline{\text{RDY}}$ pin, (1) $\overline{\text{EVTO}}$ pin, and one (1) clock output pin (MCKO). The output pins are synchronized to the Nexus3 output clock (MCKO).

All Nexus3 input functionality is controlled through the JTAG port in compliance with **IEEE 1149.1** (see Section 11.10.4, “Nexus Register Access via JTAG,” for details). The JTAG pins are incorporated as I/O to the ARM7 processor.

Table 11-13. JTAG Pins for A7S Nexus3

JTAG Pins	Input/ Output	Description of JTAG Pins (included in ARM7 CPU)
TDO	O	The Test Data Output (TDO) pin is the serial output for test instructions and data. TDO is three-stateable and is actively driven in the “shift-IR” and “shift-DR” controller states. TDO changes on the falling edge of TCK.
$\overline{\text{TDI}}$	I	The Test Data Input ($\overline{\text{TDI}}$) pin receives serial test instruction and data. $\overline{\text{TDI}}$ is sampled on the rising edge of TCK and has an internal pull-up resistor.
TMS	I	The Test Mode Select (TMS) input pin is used to sequence the JTAG test controllers’ state machine. TMS is sampled on the rising edge of TCK and has an internal pull-up resistor.

Table 11-13. JTAG Pins for A7S Nexus3 (Continued)

JTAG Pins	Input/ Output	Description of JTAG Pins (included in ARM7 CPU)
TCK	I	The Test Clock (TCK) input pin is used to synchronize the test logic, and control register access through the JTAG port.
TRST	I	The optional Test Reset ($\overline{\text{TRST}}$) input pin is used to asynchronously initialize the JTAG controller. The $\overline{\text{TRST}}$ pin has an internal pull-up resistor.

Table 11-14. A7S Nexus3 Auxiliary Pins

Auxiliary Pins	Input/ Output	Description of Auxiliary Pins
MCKO	O	Message Clock Out (MCKO) is a Nexus generated output clock to development tools for timing of MDO & $\overline{\text{MSEO}}$ pin functions. MCKO is programmable through the DC Register.
MDO[N:0]	O	Message Data Out (MDO[N:0]) are output pins used for OTM and BTM. External latching of MDO shall occur on rising edge of the Nexus3 message clock (MCKO).
$\overline{\text{MSEO}}$[1:0]	O	Message Start/End Out ($\overline{\text{MSEO}}$) are output pins which indicate when a message on the MDO pins has started, when a variable length packet has ended, and when the message has ended. External latching of $\overline{\text{MSEO}}$ shall occur on rising edge of the Nexus3 clock (MCKO). One or two pin $\overline{\text{MSEO}}$ functionality is determined at integration time.
RDY	O	Ready ($\overline{\text{RDY}}$) is an output pin used to indicate to the external tool that the Nexus block is ready for the next Read/Write Access. If Nexus is enabled, this signal is asserted upon successful (without error) completion of an AHB transfer (Nexus read or write) & is held asserted until the JTAG state machine reaches the “Capture_DR” state. Upon exit from system reset or if Nexus is disabled, $\overline{\text{RDY}}$ remains de-asserted.
EVTO	O	Event Out ($\overline{\text{EVTO}}$) is an output which, when asserted, indicates one of two events has occurred based on the EOC bits in the DC Register. $\overline{\text{EVTO}}$ is held asserted for one (1) cycle of MCKO: 1) one of four watchpoints has occurred & EOC = 2'b00 2) debug mode was entered (DBGACK from ARM7) & EOC = 2'b01
$\overline{\text{EVTI}}$	I	Event In ($\overline{\text{EVTI}}$) is an input which, when asserted, will initiate one of two events based on the EIC bits in the DC Register (if the Nexus module is enabled at reset; see Section 11.1.3.3, “Disabled”): 1) Program Trace synchronization messages (provided Program Trace is enabled & EIC = 2'b00). 2) Debug request (EDBGRQ) to ARM7 EmbeddedICE module (provided EIC = 2'b01 and this feature is implemented).

11.9.3 Pin Protocol

The protocol for the ARM7 processor transmitting messages via the auxiliary pins shall be accomplished with the $\overline{\text{MSEO}}$ pin(s) function (outlined in Table 11-15).

$\overline{\text{MSEO}}$ is used to signal the end of variable-length packets, and not fixed length packets. $\overline{\text{MSEO}}$ is sampled on the rising edge of the message clock (MCKO).

Table 11-15. $\overline{\text{MSEO}}$ Pin(s) Protocol

$\overline{\text{MSEO}}$ Function	Single $\overline{\text{MSEO}}$ data (serial)	Dual $\overline{\text{MSEO}}$ data
Start of message	1–1–0	11–00
End of message	0–1–1–(more 1's)	00 (or 01)–11–(more 11's)
End of variable length packet	0–1–0	00–01
Message transmission	0's	00's
Idle (no message)	1's	11's

Figure 11-2 illustrates the state diagram for single pin $\overline{\text{MSEO}}$ transfers.

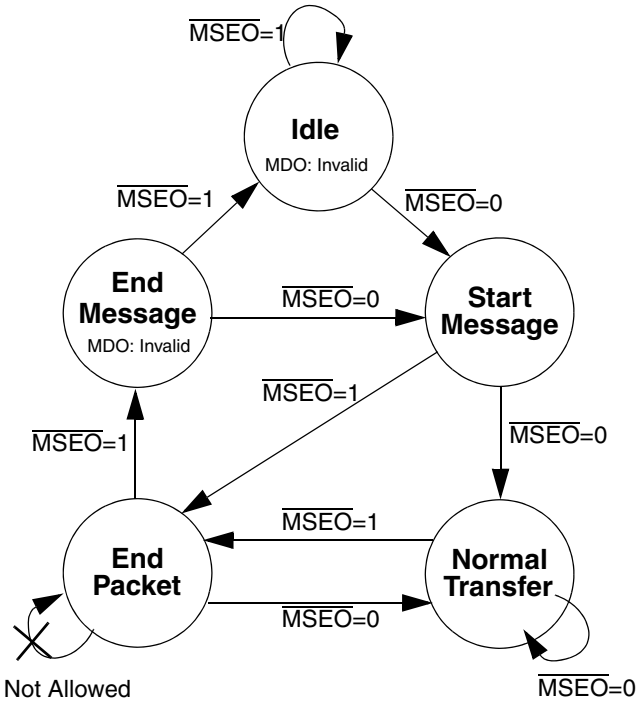


Figure 11-2. Single Pin $\overline{\text{MSEO}}$ Transfers

Note that the "End Message" state does not contain valid data on the MDO pins. Also, It is not possible to have two consecutive "End Packet" messages. This implies the minimum packet size for a variable length packet is 2x the number of MDO pins. This ensures that a false end of message state is not entered by emitting two consecutive 1's on the $\overline{\text{MSEO}}$ pin before the actual end of message.

Figure 11-3 illustrates the state diagram for two pin $\overline{\text{MSEO}}$ transfers.

The two-pin $\overline{\text{MSEO}}$ option is more efficient that the single pin option. Termination of the current message may immediately be followed by the start of the next message on the consecutive clocks. An extra clock to end the message is not necessary as with the one $\overline{\text{MSEO}}$ pin option. The two-pin option also allows for consecutive "End Packet" states. This can be an advantage when small, variable sized packets are transferred.

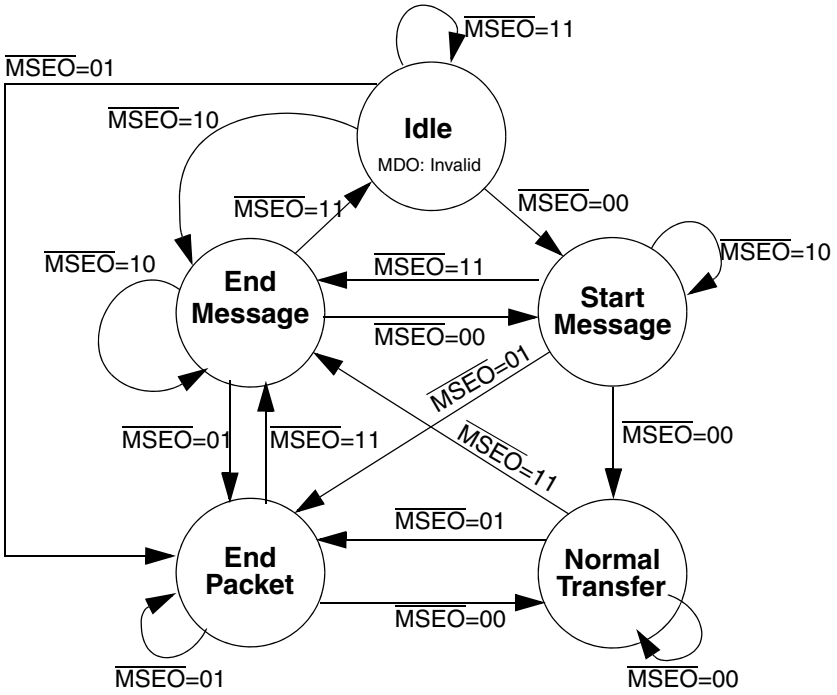


Figure 11-3. Two Pin $\overline{\text{MSEO}}$ Transfers

NOTE

The “End Message” state may also indicate the end of a variable-length packet as well as the end of the message when using the two-pin option.

11.9.4 Rules for Output Messages

ARM7 based Class 3 compliant embedded processors must provide messages via the auxiliary port in a consistent manner as described below:

- A variable-sized packet within a message must end on a port boundary.
- Whenever a variable-length packet is sized such that it does not end on a port boundary, it is necessary to extend and zero fill the remaining bits after the highest-order bit so that it can end on a port boundary.

  For example, if the MDO port is 2 bits wide, and the unique portion of an indirect branch address is 5 bits, then the remaining 1 bit of MDO must be packed with a 0.
- A variable-sized packet may start within a port boundary only when following a fixed length packet. (If two variable-sized packets end and start on the same clock, it is impossible to know which bits are from the last packet and which bits are from the next packet.

11.9.5 Examples

The following are examples of Program Trace Messages.

Table 11-16 illustrates an example Indirect Branch Message (traditional, Thumb mode) with 2 MDO / 1 $\overline{\text{MSEO}}$ configuration. Table 11-17 illustrates the same example with the 8 MDO / 2 $\overline{\text{MSEO}}$ configuration.

Note that T0 and S0 are the least significant bits where:

- Tx = TCODE number (fixed)
- Sx = Source processor (fixed)
- Ix = Number of instructions (variable)
- Ax = Unique portion of the address (variable)

Note that during clock 13, the MDO pins are ignored in the single $\overline{\text{MSEO}}$ case.

Table 11-16. Indirect Branch Message Example (2 MDO / 1 $\overline{\text{MSEO}}$)

Clock	MDO[1:0]			$\overline{\text{MSEO}}$	State
0	X	X		1	Idle (or end of last message)
1	T1	T0		0	Start Message
2	T3	T2		0	Normal Transfer
3	T5	T4		0	Normal Transfer
4	S1	S0		0	Normal Transfer
5	S3	S2		0	Normal Transfer
6	I1	I0		0	Normal Transfer
7	I3	I2		0	Normal Transfer
8	I5	I4		0	End Packet
9	A1	A0		0	Normal Transfer
10	A3	A2		0	Normal Transfer
11	A5	A4		0	Normal Transfer
12	A7	A6		1	End Packet
13	0	0		1	End Message
14	T1	T0		0	Start Message

Table 11-17. Indirect Branch Message Example (8 MDO / 2 $\overline{\text{MSEO}}$)

Clock	MDO[7:0]									$\overline{\text{MSEO}}$[1:0]		State
0	X	X	X	X	X	X	X	X		1	1	Idle (or end of last message)
1	S1	S0	T5	T4	T3	T2	T1	T0		0	0	Start Message
2	I5	I4	I3	I2	I1	I0	S3	S2		0	1	End Packet
3	A7	A6	A5	A4	A3	A2	A1	A0		1	1	End Packet/End Message
4	S1	S0	T5	T4	T3	T2	T1	T0		0	0	Start Message

Table 11-18 and Table 11-19 illustrate examples of Direct Branch Messages: one with 2 MDO / 1 $\overline{\text{MSEO}}$, and one with 8 MDO / 2 $\overline{\text{MSEO}}$.

Note that T0 and I0 are the least significant bits where:

- Tx = TCODE number (fixed)
- Sx = Source processor (fixed)
- Ix = Number of Instructions (variable)

Table 11-18. Direct Branch Message Example (2 MDO / 1 $\overline{\text{MSEO}}$)

Clock	MDO[1:0]			$\overline{\text{MSEO}}$	State
0	X	X		1	Idle (or end of last message)
1	T1	T0		0	Start Message
2	T3	T2		0	Normal Transfer
3	T5	T4		0	Normal Transfer
4	S1	S0		0	Normal Transfer
5	S3	S2		0	Normal Transfer
6	I1	I0		1	End Packet
7	0	0		1	End Message
8	T1	T0		0	Start Message

Table 11-19. Direct Branch Message Example (8 MDO / 2 $\overline{\text{MSEO}}$)

Clock	MDO[7:0]									$\overline{\text{MSEO}}$[1:0]		State
0	X	X	X	X	X	X	X	X		1	1	Idle (or end of last message)
1	S1	S0	T5	T4	T3	T2	T1	T0		0	0	Start Message
2	0	0	0	0	I1	I0	S3	S2		1	1	End Packet/End Message
3	S1	S0	T5	T4	T3	T2	T1	T0		0	0	Start Message

11.10 A7S Nexus3 Programmers Model

This section describes the A7S Nexus3 programmers model. Nexus registers are accessed using the JTAG port in compliance with IEEE 1149.1 (discussed in Section 11.10.4, “Nexus Register Access via JTAG”).

11.10.1 JTAG ID Register

This JTAG ID Register (located within the ARM7 CPU) provides key development attributes to the development tool concerning the ARM7 processor and the A7S Nexus3 block. This register is fixed for each ARM7 embedded system.

This register is accessed through the standard JTAG IR/DR paths.

Offset SoC defined — Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	VERSION				PART NUMBER											
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PART NUMBER				MANUFACTURER ID #											1
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-4. JTAG ID Register

Table 11-20. JTAG ID Field Descriptions

Field	Description
31–28 VER	Embedded Product Version Number X SoC defined
27–12 PNUM	ARM7 Based Part Number M SoC defined
11–1 MANID	Manufacturer ID Number 00E Freescale
0	Fixed per IEEE 1149.1 (JTAG)

11.10.2 Nexus3 Register Map

Table 11-21. A7S Nexus3 Register Map

Nexus Register	Nexus Access Opcode	Read/ Write	Read Address	Write Address
Client Select Control (CSC) (see Note below)	0x1	R	0x02	—
Development Control (DC)	0x2	R/W	0x04	0x05
Development Status (DS)	0x4	R	0x08	—
User Base Address (UBA)	0x6	R/W	0x0C	0x0D
Read/Write Access Control/Status (RWCS)	0x7	R/W	0x0E	0x0F
Read/Write Access Address (RWA)	0x9	R/W	0x12	0x13
Read/Write Access Data (RWD)	0xA	R/W	0x14	0x15
Watchpoint Trigger (WT)	0xB	R/W	0x16	0x17
Data Trace Control (DTC)	0xD	R/W	0x1A	0x1B
Data Trace Start Address1 (DTSA1)	0xE	R/W	0x1C	0x1D
Data Trace Start Address2 (DTSA2)	0xF	R/W	0x1E	0x1F
Data Trace End Address1 (DTEA1)	0x12	R/W	0x24	0x25
Data Trace End Address2 (DTEA2)	0x13	R/W	0x26	0x27
Break/Watchpoint Control 1 (BWC1)	0x16	R/W	0x2C	0x2D
Break/Watchpoint Control 2 (BWC2)	0x17	R/W	0x2E	0x2F
Break/Watchpoint Control 3 (BWC3)	0x18	R/W	0x30	0x31
Break/Watchpoint Control 4 (BWC4)	0x19	R/W	0x32	0x33
Break/Watchpoint Control 5 (BWC5)	0x1A	R/W	0x34	0x35
Break/Watchpoint Control 6 (BWC6)	0x1B	R/W	0x36	0x37
Break/Watchpoint Address 1 (BWA1)	0x1E	R/W	0x3C	0x3D
Break/Watchpoint Address 2 (BWA2)	0x1F	R/W	0x3E	0x3F
Break/Watchpoint Address 3 (BWA3)	0x20	R/W	0x40	0x41
Break/Watchpoint Address 4 (BWA4)	0x21	R/W	0x42	0x43
Break/Watchpoint Address 5 (BWA5)	0x22	R/W	0x44	0x45
Break/Watchpoint Address 6 (BWA6)	0x23	R/W	0x46	0x47
Break/Watchpoint Data 1 (BWD1)	0x26	R/W	0x4C	0x4D
Break/Watchpoint Data 2 (BWD2)	0x27	R/W	0x4E	0x4F
Break/Watchpoint Address Mask 1 (BWAM1)	0x42	R/W	0x84	0x85
Break/Watchpoint Address Mask 2 (BWAM2)	0x43	R/W	0x86	0x87
Break/Watchpoint Data Mask 1 (BWDM1)	0x44	R/W	0x88	0x89
Break/Watchpoint Data Mask 2 (BWDM2)	0x45	R/W	0x8A	0x8B
Port Configuration Register (PCR)	0x7F	R/W	0xFE	0xFF

NOTE

In a multi-nexus environment, the CSC and PCR may be implemented in a separate module at the SoC level. If implemented, the values at the SoC level will override the settings in the Nexus module.

11.10.3 A7S Nexus3 Register Definitions

Detailed register definitions are as follows:

11.10.3.1 Client Select Control (CSC)

The Client Select Control Register determines which Nexus client is under development.

Offset see Table 11-21 Access: User read/write

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

Figure 11-5. Client Select Control Register (CSC)

Table 11-22. Client Select Register Field Description

Field	Description
7–0 CSC	Client Select Control 0xXX Nexus client (SoC level)

11.10.3.2 Development Control (DC)

The Development Control Register is used to control basic development features of A7S Nexus3.

Offset see Table 11-21 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	EOC		0	PTM	WEN	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	OVC			EIC		TM		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-6. Development Control Register (DC)

Table 11-23. DC Field Descriptions

Field	Description
31–29	Reserved for future functionality (read as 0)
28–27 EOC	EVTO Control 00 EVTO upon occurrence of Watchpoint 01 EVTO upon entry into Debug Mode (DBGACK) 1X Reserved
26	Reserved for future functionality (read as 0)
25 PTM	Program Trace Method (Thumb mode only) 0 Program Trace in Thumb mode uses Branch History Messages 1 Program Trace in Thumb mode uses traditional Branch Messages
24 WEN	Watchpoint Trace Enable 0 Watchpoint Messaging disabled 1 Watchpoint Messaging enabled
23–8	Reserved for future functionality (read as 0)
7–5 OVC	Overrun Control 000 Generate overrun messages 001 Reserved 010 Reserved 011 Delay processor for trace message overruns (FIFOFULL) 1xx Reserved
4–3 EIC	EVTI Control 00 EVTI for synchronization (Program Trace) 01 EVTI for Debug request (if implemented) 10 EVTI disabled 11 Reserved
2–0 TM	Trace Mode 000 No Trace 1xx Program Trace enabled x1x Data Trace enabled xx1 Ownership Trace enabled

11.10.3.3 Development Status (DS)

The Development Status Register is used to report system debug status. When Debug Mode is entered or exited or an SoC or ARM7 defined Low Power Mode is entered (see Note below), a Debug Status Message is transmitted with DS[31:24]. The external tool can read this register at any time.

Offset see Table 11-21 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	DBG	LPS			LPC		0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-7. Development Status Register (DS)

Table 11-24. DS Field Descriptions

Field	Description
31 DBG	ARM7 CPU Debug Mode Status 0 CPU not in Debug Mode 1 CPU in Debug Mode (DBGACK asserted)
30–28 LPS	ARM7 System Low Power Mode Status 000 Normal (Run) mode *nnn* System Low Power Status (SoC level)
27–26 LPC	ARM7 CPU Low Power Mode Status 00 Normal (Run) mode 01 CPU in powered-down state 1x Reserved
25–0	Reserved for future functionality (read as 0)

NOTE

System low power mode bits (LPS) functionality is determined at the platform or SoC integration level. Any entry into a system-level low power mode (LPS != 3'b000) will trigger a Debug Status Message sending the DS register value to the external tool. These bits are recommended for non-CPU related power-down modes. It is the tool's responsibility to decode the specific type of low power mode based on the encodings for the specific SoC.

The CPU Low Power Mode bits (LPC) must be tied to the logic which indicates that the ARM7 processor is in a powered-down state. The CPU low power state may be independent of the SoC defined low power mode(s) determined by the LPS bits.

11.10.3.4 User Base Address (UBA)

For ARM7 processors, Ownership Trace Messaging is implemented using the Nexus defined User Base Address Register. The User Base Address Register defines the memory mapped base address for the

Ownership Trace Register (OTR). The operating system writes the ID for the current task/process in the OTR. The UBA is read and written to by the external development tool.

Offset see Table 11-21 — Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	OWNERSHIP TRACE REGISTER ADDRESS															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	OWNERSHIP TRACE REGISTER ADDRESS															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-8. User Base Address Register (UBA)

NOTE

It is recommended that the UBA only be modified while system reset is asserted or in debug mode. Caution should be taken when modifying the UBA when system reset is negated.

11.10.3.5 Read/Write Access Control/Status (RWCS)

The Read Write Access Control/Status Register provides control for Read/Write Access. Read/Write access provides DMA-like access to Advanced High-performance Bus (AHB) memory mapped resources when the processor is halted or during runtime. The RWCS Register also provides Read/Write Access Status information per Table 11-26.

Offset see Table 11-21 — Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	AC	RW	SZ			MAP			PR		0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	CNT														ERR	DV
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-9. Read/Write Access Control Register (RWCS)

Table 11-25. RWCS Field Descriptions

Field	Description
31 AC	Access Control 0 End access 1 Start access
30 RW	Read/Write Select 0 Read access 1 Write access
29–27 SZ	Word Size 000 8-bit (byte) 001 16-bit (halfword) 010 32-bit (word) 011–111Reserved (default to word)
26–24 MAP	Map Select 000 Primary memory map 001–111Reserved
23–22 PR	Read/Write Access Priority 00 Lowest access priority 01 Reserved (default to lowest priority) 10 Reserved (default to lowest priority) 11 Highest access priority
21–16	Reserved for future functionality
15–2 CNT	Access Control Count hhhh Number of accesses of word size SZ
1 ERR	Read/Write Access Error (see Table 11-26)
0 DV	Read/Write Access Data Valid (see Table 11-26)

Table 11-26. Read/Write Access Status Bit Encoding

Read Action	Write Action	ERR	DV
Read Access has not completed	Write Access completed without error	0	0
Read Access error has occurred	Write Access error has occurred	1	0
Read Access completed without error	Write Access has not completed	0	1
Not Allowed	Not allowed	1	1

11.10.3.6 Read/Write Access Data (RWD)

The Read/Write Access Data Register provides the data to/from AHB memory mapped locations when initiating a read or a write access.

Offset see Table 11-21 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	READ/WRITE DATA															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	READ/WRITE DATA															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-10. Read/Write Access Data Register (RWD)

11.10.3.7 Read/Write Access Address (RWA)

The Read/Write Access Address Register provides the AHB memory mapped address to be accessed when initiating a read or a write access

Offset see Table 11-21 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	READ/WRITE ADDRESS															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	READ/WRITE ADDRESS															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-11. Read/Write Access Address Register (RWA)

11.10.3.8 Watchpoint Trigger (WT)

The Watchpoint Trigger Register allows the four watchpoints to trigger actions. These watchpoints can control Program Trace and Data Trace. The WT bits can be used to produce an address related “window” for triggering Trace Messages.

Offset see Table 11-21 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	PTS				PTE				DTS				DTE				0	0	0	0
W																				
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-12. Watchpoint Trigger Register (WT)

Table 11-27. WT Field Descriptions

Field	Description
31–29 PTS	Program Trace Start Control 000 Trigger disabled 001 Use ARM7 Watchpoint 0 010 Use ARM7 Watchpoint 1 011 Use Nexus Watchpoint 1 100 Use Nexus Watchpoint 2 101 Use Nexus Watchpoint 3 110 Use Nexus Watchpoint 4 111 Use Nexus Watchpoint 5
28–26 PTE	Program Trace End Control 000 Trigger disabled 001 Use ARM7 Watchpoint 0 010 Use ARM7 Watchpoint 1 011 Use Nexus Watchpoint 1 100 Use Nexus Watchpoint 2 101 Use Nexus Watchpoint 3 110 Use Nexus Watchpoint 4 111 Use Nexus Watchpoint 5
25–23 DTS	Data Trace Start Control 000 Trigger disabled 001 Use ARM7 Watchpoint 0 010 Use ARM7 Watchpoint 1 011 Use Nexus Watchpoint 1 100 Use Nexus Watchpoint 2 101 Use Nexus Watchpoint 3 110 Use Nexus Watchpoint 4 111 Use Nexus Watchpoint 5
22–20 DTE	Data Trace End Control 000 Trigger disabled 001 Use ARM7 Watchpoint 0 010 Use ARM7 Watchpoint 1 011 Use Nexus Watchpoint 1 100 Use Nexus Watchpoint 2 101 Use Nexus Watchpoint 3 110 Use Nexus Watchpoint 4 111 Use Nexus Watchpoint 5
19–0	Reserved for future functionality (read as 0)

NOTE

The WT bits will control Program/Data Trace only if the TM bits within the Development Control Register (DC) have not already been set to enable Program Trace.

11.10.3.9 Data Trace Control (DTC)

The Data Trace Control Register controls whether DTM Messages are restricted to reads, writes or both for a user programmable address range. There are two Data Trace channels controlled by the DTC for the A7S Nexus3 module.

Offset see Table 11-21 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	RWT1		RWT2		0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	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	RC1	RC2	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-13. Data Trace Control Register (DTC)

Table 11-28. DTC Field Description

Field	Description
31–30 RWT1	Read/Write Trace 1 00 No trace messages generated x1 Enable data read trace 1x Enable data write trace
29–28 RWT2	Read/Write Trace 2 00 No trace messages generated x1 Enable data read trace 1x Enable data write trace
27–8	Reserved for future functionality (read as 0)
7 RC1	Range Control 1 0 Condition trace on address within range 1 Condition trace on address outside of range
6 RC2	Range Control 2 0 Condition trace on address within range 1 Condition trace on address outside of range
5–0	Reserved for future functionality

11.10.3.10 Data Trace Start Address (DTSA1, DTSA2)

The Data Trace Start Address Registers define the start addresses for each trace channel.

Offset see Table 11-21 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	DATA TRACE START ADDRESS															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	DATA TRACE START ADDRESS															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-14. Data Trace Start Address Registers (DTSA1, DTSA2)

11.10.3.11 Data Trace End Address (DTEA1, DTEA2)

The Data Trace End Address Registers define the end addresses for each trace channel.

Offset see Table 11-21 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	DATA TRACE END ADDRESS															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	DATA TRACE END ADDRESS															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-15. Data Trace End Address Registers (DTEA1, DTEA2)

Table 11-29 below illustrates the range that will be selected for Data Trace for various cases of DTSA being less than, greater than, or equal to DTEA.

Table 11-29. Data Trace – Address Range Options

Programmed Values	Range Control Bit Value	Range Selected
DTSA ⇐ DTEA	0	DTSA → ← DTEA
DTSA ⇐ DTEA	1	← DTSA DTEA →
DTSA > DTEA	N/A	Invalid Range – no trace

NOTE

For inside range traces, the trace range is inclusive of the DTSA and DTEA. For outside range traces, the trace range is exclusive of the DTSA and DTEA addresses.

11.10.3.12 Breakpoint / Watchpoint Control (BWC1, BWC2)

Breakpoint/Watchpoint Control Registers 1 and 2 control the generation of Nexus Internal Watchpoints 1 and 2.

Offset see Table 11-21 — Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	BWE		BRW		0	0	0	0	0	0	0	0	0	BWO		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	BWT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-16. Breakpoint/Watchpoint Control Registers (BWC1, BWC2)

Table 11-30. BWC1, BWC2 Field Description

Field	Description
31–30 BWE	Breakpoint/Watchpoint Enable 00 Internal Nexus Watchpoint Unit disabled 01–10Reserved 11 Internal Nexus Watchpoint enabled
29–28 BRW	Breakpoint/Watchpoint Read/Write Select 00 Watchpoint hit on read accesses 01 Watchpoint hit on write accesses 10 Watchpoint on read or write accesses 11 Reserved
27–19	Reserved for future functionality (read as 0)
18–16 BWO	Breakpoint/Watchpoint Operand 000 No Register Compare (same as BWC1[31:30]2'b00) 001 Compare with BWD1 value only 010 Compare with BWA1 value only 011 Compare with BWA1 and BWD1 values 1xx Reserved
15 BWT	Breakpoint/WatchpointType 0 Watchpoint #1 on instruction accesses 1 Watchpoint #1 on data accesses
14–0	Reserved for future functionality (read as 0)

NOTE

Watchpoint units 1 and 2 do not have breakpoint capability.

11.10.3.13 Breakpoint / Watchpoint Control (BWC3-6)

Breakpoint/Watchpoint Control registers 3 - 6 control the generation of Nexus Internal Watchpoints 3-6.

Offset see Table 11-21 — Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	BWE		0	0	0	0	0	0	0	0	0	0	0	0	1	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-17. Breakpoint / Watchpoint Control Registers (BWC3-6)

Table 11-31. BWC3-6 Field Description

Field	Description
31–30 BWE	Breakpoint/Watchpoint Enable 00 Internal Nexus Watchpoint Unit disabled 01 Internal Nexus Breakpoint enabled 10 Reserved 11 Internal Nexus Watchpoint enabled
29–19	Reserved for future functionality (read as 0)
18–16	Reserved for future functionality (read as 010)
15–0	Reserved for future functionality (read as 0)

NOTE

Watchpoint units 3 - 6 only perform instruction address comparisons.

11.10.3.14 Breakpoint / Watchpoint Address (BWA1-6)

The Breakpoint / Watchpoint Address registers are compared with ARM bus addresses in order to generate internal watchpoints.

Offset see Table 11-21 — Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	BREAKPOINT / WATCHPOINT ADDRESS															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	BREAKPOINT / WATCHPOINT ADDRESS															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-18. Breakpoint / Watchpoint Address Registers (BWA1-6)

11.10.3.15 Breakpoint / Watchpoint Address Mask (BWAM1, BWAM2)

The Breakpoint / Watchpoint Address Mask Registers allow the user to mask ARM bus addresses for internal watchpoint units 1 and 2 on a bit-wise granularity.

Offset see Table 11-21 — Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	BREAKPOINT / WATCHPOINT ADDRESS MASK															
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	BREAKPOINT / WATCHPOINT ADDRESS MASK															
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 11-19. Breakpoint / Watchpoint Address Mask Registers (BWAM1, BWAM2)

Table 11-32. BWAM Field Description

Field	Description
31–0	Breakpoint / Watchpoint Address Mask Enable 0 Corresponding BWA bit is masked when generating internal watchpoints 1 Corresponding BWA bit is not masked when generating internal watchpoints

11.10.3.16 Breakpoint / Watchpoint Data (BWD1, BWD2)

The Breakpoint / Watchpoint Data Registers are compared with ARM bus values for internal watchpoint units 1 and 2.

Offset see Table 11-21 — Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	BREAKPOINT / WATCHPOINT DATA															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	BREAKPOINT / WATCHPOINT DATA															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-20. Breakpoint / Watchpoint Data Registers (BWD1, BWD2)

11.10.3.17 Breakpoint / Watchpoint Data Mask (BWDM1, BWDM2)

The Breakpoint / Watchpoint Data Mask Registers allow the user to mask ARM bus data values on a byte-wise granularity for internal watchpoint units 1 and 2.

Offset see Table 11-21 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	BWDME			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

Figure 11-21. Breakpoint / Watchpoint Data Mask Registers (BWDM1, BWDM2)

Table 11-33. BWDM Field Description

Field	Description
31–4	Reserved for future functionality (read as 0)
3–0 BWDME	Breakpoint / Watchpoint Data Mask Enable 0xxx mask data byte (31:24) xxx0 mask data byte (7:0)

11.10.3.18 Port Configuration (PCR)

The Port Configuration Register controls the basic port functions including clock control and auxiliary output port width.

NOTE

If the PCR register exists in a separate arbiter module at the SoC level in a multi-Nexus environment, then the settings at the SoC level override the settings in the A7S Nexus3 module.

Offset see Table 11-21 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	OPC		MCK_EN	MCK_DIV			MSC	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-22. Port Configuration Register (PCR)

Table 11-34. PCR Field Description

Field	Description
31–30 OPC	Output Port Mode Control 00 Reduced Port Mode (2 bits) 01 Reduced Port Mode (2 bits) 10 Reduced Port Mode (2 bits) 11 Full Port Mode (8 bits)
29 MCK_EN	Nexus Message Clock Enable 1 Nexus Message Clock (MCKO) is enabled 0 Nexus Message Clock (MCKO) is disabled
28–26 MCK_DIV	Nexus Message Clock Divide Ratio 000 MCKO is 1x the processor clock freq. 001 MCKO is 1/2x the processor clock freq. 010 Reserved 011 MCKO is 1/4x the processor clock freq. 100–110 Reserved 111 MCKO is 1/8x the processor clock freq.
25 MSC	$\overline{\text{MSEO}}$ Pin Control 0 Select 1 $\overline{\text{MSEO}}$ pin 1 Select 2 $\overline{\text{MSEO}}$ pins
24–0	Reserved for future functionality (read as 0)

NOTE

The PCR must only be modified during system reset, or while the Nexus is otherwise idle or disabled to insure correct output port and output clock functionality

11.10.4 Nexus Register Access via JTAG

Access to Nexus register resources is enabled by loading a single instruction (“*NEXUS-ACCESS*”) into the JTAG Instruction Register (IR). For the A7S Nexus3 block, the JTAG IR value is programmable at the platform or SoC integration level. It can be programmed to any of the non-ARM7 implemented IR values. Table below shows the current ARM7 IR values (as defined by ARM).

Table 11-35. ARM7 JTAG Instructions

IR[3:0]	Usable for NEXUS-ACCESS?	JTAG Instruction
0x0	NO	EXTEST (ARM720T)
0x1	YES	not publicly implemented
0x2	NO	SCAN_N (ARM7TDMI-S)
0x3	NO	SAMPLE/PRELOAD (ARM720T)
0x4	NO	RESTART (ARM7TDMI-S)
0x5	NO	CLAMP (ARM720T)
0x6	YES	not publicly implemented

Table 11-35. ARM7 JTAG Instructions (Continued)

IR[3:0]	Usable for NEXUS-ACCESS?	JTAG Instruction
0x7	NO	HIGHZ (ARM720T)
0x8	YES	recommended for NEXUS-ACCESS
0x9	NO	CLAMPZ (ARM720T)
0xA	YES	not publicly implemented
0xB	YES	not publicly implemented
0xC	NO	INTEST (ARM7TDMI-S)
0xD	YES	not publicly implemented
0xE	NO	IDCODE (ARM7TDMI-S)
0xF	NO	BYPASS (ARM7TDMI-S)

NOTE

It is recommended to use IR[3:0] = 0x8 for “NEXUS-ACCESS”

Once the JTAG “NEXUS-ACCESS” instruction has been loaded, the JTAG port allows tool/target communications with all Nexus registers according to the map in Table 11-21.

Reading/writing of a Nexus register then requires two (2) passes through the Data-Scan (DR) path of the JTAG state machine (see 11.12, “IEEE 1149.1 State Machine and RD/WR Sequences”).

1. The first pass through the DR selects the Nexus register to be accessed by providing an index (see Table 11-21), and the direction (read/write). This is achieved by loading an 8-bit value into the JTAG Data Register (DR). This register has the following format:

Access: User read/write

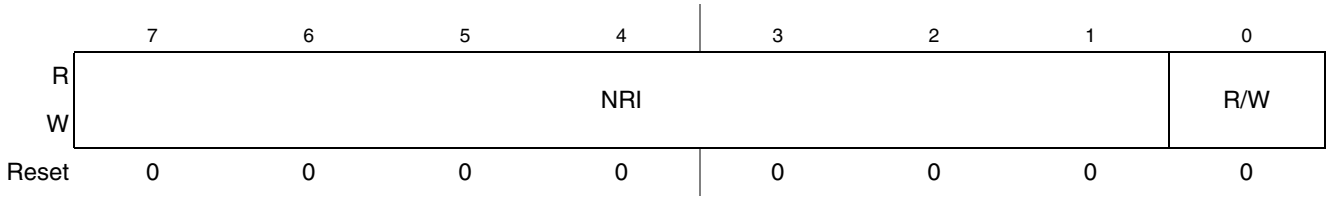


Figure 11-23. JTAG DR for Nexus Register Access

Table 11-36. JTAG DR Field Description for Nexus Register Access

Field	Description
7–1 NRI	Nexus Register Index xx Selected from values in Table 11-21
0 R/W	Read/Write (R/W) 0 Read 1 Write

2. The second pass through the DR then shifts the data in or out of the JTAG port, LSB first.
   a) During a read access, data is latched from the selected Nexus register when the JTAG state machine (see 11.12, “IEEE 1149.1 State Machine and RD/WR Sequences”) passes through the “Capture-DR” state.
   b) During a write access, data is latched into the selected Nexus register when the JTAG state machine (see 11.12, “IEEE 1149.1 State Machine and RD/WR Sequences”) passes through the “Update-DR” state.

11.10.5 Programming Considerations (RESET)

If Nexus3 register configuration is to occur during system reset (as opposed to debug mode), all Nexus3 configuration should be completed between the exit from JTAG Test-Logic-Reset state and system reset de-assertion, after the JTAG ID Register has been read by the tool.

11.11 Functional Description

11.11.1 Ownership Trace

Ownership trace provides a macroscopic view, such as task flow reconstruction, when debugging software written in a high level (or object-oriented) language. It offers the highest level of abstraction for tracking operating system software execution. This is especially useful when the developer is not interested in debugging at lower levels.

11.11.1.1 Ownership Trace Messaging (OTM)

Ownership trace information is messaged via the auxiliary port using an Ownership Trace Message (OTM). The User Base Address Register (UBA), which can be accessed via the JTAG port, contains the address of the Ownership Trace Register (OTR). The OTR is updated by the operating system software to provide task/process ID information.

An OTM is generated when new information is updated in the OTR register by the ARM7 processor, the data is latched within Nexus, and is messaged out via the auxiliary port, allowing development tools to trace ownership flow.

Ownership trace information is messaged out in the following format:

(32 bits)	(4 bits)	(6 bits)
Task / Process ID Tag	SRC	TCODE (000010)

Fixed length = 42 bits

Figure 11-24. Ownership Trace Message Format

11.11.1.2 OTM Error Messages

An Error Message occurs when a new message cannot be queued due to the message queue being full. The FIFO will discard incoming messages until it has completely emptied the queue. Once emptied, an Error

Message will be queued. The error encoding will indicate which type(s) of messages attempted to be queued while the FIFO was being emptied.

If only an OTM Message attempts to enter the queue while it is being emptied, the Error Message will incorporate the OTM only error encoding (00000). If both OTM AND either BTM or DTM messages attempt to enter the queue, the Error Message will incorporate the OTM and Program Trace error encoding (00111). If a Watchpoint also attempts to be queued while the FIFO is being emptied, then the Error Message will incorporate error encoding (01000).

NOTE

The OVC bits within the DC Register can be set to delay the CPU by asserting the FIFOFULL signal in order to alleviate (but not eliminate) potential overruns.

Error information is messaged out in the following format (see Table 11-2).

(5 bits)	(4 bits)	(6 bits)
ERROR (00000 / 00111 / 01000)	SRC	TCODE (001000)

Fixed length = 15 bits

Figure 11-25. Error Message Format

11.11.1.3 OTM Flow

Ownership Trace Messages are generated when the operating system writes to the memory mapped Ownership Trace Register.

The following flow describes the OTM process.

1. For the A7S Nexus3 module, the OTR register is a memory mapped register, whose address is located in the UBA. The UBA is internal to the Nexus module and can be accessed by the IEEE-ISTO 5001 tool through the JTAG port.
2. Only word writes to the OTR are valid. The data value written into the OTR is registered and formed into the Ownership Trace Message that is queued to be transmitted.
3. OTR reads do not cause Ownership Trace Messages to be transmitted by the A7S Nexus3 module.

11.11.2 Program Trace

This section details the program trace mechanisms supported by Nexus 3 for the ARM7 processor. Program trace is implemented via Branch Trace Messaging (BTM) as per the Class 2 IEEE-ISTO 5001 standard definition.

11.11.2.1 Branch Trace Messaging (BTM)

Traditional Branch Trace Messaging (Thumb mode only) facilitates program trace by providing the following types of information:

- Messaging for taken direct branches includes how many sequential instructions were executed since the last taken branch or exception. Direct (or indirect) branches not taken are counted as sequential instructions.
- Messaging for taken indirect branches and exceptions includes how many sequential instructions were executed since the last taken branch or exception and the unique portion of the branch target address or exception vector address.

Branch History Messaging (ARM and Thumb modes) facilitates program trace by providing the following information.

- Messaging for taken indirect branches and exceptions includes how many sequential instructions were executed since the last predicate instruction, exception, or taken indirect branch, the unique portion of the branch target address or exception vector address, as well as a branch/predicate instruction history field. Each bit in the history field represents a direct branch or predicated instruction where a value of one (1) indicates taken, and a value of zero (0) indicates not taken.

11.11.2.1.1 ARM7 Indirect Branch Message Instructions

The table below shows the types of instructions and events which cause Indirect Branch Messages or Branch History Messages to be encoded.

Table 11-37. Indirect Branch / Branch History Message Instructions

Source of Indirect Branch Message	Instructions (ARM mode)	Instr. (Thumb mode)
Taken Register / PC Indirect Branch instruction	bx	bx
Sequential instruction w/ PC as destination reg.	any that write to R15 (PC)	any that write to R15 (PC)
Interrupt / exception	swi, undefined instructions	undefined instructions
Return from interrupt / exception	movs, subs, ldm(3)	N/A

NOTE

Instructions with the Program Counter (PC) as the destination register (R15) that are interrupted may or may not cause a BTM to be queued depending on which stage of the pipe the instruction has reached when the interrupt occurs.

11.11.2.1.2 ARM7 Direct Branch Message Instructions

The table below shows the types of instructions that will cause Direct Branch Messages or will toggle a bit in the instruction history buffer to be messaged out in a Resource Full Message or Branch History Message.

Table 11-38. Direct Branch Message Instructions

Source of Direct Branch Message	Instructions (ARM mode)	Instr. (Thumb mode)
Taken Direct Branch instruction	b, bl	b(1), b(2), bl

11.11.2.1.3 BTM in ARM mode

Due to the conditional nature of 32-bit ARM instructions, traditional BTM Messaging can accurately track the number of sequential instructions between branches, but cannot accurately indicate which instructions were conditionally executed, and which were not.

Branch History Messaging solves this problem by providing a predicated instruction history field in each Indirect Branch Message. Each bit in the history represents a predicated instruction or direct branch. A value of one (1) indicates the conditional instruction was executed or the direct branch was taken. A value of zero (0) indicates the conditional instruction was not executed or the direct branch was not taken.

Branch History Messages solve predicated instruction tracking and save bandwidth since only indirect branches cause messages to be queued.

11.11.2.1.4 BTM in Thumb mode

Based on the PTM bit in the DC Register (DC[25]), Program Tracing can utilize either Branch History Messages (DC[25]=1'b0) or traditional Direct/Indirect Branch Messages (DC[25]=1'b1).

Branch History will save bandwidth and keep consistency between methods of Program Trace in ARM and Thumb modes, yet may lose temporal order between branch events and other types of messages. Since direct branches are not messaged, but included in the history field of the Indirect Branch History Message, other types of messages may enter the FIFO between Branch History Messages. The development tool cannot determine the ordering of events that occurred with respect to direct branches simply by the order in which messages are sent out.

Traditional BTM messages maintain their temporal ordering because each event that can cause a message to be queued will enter the FIFO in the order it occurred and will be messaged out maintaining that order.

11.11.2.2 Branch Trace Message Formats (History and Traditional)

The A7S Nexus3 block supports three types of traditional BTM Messages: Direct, Indirect, and Synchronization Messages. It supports two types of branch history BTM Messages: Indirect Branch History, and Indirect Branch History with Synchronization Messages. Debug Status Messages, Program Correlation Messages and Error Messages are also supported.

11.11.2.2.1 Indirect Branch Messages (History)

Indirect branches include all taken branches whose destination is determined at run time, interrupts and exceptions. If DC[25] is cleared while in Thumb mode, or the ARM7 processor is in full 32-bit ARM mode, indirect branch information is messaged out in the following format

(1–32 bits)	(1–32 bits)	(1–8 bits)	(4 bits)	(6 bits)
HIST	U-ADDR	I-CNT	SRC	TCODE (011100)

Max length = 82 bits; Min length = 13 bits

Figure 11-26. Indirect Branch Message (History) Format

11.11.2.2.2 Indirect Branch Messages (Traditional)

If DC[25] is set in Thumb mode, indirect branch information is messaged out in the following format:

(1–32 bits)	(1–8 bits)	(4 bits)	(6 bits)
U-ADDR	I-CNT	SRC	TCODE (000100)

Max length = 50 bits; Min length = 12 bits

Figure 11-27. Indirect Branch Message (Traditional) Format

11.11.2.2.3 Direct Branch Messages (Traditional)

Direct branches (conditional or unconditional) are all taken branches whose destination is fixed in the instruction opcode. If DC[25] is set while in Thumb mode, direct branch information is messaged out in the following format:

(1–8 bits)	(4 bits)	(6 bits)
I-CNT	SRC	TCODE (000100)

Max length = 18 bits; Min length = 11 bits

Figure 11-28. Direct Branch Message Format

NOTE

When DC[25] is cleared in Thumb mode or the ARM7 processor is in full 32-bit ARM mode, Direct Branch Messages will not be transmitted. Instead, each direct branch or predicated instruction will toggle a bit in the history buffer.

11.11.2.2.4 Resource Full Messages

The Resource Full Message is used in conjunction with the Branch History Messages. The Resource Full Message is generated when the internal branch/predicate history buffer is full. If synchronization is needed at the time this message is generated, the synchronization is delayed until the next Branch Trace Message that is not a Resource Full Message.

The current value of the history buffer is transmitted as part of the Resource Full Message. This information can be concatenated by the tool with the branch/predicate history information from subsequent messages to obtain the complete branch history for a message. The history value is reset by this message, and the I-CNT value is reset as a result of a bit being added to the history buffer.

(1–32 bits)	(4 bits)	(4 bits)	(6 bits)
HIST	RCODE (0001)	SRC	TCODE (011011)

Max length = 46 bits; Min length = 15 bits

Figure 11-29. Resource Full Message Format

11.11.2.2.5 Program Correlation Messages

Program Correlation Messages (PCM) are used to correlate events to the program flow that may not be associated with the instruction stream. In order to maintain accurate instruction tracing information when entering debug mode, program trace is disabled or entering CPU low power mode a PCM is sent. It includes the instruction count and branch history. Program Correlation is messaged out in the following format:

(1–32 bits)	(1–8 bits)	(4 bits)	(4 bits)	(6 bits)
HIST	I-CNT	ECODE	SRC	TCODE (100001)

Max length = 54 bits; Min length = 16 bits

Figure 11-30. Program Correlation Message Format

11.11.2.2.6 BTM Overflow Error Messages

An Error Message occurs when a new message cannot be queued due to the message queue being full. The FIFO will discard incoming messages until it has completely emptied the queue. Once emptied, an Error Message will be queued. The error encoding will indicate which type(s) of messages attempted to be queued while the FIFO was being emptied.

If only a Program Trace Message attempts to enter the queue while it is being emptied, the Error Message will incorporate the Program Trace only error encoding (00001). If both OTM and Program Trace Messages attempt to enter the queue, the Error Message will incorporate the OTM and Program Trace error encoding (00111). If a Watchpoint also attempts to be queued while the FIFO is being emptied, then the Error Message will incorporate error encoding (01000).

NOTE

The OVC bits within the DC Register can be set to delay the CPU by asserting the FIFOFULL signal in order to alleviate (but not eliminate) potential overruns.

Error information is messaged out in the following format:

(5 bits)	(4 bits)	(6 bits)
ERROR (00001 / 00111 / 01000)	SRC	TCODE (001000)

Fixed length = 15 bits

Figure 11-31. Error Message Format

11.11.2.2.7 Program Trace Synchronization Messages

A Program Trace Direct/Indirect Branch with Sync. or Indirect Branch History with Sync. Message is messaged via the auxiliary port (provided Program Trace is enabled) for the following conditions (see Table 11-39):

- Initial Program Trace Message upon the first direct (traditional only) or indirect branch after exit from system reset or whenever program trace is enabled.
- Upon direct (traditional only) or indirect branch after returning from a Low Power state.

- Upon direct (traditional only) or indirect branch after returning from Debug Mode.
- Upon direct (traditional only) or indirect branch after occurrence of queue overrun (can be caused by any trace message), provided Program Trace is enabled.
- Upon direct (traditional only) or indirect branch after the periodic program trace counter has expired indicating 255 *without-sync* Program Trace Messages have occurred since the last *with-sync* message occurred.
- Upon direct (traditional only) or indirect branch after assertion of the Event In ($\overline{\text{EVTI}}$) pin if the EIC bits within the DC Register have enabled this feature.
- Upon direct (traditional only) or indirect branch after the sequential instruction counter has expired indicating 255 instructions have occurred between branches or since the last bit was entered in the history field.
- Upon direct (traditional only) or indirect branch after a BTM Message was lost due to an attempted access to a secure memory location (for SOCs with security).
- Upon direct (traditional only) or indirect branch after a BTM Message was lost due to a collision with two other higher priority messages entering the FIFO.

If the A7S Nexus3 module is enabled at reset, an $\overline{\text{EVTI}}$ assertion initiates a Program Trace Indirect Branch History with Sync. Message (if Program Trace is enabled) upon the first indirect branch. The message will be a history type message because the ARM7 core will be in full 32-bit ARM mode upon exit from reset. The history field will contain all taken/not taken direct branch and predicated instructions which occur before the first indirect branch.

The formats for Program Trace Direct/Indirect Branch with Sync. Messages and Indirect Branch History with Sync. Messages are as follows:

(1–32 bits)	(1–32 bits)	(1–8 bits)	(4 bits)	(6 bits)
HIST	F-ADDR	I-CNT	SRC	TCODE (011101)

Max length = 82 bits; Min length = 13 bits

Figure 11-32. Indirect Branch History w/ Sync. Message Format

(1–32 bits)	(1–8 bits)	(4 bits)	(6 bits)
F-ADDR	I-CNT	SRC	TCODE (001011 or 001100)

Max length = 50 bits; Min length = 12 bits

Figure 11-33. Direct/Indirect Branch with Sync. Message Format (traditional)

Exception conditions that result in Program Trace Synchronization are summarized in Table 11-39.

Table 11-39. Program Trace Exception Summary

Exception Condition	Exception Handling
System Reset Negation	Upon entry into JTAG Test-Logic-Reset state, queue pointers, counters, state machines, and registers within the ARM7 Nexus module are reset. Upon the first branch out of system reset (if Program Trace is enabled), the first Program Trace Message is a Direct/Indirect Branch w/ Sync. Message.
Program Trace Enabled	The first Program Trace Message (after Program Trace has been enabled) is a synchronization message.
Exit from Low Power/Debug	Upon exit from a Low Power mode or Debug mode the next direct/indirect branch will be converted to a Direct/Indirect Branch with Sync. Message.
Queue Overrun	An Error Message occurs when a new message cannot be queued due to the message queue being full. The FIFO will discard messages until it has completely emptied the queue. Once emptied, an Error Message will be queued. The error encoding will indicate which type(s) of messages attempted to be queued while the FIFO was being emptied. The next BTM message in the queue will be a Direct/Indirect Branch w/ Sync. Message.
Periodic Program Trace Synchronization	A forced synchronization occurs periodically after 255 Program Trace Messages have been queued. A Direct/Indirect Branch w/ Sync. Message is queued. The periodic program trace message counter then resets.
Event In	If the Nexus module is enabled, an $\overline{\text{EVTI}}$ assertion initiates a Direct/Indirect Branch w/ Sync. Message upon the next direct/indirect branch (if Program Trace is enabled and the EIC bits of the DC Register have enabled this feature).
Sequential Instruction Count Overflow	When the sequential instruction counter reaches its maximum count (up to 255 sequential instructions may be executed), a forced synchronization occurs. The sequential counter then resets. A Program Trace Direct/Indirect Branch w/ Sync.Message is queued upon execution of the next branch.
Attempted Access to Secure Memory	For SOCs that implement security, any attempted branch to secure memory locations will temporarily disable Program Trace & cause the corresponding BTM to be lost. The following direct/indirect branch will queue a Direct/Indirect Branch w/ Sync. Message. The count value within this message will be inaccurate since the re-enable of Program Trace is not necessarily aligned on an instruction boundary.
Collision Priority	Messages have the following priority: Error, Ownership Trace, Watchpoint, Debug, Program Correlation, Branch Trace (BTM), Data Trace. A BTM message which attempts to enter the queue at the same time as two other higher priority messages will be lost. An Error Message will be sent indicating the BTM was lost. Instruction counts are not reset when a BTM is lost so subsequent instructions will be added to the preempted message's instruction count until a change of flow or predicated instruction is reached. If a message is generated as a result of the subsequent change of flow, then the instruction count in that message will include the instruction count of the preempted message. Similarly, the history buffer is not reset when a BTM is lost due to collision. In ARM mode, the branch that caused the preempted message will receive a history bit since indirect branches may be conditional in ARM mode.

11.11.2.3 BTM Operation

11.11.2.3.1 Enabling Program Trace

Both types of Branch Trace Messaging can be enabled in one of two ways.

- Setting the TM field of the DC Register to enable Program Trace (DC[2]).
- Using the PTS field of the WT Register to enable Program Trace on Watchpoint hits (ARM7 watchpoints are configured within the CPU).

NOTE

Setting DC[25] will select the traditional Branch Trace Messaging format when in Thumb mode. By default, Branch History format is used (DC[25]=1'b0). Full 32-bit ARM mode always utilizes the Branch History Message format.

11.11.2.3.2 Addressing

The ARM7 architecture supports a processor mode switch into Thumb mode with the least significant bit of the address bus set. The A7S Nexus3 module ignores this bit and always treats it as if it is zero (all instruction addresses are aligned) for the purpose of program trace messaging.

The relative address feature is compliant with the IEEE-ISTO 5001 standard recommendations, and is designed to reduce the number of bits transmitted for addresses of Indirect Branch Messages.

The address transmitted is relative to the target address of the instruction which triggered the previous Indirect Branch (or Sync) Message. It is generated by XORing the new address with the previous address, and then using only the results up to the most significant '1' in the result. To recreate this address, an XOR of the (most-significant 0-padded) message address with the previously decoded address gives the current address.

Previous Address (A1) =0x0003FC01, New Address (A2) = 0x0003F365

Message Generation:

A1 = 0000 0000 0000 0011 1111 1100 0000 0001
A2 = 0000 0000 0000 0011 1111 0011 0110 0101

A1 ⊕ A2 = 0000 0000 0000 0000 0000 1111 0110 0100

Address Message (M1) = 1111 0110 0100

Address Re-creation:

A1 ⊕ M1 = A2
A1 = 0000 0000 0000 0011 1111 1100 0000 0001
M1 = 0000 0000 0000 0000 0000 1111 0110 0100

A2 = 0000 0000 0000 0011 1111 0011 0110 0101

Figure 11-34. Relative Address Generation and Re-creation

11.11.2.3.3 Branch/Predicate Instruction History (HIST)

In full 32-bit ARM mode (and optionally in Thumb mode), BTM messaging will use the Branch History format. The branch history (HIST) packet in these messages provides a history of direct branch execution used for reconstructing the program flow. This packet is implemented as a left-shifting shift register. The register is always pre-loaded with a value of one (1). This bit acts as a stop bit so that the development

tools can determine which bit is the end of the history information. The pre-loaded bit itself is not part of the history, but is transmitted with the packet.

A value of one (1) is shifted into the history buffer on a taken direct branch (conditional or unconditional) and on any instruction whose predicate condition resolved as true. A value of zero (0) is shifted into the history buffer on any instruction whose predicate condition executed as false as well as on branches not taken. This will include indirect as well as direct branches not taken.

11.11.2.3.4 Sequential Instruction Count (I-CNT)

The I-CNT packet, is present in all BTM Messages. For traditional Branch Messages (Thumb mode only), I-CNT represents the number of sequential ARM7 instructions, or non-taken branches in between Direct/Indirect Branch Messages.

For Branch History Messages in Thumb mode, I-CNT represents the number of ARM7 instructions executed since the last taken/non-taken direct branch, last taken indirect branch or exception. Not taken indirect branches are considered sequential instructions and cause the instruction count to increment. For Branch History Messages in ARM mode, I-CNT also represents the number of ARM7 instructions executed since the last predicate instruction.

The sequential instruction counter overflows when its value reaches 255. The next BTM Message following an instruction counter overflow will be converted to a synchronization type message.

NOTE

When an undefined instruction causes an exception, the undefined instruction itself will be included in the BTM instruction count.

11.11.2.3.5 Program Trace Queueing

A7S Nexus3 implements a programmable depth queue (32 minimum entry recommended) for queuing all messages. Messages that enter the queue are transmitted via the auxiliary pins in the order in which they are queued.

11.11.2.4 Program Trace Timing Diagrams (2 MDO / 1 $\overline{\text{MSEO}}$ configuration)

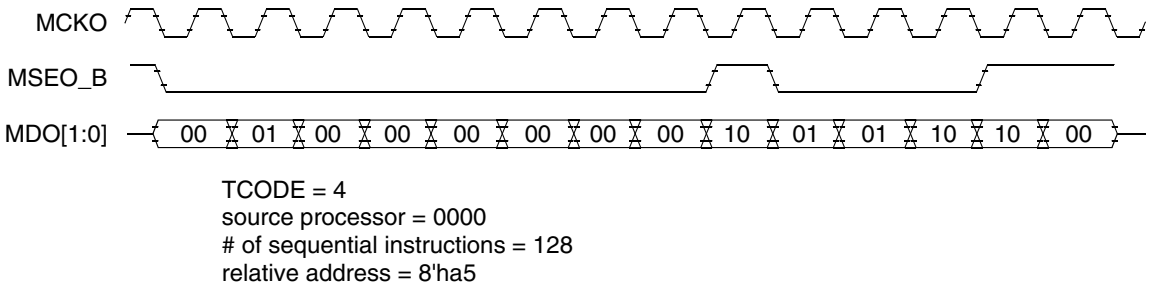
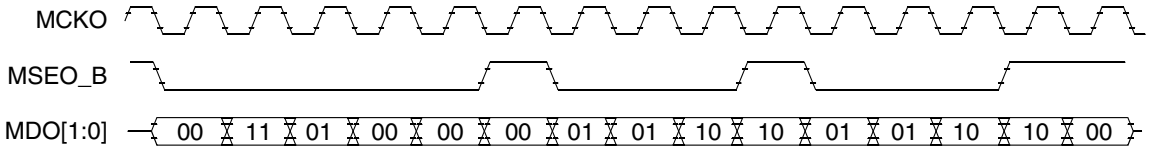
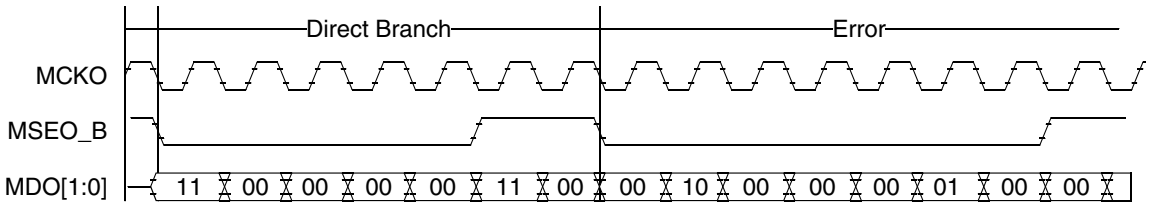


Figure 11-35. Program Trace – Indirect Branch Message (Traditional)



TCODE = 28
source processor = 0000
# of sequential instructions = 0
relative address = 8'ha5
branch history = 8'b10100101 (w/ stop)

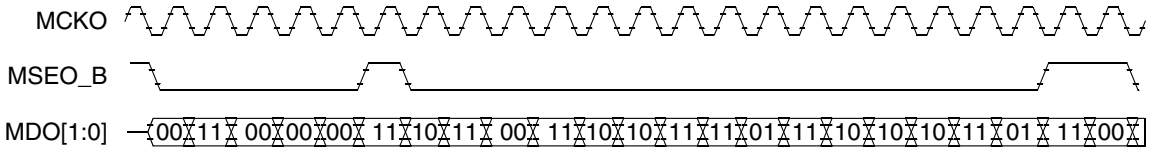
Figure 11-36. Program Trace – Indirect Branch Message (History)



DBM:
TCODE = 3
source processor = 0000
# of sequential instructions = 3

Error:
TCODE = 8
source processor = 0000
error code = 1 (queue overrun, BTM only)

Figure 11-37. Program Trace – Direct Branch (Traditional) and Error Messages



TCODE = 12
source processor = 0000
# of sequential instructions = 3
full target address = 32'hdeadface

Figure 11-38. Program Trace – Indirect Branch w/ Sync. Message (Traditional)

11.11.3 Data Trace

This section deals with the Data Trace mechanism supported by the A7S Nexus3 module. Data Trace is implemented via Data Write Messaging (DWM) and Data Read Messaging (DRM), as per the IEEE-ISTO 5001 standard.

11.11.3.1 Data Trace Messaging (DTM)

Data Trace Messaging for ARM7 is accomplished by snooping the ARM7 bus, and storing the information for qualifying accesses (based on enabled features and matching target addresses). The A7S Nexus3 module traces all data access that meet the selected range and attributes.

NOTE

Data Trace is only performed on the ARM7 bus. DMA accesses to the Advanced High-performance Bus (AHB) are not traced.

11.11.3.2 DTM Message Formats

The A7S Nexus3 block supports five types of DTM Messages: Data Write, Data Read, Data Write Synchronization, Data Read Synchronization and Error Messages.

11.11.3.2.1 Data Write Messages

The Data Write Message contains the data write value and the address of the write access, relative to the previous Data Trace Message. Data Write Message information is messaged out in the following format:

(1–32 bits)	(1–32 bits)	(3 bits)	(4 bits)	(6 bits)
Data Value	Relative Address	Data Size	Src. Proc.	TCODE (000101)

Max length = 77 bits; Min length = 15 bits

Figure 11-39. Data Write Message Format

11.11.3.2.2 Data Read Messages

The Data Read Message contains the data read value and the address of the read access, relative to the previous Data Trace Message. Data Read Message information is messaged out in the following format:

(1–32 bits)	(1–32 bits)	(3 bits)	(4 bits)	(6 bits)
Data Value	Relative Address	Data Size	Src. Proc.	TCODE (000110)

Max length = 77 bits; Min length = 15 bits

Figure 11-40. Data Read Message Format

11.11.3.2.3 DTM Overflow Error Messages

An Error Message occurs when a new message cannot be queued due to the message queue being full. The FIFO will discard incoming messages until it has completely emptied the queue. Once emptied, an Error Message will be queued. The error encoding will indicate which type(s) of messages attempted to be queued while the FIFO was being emptied.

If only a Data Trace Message attempts to enter the queue while it is being emptied, the Error Message will incorporate the Data Trace only error encoding (00010). If both OTM and Data Trace Messages attempt to enter the queue, the Error Message will incorporate the OTM and Data Trace error encoding (00111). If a Watchpoint also attempts to be queued while the FIFO is being emptied, then the Error Message will incorporate error encoding (01000).

NOTE

The OVC bits within the DC Register can be set to delay the CPU by asserting the FIFOFULL signal in order to alleviate (but not eliminate) potential overruns.

Error information is messaged out in the following format:

(5 bits)	(4 bits)	(6 bits)
Error Code (00010 / 00111 / 01000)	Src. Proc.	TCODE (001000)

Fixed length = 15 bits

Figure 11-41. Error Message Format

11.11.3.2.4 Data Trace Synchronization Messages

A Data Trace Write/Read with Sync. Message is messaged via the auxiliary port (provided Data Trace is enabled) for the following conditions (see Table 11-40):

- Initial Data Trace Message upon exit from system reset or whenever Data Trace is enabled will be a synchronization message.
- Upon returning from a Low Power state, the first Data Trace Message will be a synchronization message.
- Upon returning from Debug Mode, the first Data Trace Message will be a synchronization message.
- After occurrence of queue overrun (can be caused by any trace message), the first Data Trace Message will be a synchronization message.
- After the periodic data trace counter has expired indicating 255 *without-sync* Data Trace Messages have occurred since the last *with-sync* message occurred.
- Upon assertion of the Event In ($\overline{\text{EVTI}}$) pin, the first Data Trace Message will be a synchronization message if the EIC bits of the DC Register have enabled this feature.
- Upon Data Trace Write/Read after the previous DTM Message was lost due to an attempted access to a secure memory location (for SoCs with security).
- Upon Data Trace Write/Read after the previous DTM Message was lost due to a collision entering the FIFO between the DTM Message and two other higher priority messages.

Data Trace Synchronization Messages provide the full address (without leading zeros) and insure that development tools fully synchronize with Data Trace regularly. Synchronization messages provide a reference address for subsequent DTMs, in which only the unique portion of the Data Trace address is transmitted. The format for Data Trace Write/Read with Sync. Messages is as follows:

(1–32 bits)	(1–32 bits)	(3 bits)	(4 bits)	(6 bits)
Data Value	Full Address	Data Size	Src. Proc.	TCODE (001101 or 001110)

Max length = 77 bits; Min length = 15 bits

Figure 11-42. Data Write/Read with Sync. Message Format

Exception conditions that result in Data Trace Synchronization are summarized in Table 11-40.

Table 11-40. Data Trace Exception Summary

Exception Condition	Exception Handling
System Reset Negation	At the negation of JTAG reset ($\overline{\text{TRST}}$), queue pointers, counters, state machines, and registers within the A7S Nexus3 module are reset. If Data Trace is enabled, the first Data Trace Message is a Data Write/Read w/ Sync. Message.
Data Trace Enabled	The first Data Trace Message (after Data Trace has been enabled) is a synchronization message.
Exit from Low Power/Debug	Upon exit from a Low Power mode or Debug mode the next Data Trace Message will be converted to a Data Write/Read with Sync. Message.
Queue Overrun	An Error Message occurs when a new message cannot be queued due to the message queue being full. The FIFO will discard messages until it has completely emptied the queue. Once emptied, an Error Message will be queued. The error encoding will indicate which type(s) of messages attempted to be queued while the FIFO was being emptied. The next DTM message in the queue will be a Data Write/Read w/ Sync. Message.
Periodic Data Trace Synchronization	A forced synchronization occurs periodically after 255 Data Trace Messages have been queued. A Data Write/Read w/ Sync. Message is queued. The periodic data trace message counter then resets.
Event In	If the Nexus module is enabled, an $\overline{\text{EVTI}}$ assertion initiates a Data Trace Write/Read w/ Sync. Message upon the next data write/read (if Data Trace is enabled and the EIC bits of the DC Register have enabled this feature).
Attempted Access to Secure Memory	For SOCs that implement security, any attempted read or write to secure memory locations will temporarily disable Data Trace & cause the corresponding DTM to be lost. A subsequent read/write will queue a Data Trace Read/Write w/ Sync. Message.
Collision Priority	Messages have the following priority: Error, Ownership Trace, Watchpoint, Debug, Program Correlation, Branch Trace, Data Trace. A Data Trace message which attempts to enter the queue at the same time as two other higher priority messages will be lost.

11.11.3.3 DTM Operation

11.11.3.3.1 Enabling Data Trace Messaging

Data Trace Messaging can be enabled in one of two ways.

- Setting the TM field of the DC Register to enable Data Trace (DC[1]).
- Using the DTS field of the WT Register to enable Data Trace on Watchpoint hits (ARM7 watchpoints are configured within the CPU).

11.11.3.3.2 DTM Queueing

A7S Nexus3 implements a programmable depth queue (32 minimum entry recommended) for queuing all messages. Messages that enter the queue are transmitted via the auxiliary pins in the order in which they are queued.

11.11.3.3.3 Relative Addressing

The relative address feature is compliant with the IEEE-ISTO 5001 standard recommendations, and is designed to reduce the number of bits transmitted for addresses of Data Trace Messages. Refer to Section 11.11.2.3.2, “Addressing,” for details.

11.11.3.3.4 Data Trace Windowing

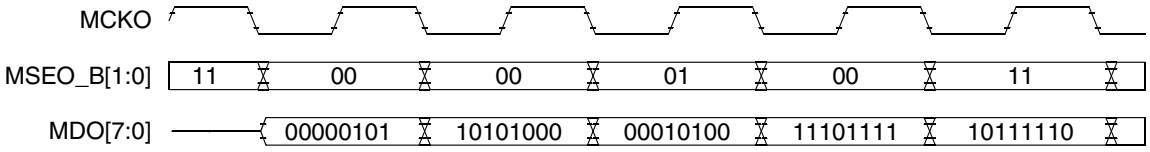
Data Write/Read Messages are enabled via the RWT1(2) field in the Data Trace Control Register (DTC) for each DTM channel. Data Trace windowing is achieved via the address range defined by the DTEA and DTSA Registers and by the RC1(2) field in the DTC. All ARM7 initiated read/write accesses which fall inside or outside these address ranges, as programmed, are candidates to be traced.

11.11.3.3.5 ARM7 Bus Cycle Cases

Table 11-41. ARM7 Bus Cycle Cases

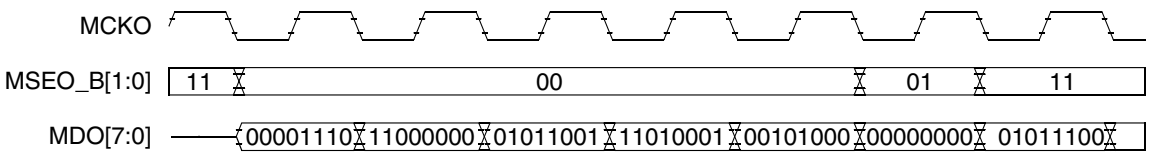
Special Case	Action
ARM7 bus cycle aborted	Cycle ignored
ARM7 bus cycle completed without error	Cycle captured & transmitted
AHB bus cycle initiated by Nexus	Cycle ignored
ARM7 bus cycle is an instruction fetch	Cycle ignored

11.11.3.4 Data Trace Timing Diagrams (8 MDO / 2 $\overline{\text{MSEO}}$ configuration)



TCODE = 5
source processor = 0000
data size = 010 (halfword)
relative address = 8'ha5
write data = 16'hbeef

Figure 11-43. Data Trace – Data Write Message



TCODE = 14
source processor = 0000
data size = 000 (byte)
full access address = 32'h01468ace
write data = 8'h5c

Figure 11-44. Data Trace – Data Read w/ Sync Message

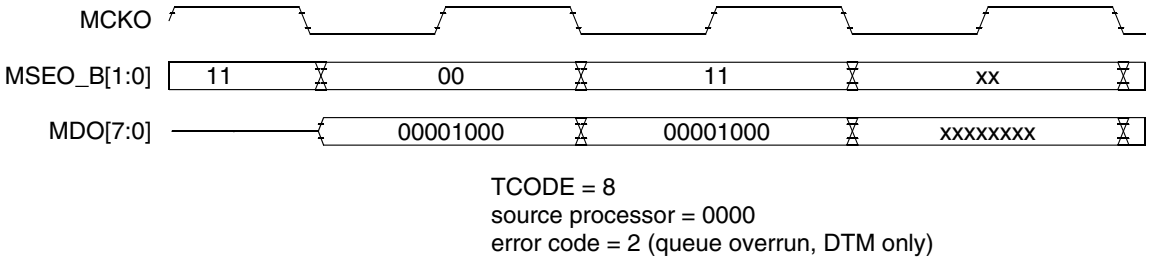


Figure 11-45. Error Message (Data Trace only encoded)

11.11.4 Watchpoint Units

The A7S Nexus3 module includes watchpoint units for watchpoint messaging and processor breakpoints. Watchpoint messages can be generated by either using the Nexus internal watchpoints or by ARM7 watchpoints. The A7S Nexus3 module supports using the internal and external watchpoint sources for triggering the start and stop of program and data trace messaging (see Section 11.10.3.8, “Watchpoint Trigger (WT)”). The occurrence of any watchpoint can be programmed to assert the event out ($\overline{\text{EVTO}}$) pin.

11.11.4.1 Watchpoint Generation

11.11.4.1.1 Internal Watchpoint Units 1 and 2

Watchpoint units 1 and 2 can be configured to assert based on an address compare, a data compare, or both address and data compare. Address comparisons may be masked on a bitwise basis with the BWAM registers. A zero bit programmed in the BWAM registers will result in the corresponding address bit being ignored during the associated address comparisons.

Data value comparisons can be masked on a byte basis with the BWDM registers. The data value compare registers (BWD1 and BWD2) are 32 bit registers. A zero bit programmed in the BWDM registers will result in the corresponding byte being ignored during the associated data value comparison. Only active byte lanes are compared. Inactive byte lanes are considered mismatching bytes unless they are masked by the BWDM registers.

The following compare modes are supported:

1. When the BWO field within a BWC register is initialized to an address only compare, a core access address that matches the associated BWA register will cause an internal watchpoint signal to be generated.
2. When the BWO field within a BWC register is initialized to a data only compare, a core access data value that matches on the associated BWD registers will cause an internal watchpoint to be generated.
3. When the BWO field within a BWC register is initialized to both address and data compare, a core access that matches both the associated BWA and BWD registers will cause an internal watchpoint to be generated.

Table 11-42 provides some example configurations for internal watchpoints.

Table 11-42. Internal Data Watchpoint Configuration Examples

Compare Operation	BWO	BWAM	BWDM	Compare Target
Single address only	010	FFFF FFFF	xxxx	BWA[31:0]
Address range only	010	FFFF FFF0	xxxx	BWA[31:4]
Signal address and word value	011	FFFF FFFF	1111	BWA[31:0] and BWD[31:0]
Word value only	001	xxxx xxxx	1111	BWD[31:0]
Signal address and byte value	011	FFFF FFFF	0001	BWA[31:0] and BWD[7:0]
			0010	BWA[31:0] and BWD[15:8]
			0100	BWA[31:0] and BWD[23:16]
			1000	BWA[31:0] and BWD[31:24]
Single address and halfword value	011	FFFF FFFF	0011	BWA[31:0] and BWD[15:0]
			1100	BWA[31:0] and BWD[31:24]

11.11.4.1.2 Internal Watchpoint Units 3 - 6

Watchpoint units 3 - 6 can be enabled to assert on an instruction address compare. The two least significant bits of the address are automatically masked for word accesses (ARM mode) and the least significant bit is masked for half word accesses (thumb mode).

11.11.4.1.3 ARM7 Watchpoints

The ARM7 EmbeddedICE module is capable of setting up to two (2) address and/or data watchpoints. Please refer to the debug chapter of the *ARM7TDMI-S (Rev 4) Technical Reference Manual* (ARM DDI 0234A) for details on watchpoint initialization.

11.11.4.2 Processor Breakpoints

The ARM watchpoint units and the Nexus watchpoint units 3 - 6 can be used to assert a processor breakpoint. If the processor is configured for halt mode, the processor will enter into debug state before executing the instruction at the specified address.

11.11.4.3 Watchpoint Messaging (WPM)

The A7S Nexus3 module provides watchpoint messaging using the IEEE-ISTO 5001 defined TCODE. Enabling watchpoint messaging is done by setting the watchpoint enable (WEN) bit in the DC register.

11.11.4.3.1 Watchpoint Message

When watchpoint messaging is enabled, and any of the eight (8) possible watchpoint sources asserts, a watchpoint message will be sent to the queue to be transmitted. The message indicates which watchpoint(s) asserted based on the encodings in Table 11-43.

(8 bits)	(4 bits)	(6 bits)
Watchpoint Source	Src. Proc.	TCODE (001111)

Fixed length = 18 bits

Figure 11-46. Watchpoint Message Format

Table 11-43. Watchpoint Source Description

Watchpoint Source (8-bits)	Watchpoint Description
xxxx xxx1	ARM7 Watchpoint 0
xxxx xx1x	ARM7 Watchpoint 1
xxxx x1xx	Nexus Watchpoint 1
xxxx 1xxx	Nexus Watchpoint 2
xxx1 xxxx	Nexus Watchpoint 3
xx1x xxxx	Nexus Watchpoint 4
x1xx xxxx	Nexus Watchpoint 5
1xxx xxxx	Nexus Watchpoint 6

11.11.4.4 Watchpoint Error Message

An Error Message occurs when a new message cannot be queued due to the message queue being full. The FIFO will discard messages until it has completely emptied the queue. Once emptied, an Error Message will be queued. The error encoding will indicate which type(s) of messages attempted to be queued while the FIFO was being emptied.

If only a Watchpoint Message attempts to enter the queue while it is being emptied, the Error Message will incorporate the Watchpoint only error encoding (00110). If an OTM and/or Program Trace Message also attempts to enter the queue while it is being emptied, the Error Message will incorporate error encoding (01000).

NOTE

The OVC bits within the DC Register can be set to delay the CPU by asserting the FIFOFULL signal in order to alleviate (but not eliminate) potential overruns.

Error information is messaged out in the following format (see Table 11-2):

(5 bits)	(4 bits)	(6 bits)
Error Code (00110 / 01000)	Src. Proc.	TCODE (001000)

Fixed length = 15 bits

Figure 11-47. Error Message Format

11.11.4.5 Watchpoint Timing Diagram (2 MDO / 1 $\overline{\text{MSEO}}$ configuration)

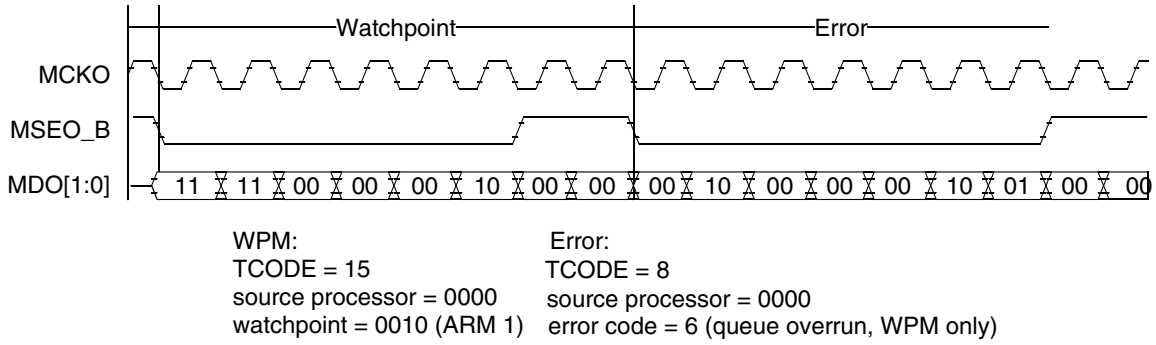


Figure 11-48. Watchpoint Message & Watchpoint Error Message

11.11.5 Read/Write Access

The Read/Write access feature allows access to internal memory mapped resources via the JTAG port. The Read/Write mechanism supports single as well as block reads and writes via an AHB system bus.

11.11.5.1 Functional Description

The Nexus3 module includes the capability of accessing resources on the AHB. All accesses are setup and initiated by the Read/Write Access Control/Status Register (RWCS), as well as the Read/Write Access Address (RWA) and Read/Write Access Data Registers (RWD).

11.11.5.2 Read/Write Access to Internal Nexus Registers

Access to Nexus register resources is enabled by loading a single instruction ("*NEXUS-ACCESS*") into the JTAG Instruction Register (IR). For the A7S Nexus3 block, the JTAG IR value is programmable at the platform or SOC integration level. It can be programmed to any of the non-ARM7 implemented IR values (see Table 11-35).

Table 11-44. JTAG Nexus3 Register Select

JTAG Instruction	JTAG Access Opcode	Read/Write
NEXUS-ACCESS	0x*nn* (hex) (programmable)	W

Once the JTAG "*NEXUS-ACCESS*" instruction has been loaded, the JTAG port allows communication with all Nexus registers according to the map in Table 11-21.

Reading/writing of a Nexus register then requires two (2) passes through the Data-Scan (DR) path of the JTAG state machine (see 11.12, "IEEE 1149.1 State Machine and RD/WR Sequences").

1. The first pass through the DR selects the Nexus register to be accessed by providing an index (see Table 11-21), and the direction (read/write). This is achieved by loading an 8-bit value into the JTAG Data Register (DR). This register has the following format:

Access: User read/write

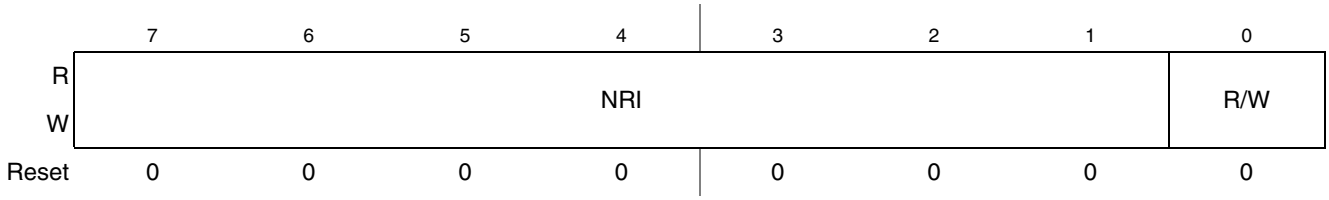


Figure 11-49. JTAG Data Register (DR)

Table 11-45. JTAG Data Register Field Description

Field	Description
7–1 NRI	Nexus Register Index *nn* Selected from values in Table 11-21
0 R/W	R/W – Read Write 0 Read 1 Write

2. The second pass through the DR then shifts the data in or out of the JTAG port, LSB first.
   a) During a read access, data is latched from the selected Nexus register when the JTAG state machine (see 11.12, “IEEE 1149.1 State Machine and RD/WR Sequences”) passes through the “Capture-DR” state.
   b) During a write access, data is latched into the selected Nexus register when the JTAG state machine (see 11.12, “IEEE 1149.1 State Machine and RD/WR Sequences”) passes through the “Update-DR” state.

11.11.5.3 Memory Mapped Register Access via JTAG

Using the Read/Write Access Registers (RWCS/RWA/RWD), memory mapped AHB resources can be accessed through Nexus. The following steps are required to access memory mapped resources:

NOTE

Read/Write Access can only access memory mapped resources when system reset is negated.

11.11.5.3.1 Single Write Access

1. Initialize the Read/Write Access Address Register (RWA) through the JTAG access method outlined in Section 11.11.5.2, “Read/Write Access to Internal Nexus Registers,” using the Nexus Register Index of 0x9 (see Table 11-21). Configure as follows:
   — Write Address → 32'hxxxxxxxx (write address)
2. Initialize the Read/Write Access Control/Status Register (RWCS) through the JTAG access method outlined in Section 11.11.5.2, “Read/Write Access to Internal Nexus Registers,” using the Nexus Register Index of 0x7 (see Table 11-21). Configure the bits as follows:
   — Access Control (AC) → 1'b1 (to indicate start access)
   — Map Select (MAP) → 3'b000 (primary memory map)

— Access Priority (PR) → 2'b00 (lowest priority)
— Read/Write (RW) → 1'b1 (write access)
— Word Size (SZ) → 3'b0xx (32-bit, 16-bit, 8-bit)
— Access Count (CNT) → 14'h0000 or 14'h0001(single access)

NOTE

Access Count (CNT) of 14'h0000 or 14'h0001 will perform a single access.

3. Initialize the Read/Write Access Data Register (RWD) through the JTAG access method outlined in Section 11.11.5.2, “Read/Write Access to Internal Nexus Registers,” using the Nexus Register Index of 0xA (see Table 11-21). Configure as follows:
   — Write Data → 32'hxxxxxxxx (write data)
4. The Nexus block will then arbitrate for the AHB and transfer the data value from the RWD Register to the memory mapped address in the Read/Write Access Address Register (RWA).

   When the access has completed without error (ERR=1'b0), the Nexus block asserts the $\overline{\text{RDY}}$ pin (see Table 11-13 for detail on $\overline{\text{RDY}}$) and clears the DV bit in the RWCS Register. This indicates that the device is ready for the next access.

NOTE

Only the $\overline{\text{RDY}}$ pin as well as the DV and ERR bits within the RWCS provide Read/Write Access status to the external development tool. The development tool needs to provide at least one clock after entering the Run-Test-Idle state to ensure that the $\overline{\text{RDY}}$ pin is asserted.

11.11.5.3.2 Block Write Access

1. For a block write access, follow Steps 1, 2, and 3 outlined in Section 11.11.5.3.1, “Single Write Access,” to initialize the registers, using a value greater than one (14'h0001) for the CNT field in the RWCS Register.
2. The Nexus block will then arbitrate for the AHB and transfer the first data value from the RWD Register to the memory mapped address in the Read/Write Access Address Register (RWA).

   When the transfer has completed without error (ERR=1'b0), the address from the RWA Register is incremented to the next word size (specified in the SZ field) and the number from the CNT field is decremented. The Nexus block will then assert the $\overline{\text{RDY}}$ pin. This indicates that the device is ready for the next access.

NOTE

The actual RWA value as well as the CNT field within the RWCS are not changed when executing a block write access. The original values can be read by the external development tool at any time.

3. Repeat Step 3 in Section 11.11.5.3.1, “Single Write Access,” until the internal CNT value is zero (0). When this occurs, the DV bit within the RWCS will be cleared to indicate the end of the block write access.

11.11.5.3.3 Single Read Access

1. Initialize the Read/Write Access Address Register (RWA) through the JTAG access method outlined in Section 11.11.5.2, “Read/Write Access to Internal Nexus Registers,” using the Nexus Register Index of 0x9 (see Table 11-21). Configure as follows:
   — Read Address → 32'hxxxxxxxx (read address)
2. Initialize the Read/Write Access Control/Status Register (RWCS) through the JTAG access method outlined in Section 11.11.5.2, “Read/Write Access to Internal Nexus Registers,” using the Nexus Register Index of 0x7 (see Table 11-21). Configure the bits as follows:
   — Access Control (AC) → 1'b1 (to indicate start access)
   — Map Select (MAP) → 3'b000 (primary memory map)
   — Access Priority (PR) → 2'b00 (lowest priority)
   — Read/Write (RW) → 1'b0 (read access)
   — Word Size (SZ) → 3'b0xx (32-bit, 16-bit, 8-bit)
   — Access Count (CNT) → 14'h0000 or 14'h0001(single access)

NOTE

Access Count (CNT) of 14'h0000 or 14'h0001 will perform a single access.

3. The Nexus block will then arbitrate for the AHB and the read data will be transferred from the AHB to the RWD Register.
   When the transfer completed without error (ERR=1'b0), the Nexus block asserts the $\overline{\text{RDY}}$ pin (see Table 11-13 for detail on $\overline{\text{RDY}}$) and sets the DV bit in the RWCS Register. This indicates that the device is ready for the next access.
4. The data can then be read from the Read/Write Access Data Register (RWD) through the JTAG access method outlined in Section 11.11.5.2, “Read/Write Access to Internal Nexus Registers using the Nexus Register Index of 0xA (see Table 11-21).

NOTE

Only the $\overline{\text{RDY}}$ pin as well as the DV and ERR bits within the RWCS provide Read/Write Access status to the external development tool. The development tool needs to provide at least one clock after entering the Run-Test-Idle state to ensure that the $\overline{\text{RDY}}$ pin is asserted

11.11.5.3.4 Block Read Access

1. For a block read access, follow Steps 1 and 2 outlined in Section 11.11.5.3.3, “Single Read Access,” to initialize the registers, using a value greater than one (14'h0001) for the CNT field in the RWCS Register.
2. The Nexus block will then arbitrate for the AHB and the read data will be transferred from the AHB to the RWD Register.
   When the transfer has completed without error (ERR=1'b0), the address from the RWA Register is incremented to the next word size (specified in the SZ field) and the number from the CNT field is decremented. The Nexus block will then assert the $\overline{\text{RDY}}$ pin. This indicates that the device is ready for the next access.

NOTE

The actual RWA value as well as the CNT field within the RWCS are not changed when executing a block read access. The original values can be read by the external development tool at any time.

3. The data can then be read from the Read/Write Access Data Register (RWD) through the JTAG access method outlined in Section 11.11.5.2, "Read/Write Access to Internal Nexus Registers," using the Nexus Register Index of 0xA (see Table 11-21).
4. Repeat Steps 3 and 4 in Section 11.11.5.3.3, "Single Read Access," until the CNT value is zero (0). When this occurs, the DV bit within the RWCS is set to indicate the end of the block read access.

11.11.5.4 Error Handling

The A7S Nexus3 module handles various error conditions as follows:

11.11.5.4.1 AHB Read/Write Error

All address and data errors that occur on read/write accesses to the AHB will return a transfer error encoding on the HRESP[1:0] signals. If HRESP[1:0] = 2'b01:

1. The access is terminated without re-trying (AC bit is cleared)
2. The ERR bit in the RWCS Register is set
3. The Error Message is sent (TCODE = 8) indicating Read/Write Error

11.11.5.4.2 Access Termination

The following cases are defined for sequences of the Read/Write protocol that differ from those described in the above sections.

1. If the AC bit in the RWCS Register is set to start Read/Write accesses and invalid values are loaded into the RWD and/or RWA, then an AHB access error may occur. This is handled as described above.
2. If a block access is in progress (all cycles not completed), and the RWCS Register is written, then the original block access is terminated at the boundary of the nearest completed access.
   a) If the RWCS is written with the AC bit set, the next Read/Write access will begin and the RWD can be written to/ read from.
   b) If the RWCS is written with the AC bit cleared, the Read/Write access is terminated at the nearest completed access. This method can be used to break (early terminate) block accesses.

11.11.5.4.3 Read/Write Access Error Message

The Read/Write Access Error Message is sent out when an AHB access error (read or write) error has occurred.

Error information is messaged out in the following format:

(5 bits)	(4 bits)	(6 bits)
Error Code (00011)	Src. Proc.	TCODE (001000)

Fixed length = 15 bits

Figure 11-50. Error Message Format

11.11.5.5 Timing Diagram

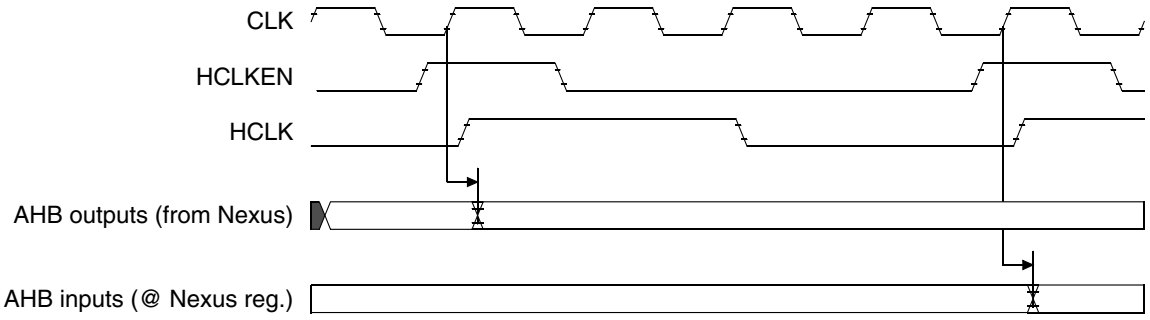


Figure 11-51. A7S Nexus3 DMA clock relationships

The A7S Nexus3 Read/Write access timing meets the timing requirements for the AHB. The Nexus module uses the processor clock gated with an AHB clock enable for all DMA transfers. This clock will correspond to the rising edge of the actual AHB clock. The timing diagram in Figure 11-51 above shows the relationship between the processor clock (CLK), the AHB clock (HCLK) and the AHB clock enable (HCLKEN) for DMA writes and reads. Using this clocking method for Nexus read/write access eliminates the need for a separate asynchronous clock input into the A7S Nexus3 module.

11.11.6 System Status

11.11.6.1 Debug Status Messages

Debug Status Messages report low power mode and debug status. Entering/exiting Debug Mode as well as entering a Low Power Mode will trigger a Debug Status Message. Debug status information is sent out in the following format:

(8 bits)	(4 bits)	(6 bits)
DS[31:24]	SRC	TCODE (000000)

Fixed length = 18 bits

Figure 11-52. Debug Status Message Format

11.12 IEEE 1149.1 State Machine and RD/WR Sequences

11.12.1 JTAG State Machine

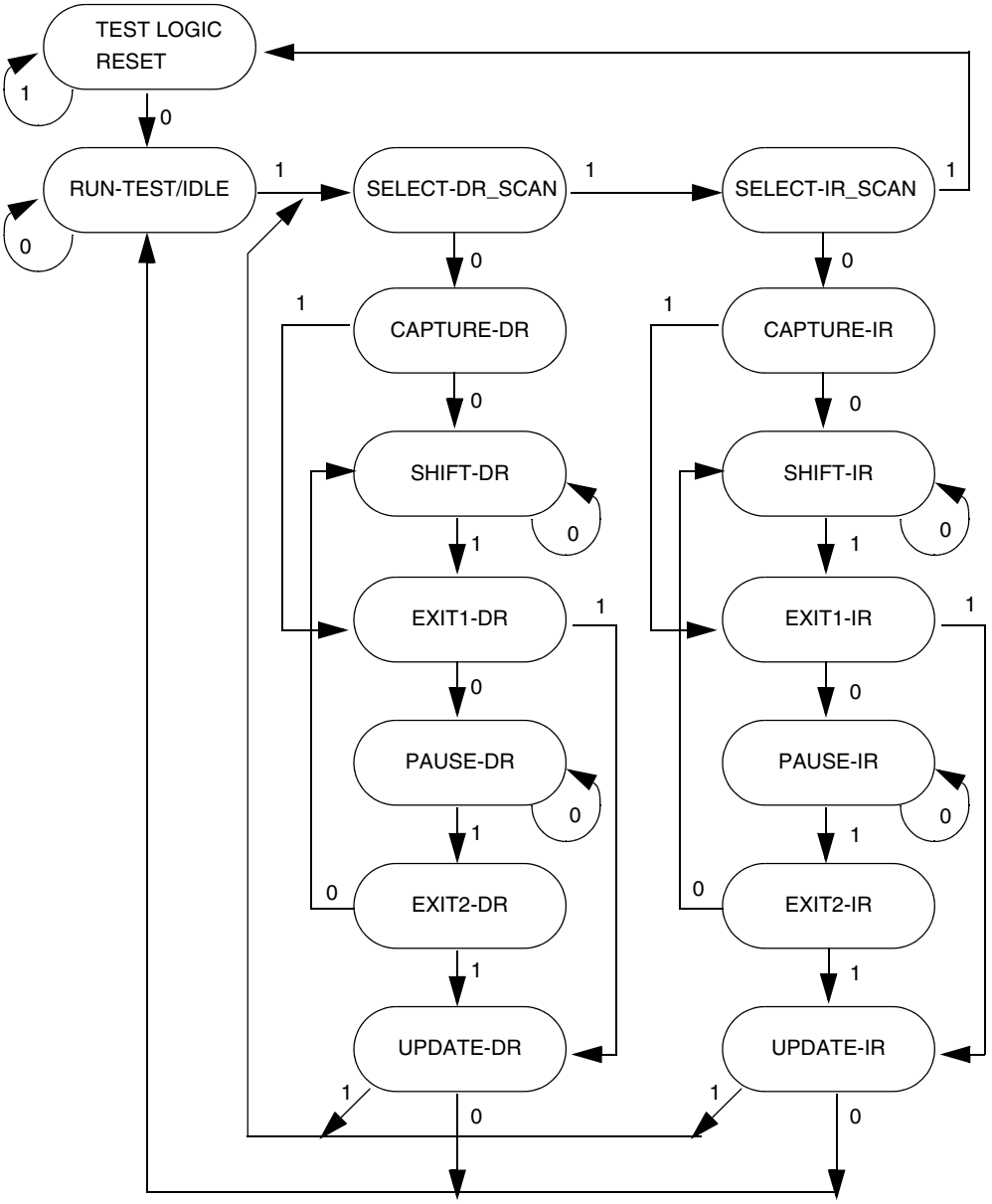


Figure 11-53. JTAG State Machine

11.12.2 JTAG Sequence for Accessing Internal Nexus Registers

Table 11-46. JTAG Sequence for Accessing Internal Nexus Registers

Step #	TMS Pin	Description
1	1	IDLE → SELECT-DR_SCAN
2	0	SELECT-DR_SCAN → CAPTURE-DR (Nexus Command Register value loaded in shifter)
3	0	CAPTURE-DR → SHIFT-DR
4	0	(7) TCK clocks issued to shift in direction (rd/wr) bit and first 6 bits of Nexus reg. addr.
5	1	SHIFT-DR → EXIT1-DR (7th bit of Nexus reg. shifted in)
6	1	EXIT1-DR → UPDATE-DR (Nexus shifter is transferred to Nexus Command Register)
7	1	UPDATE-DR → SELECT-DR_SCAN
8	0	SELECT-DR_SCAN → CAPTURE-DR (Register value is transferred to Nexus shifter)
9	0	CAPTURE-DR → SHIFT-DR
10	0	(31) TCK clocks issued to transfer register value to TDO pin while shifting in TDI value
11	1	SHIFT-DR → EXIT1-DR (MSB of value is shifted in/out of shifter)
12	1	EXIT1-DR → UPDATE-DR (if access is write, shifter is transferred to register)
13	0	UPDATE-DR → RUN-TEST/IDLE (transfer complete, Nexus controller to Reg. Select state)

11.12.3 JTAG Sequence for Read Access of Memory-Mapped Resources

Table 11-47. JTAG Sequence for Read Access of Memory-Mapped Resources

Step #	TCLK clocks	Description
1	13	Nexus Command = write to Read/Write Access Address Register (RWA)
2	37	Write RWA (initialize starting read address, data input on TDI)
3	13	Nexus Command = write to Read/Write Control/Status Register (RWCS)
4	37	Write RWCS (initialize read access mode and CNT value, data input on TDI)
5	—	Wait for falling edge of $\overline{\text{RDY}}$ pin
6	13	Nexus Command = read Read/Write Access Data Register (RWD)
7	37	Read RWD (data output on TDO)
8	—	If CNT > 0, go back to Step #6

11.12.4 JTAG Sequence for Write Access of Memory-Mapped Resources

Table 11-48. JTAG Sequence for Write Access of Memory-Mapped Resources

Step #	TCLK clocks	Description
1	13	Nexus Command = write to Read/Write Address Register (RWA)
2	37	Write RWA (initialize starting write address, data input on TDI)

Table 11-48. JTAG Sequence for Write Access of Memory-Mapped Resources (Continued)

Step #	TCLK clocks	Description
3	13	Nexus Command = write to Read/Write Access Control/Status Register
4	37	Write RWCS (initialize write access mode and CNT value, data input on TDI)
5	13	Nexus Command = read Read/Write Access Data Register (RWD)
6	37	Write RWD (data output on TDO)
7	—	Wait for falling edge of $\overline{\text{RDY}}$ pin
8	—	If CNT > 0, go back to Step #5

Chapter 12
Enhanced DMA Controller (eDMA) Module

12.1 Overview of the MAC7200 Implementation

The MAC7200 family of devices implements a Direct Memory Access controller called the Enhanced Direct Memory Access controller (eDMA). This module is implemented on other Freescale devices such as those in the PPC5500 family. It enables transfer of data between the memory, peripherals and off-chip devices with little intervention from the core, thus helping to increase system performance as well as assisting with the simplification of software development.

12.1.1 eDMA Features

- DMA transfers possible between system memories, SPIs, SCIs, I^2C, ATD, eMIOS and General Purpose I/Os
- Programmable DMA Channel Mux allows assignment of any DMA source to any of the 16 available DMA channels.
- All DMA transfers use dual address format.
- Programmable Transfer Control Descriptor stored in local DMA memory.
- Programmable Source and Destination address with configurable offset.
- Independent 32-bit Minor and 16-bit Major loop counters for ’nested’ transfers.
- Different final Source and Destination addresses allow circular Queue operation.
- Programmable priority levels for each channel.
- Bandwidth control for each channel.
- Programmable transfer sizes through Major and Minor loop counters.
- Independently Programmable read/write sizes.
- Periodic triggering of up to 8 channels.
- Round Robin channel prioritization
- Scatter-Gather functionality
- Inner Loop channel pre-emption
- Channel to Channel linking
- Software or Hardware start
- Memory Map: 32-bit byte/half-word/word addressable peripheral

NOTE

The transfer type of all transfers done by the DMA is as follows:

- Supervisor Mode
- Data
- Non-Cacheable
- Bufferable/Non-Bufferable is software programmable in the DMA

12.1.2 eDMA Implementation

The eDMA has been developed to enable it to be instantiated over a range of devices with different feature requirements. This allows the module to be reused both within families of devices and across wide product ranges. The implementation of the DMA controller on the MAC7200 family has been targeted towards cost sensitive applications while still maintaining a high level of functionality.

The MAC7200 family of devices has 16 independently programmable DMA channels available for use on the MCU. The eDMA enables the definition of transfers from memory to memory, peripherals to/from memory, and from peripheral to peripheral. The following shows all possible DMA transfer sources.

Table 12-1. DMA Channel Sources

Type	Source/Destination	DMA Requests	Comments
Memory	System RAM	-	Transfer between on-chip memories
	Program Flash	-	
	Shadow Block	-	
Peripherals	ESCI_A, ESCI_B	4	Two requests per ESCI (one for Tx, one for Rx)
	DSPI_A, DSPI_B, DSPI_C	6	Two requests per DSPI (one for Tx, one for Rx)
	I^2C	2	Two requests (one for Tx, one for Rx)
	ATD_A	2	Two requests (one for Command, one for Result)
	eMIOS	8	One request available for each timer channel
External	External Bus	-	
External (PIM)	Port A, B, C, D, E, F, G	7	Not all Ports are available on all devices. Typically used with an “Always Enabled” source
Triggered	Any of the above Sources/Destinations	8	Each request trigger can be assigned to a PIT channel or may be left "always enabled" (i.e.-continuous DMA requests)

As there are greater than 16 possible sources for the DMA controller, a channel mux is used to enable the user to define which of the sources are used, and on which channels they are assigned. Refer to Chapter 27, “Enhanced Direct Memory Access (DMA Channel MUX)” for further details.

12.1.3 eDMA External Pins

There are no eDMA signals that drive or are driven from MCU pins.

12.1.4 eDMA Bus Aborts

The DMA2 supports bus aborts on the Peripheral Bus, enforcing the following memory map:

Table 12-2. eDMA Bus Abort Memory Map

Abort	Allowed
	$0000-$0007
$0008-$000d	
	$000e-$000f
$0010-$0015	
	$0016-$001f
$0020-$0025	
	$0026-$0027
$0028-$002d	
	$002e-$002f
$0030-$00fc	
	$00fd-$010f
$0110-$0fff	
	$1000-$11ff
$1200-$3fff	

If any part of a read or write falls within an aborted region, the entire transfer is aborted. For example, a 32-bit read or write to address $000c would be aborted.

Supervisor Access: Unused

12.1.5 eDMA Differences from MAC71xx

- The debug functionality of the DMA is now enabled

12.1.6 eDMA Application Usage

NOTE

The DMA Block Guide is generic, and refers to configurations of the DMA with up to 64 channels. The DMA on the MAC72xx is 16 channels, and you must therefore take this into consideration when writing code to access DMA configuration registers and (particularly) TCDs. READs to unmapped registers will return zeros, WRITEs will be ignored.

12.1.6.1 Enabling the DMA

It is not necessary to enable the DMA before it can be used. However, the DMA Channel Mux must be configured before any DMA transfers can occur.

12.1.6.2 General Operation of the DMA

1. The channel is initialized by software loading the transfer control descriptor into the DMA2's programming model, memory-mapped through the IPS space, and implemented as local memory.
2. The channel is activated, either explicitly by software, a peripheral request or a linkage from another channel.
3. The contents of the transfer control descriptor for the activated channel is read from the local memory and loaded into the dma2_engine's registers.
4. The dma2_engine executes the data transfer defined by the inner minor loop, reading from the source and writing to the destination.
5. At the conclusion of the minor loop's execution, certain fields of the transfer control descriptor are restored from the local memory.

The entire process (steps 1-5) is repeated until the outer major loop's iteration count is exhausted. At that time, additional processing steps are completed, e.g., the optional assertion of an interrupt request signaling the transfer's completion, final adjustments to the source and destination addresses, etc.

12.1.6.3 Configuring the DMA

Configuration of the DMA is divided into four areas:

- Configuring the arbitration and system loading
- Error signalling
- DEBUG mode behavior
- Transfer Control Descriptor (TCD)
- Channel completion
- Choosing a channel activation method for each channel

12.1.6.3.1 Arbitration and System Loading

There are four arbitration schemes available, all of which can be configured by using the DMACR register.

1. Choose a channel priority scheme that fits your application:
   - Fixed-priority with Channel 0 preemption. To use this scheme, write EFPRI=1, ECH0P=1.
   - Fixed-priority with high priority Channel 0. To use this scheme, write EFPRI=1, ECH0P=0.
   - Round-robin with Channel 0 preemption. To use this scheme, write EFPRI=0, ECH0P=1.
   - Round-robin with high priority Channel 0. To use this scheme, write EFPRI=0, ECH0P=0.
2. If a fixed-priority scheme is chosen, choose the priority for each channel by writing the DCHPRI*n* registers.
3. For each channel, choose the relative priority of the DMA transfers over bus accesses from the ARM7 core and/or debugger. Write the 'bwc[1:0]' field in Word 7 of the channel's TCD (Transfer Control Descriptor). This field controls how much bandwidth the DMA consumes for each access in a minor loop. In this manner, it is possible to specify highest priority (bwc[1:0]=00) to lowest priority (bwc[1:0]=11)

12.1.6.3.2 Error Signalling

The DMA module has the capability to report three types of errors, with each error detected and reported on a channel-by-channel basis:

- Source bus error
- Destination bus error
- Configuration error

You must choose whether to implement error handling in an interrupt driven or polling method. For interrupt driven error handling, perform the following steps:

1. For each channel desired, set the corresponding bit in the DMAEEIL register. Alternatively, individual channels may also be configured by writing the corresponding DMASEEI or DMACEEI registers to respectively set or clear a DMAEEIL bit. To enable interrupt driven error handling for *all* channels, simply write $FF to the DMASEEI register.
2. When an interrupt occurs (Interrupt #0), read the DMAERRL register to determine all current pending errors. Note that a channel may have an error pending even though its corresponding Enable Error Interrupt (DMAEEIL) register bit was cleared. Also note that multiple channels may have pending errors at the same time.
3. Detailed information about the *last* recorded channel error can be obtained by reading the DMAES register.
4. Once all errors have been detected (i.e.-Step #2), clear the errors by clearing the corresponding bits in the DMAERRL register. To clear individual bits, you may also write to the DMACERR register. If you want to clear *all* pending errors, simple write $FF to the DMACERR register.

To handles error in a polled fashion, following these steps:

1. For each channel desired, clear the corresponding bit in the DMAEEIL register. Alternatively, individual channels may also be configured by writing the corresponding DMASEEI or DMACEEI registers to respectively set or clear a DMAEEIL bit.
2. When desired, read the DMAERRL register to determine all current pending errors. Note that multiple channels may have pending errors at the same time.
3. Detailed information about the *last* recorded channel error can be obtained by reading the DMAES register.
4. Once all errors have been detected (i.e.-Step #2), clear the errors by clearing the corresponding bits in the DMAERRL register. To clear individual bits, you may also write to the DMACERR register. If you want to clear *all* pending errors, simple write $FF to the DMACERR register.

NOTE

As evident from the above steps, it would be very difficult to use polling driven error handling with peripheral paced channel activation (See below). For this reason, it is recommended to use interrupt driven error handling in most cases, and especially when not using explicit software activation of channels.

12.1.6.3.3 DEBUG Mode Behavior

In any real-time system, the debugging of code affected by real-time external events can often be very difficult. For this reason, flexibility in the ability to selectively enable/disable certain kinds of events when the system is in DEBUG mode is provided. In the DMA, the user has the flexibility to configure the execution of the major loop counter in the current active channel in DEBUG mode. If desired, the EDBG bit in the DMACR register can be set in order to stop the operation of a channel.

12.1.6.3.4 Transfer Control Descriptor (TCD)

Each channel has a corresponding TCD, which must be configured to specify the source and destination information, as well as the minor and major loop counter information. Please refer to the DMA2 Block Guide for detailed information on TCDs.

12.1.6.3.5 Channel Completion

The completion of a channel (i.e.-completion of the major loop counter) can be either interrupt driven or polling driven, similar to the error signalling. If interrupt driven channel completion is desired, follow these steps:

1. Enable interrupt on major loop counter completion by setting the ‘int_maj’ bit in Word 7 of the channel’s TCD.
2. Once the interrupt is received (Interrupt #1-#16), clear the interrupt by clearing the corresponding bit in the DMAINTL register. Individual bits can be easily cleared by writing to the DMACINT register. To clear *all* pending DMA interrupts, simply write $FF to the DMACINT register. Note that it is unnecessary to read the DMAINTL register before clearing it, as each DMA Channel on the MAC72xx has a dedicated interrupt. The exact channel that caused the interrupt can be determined solely by the Interrupt Vector number in the Interrupt Controller.
3. Besides producing an interrupt at the completion of a channel, the DMA is also capable of causing an interrupt when the major loop counter is half completed. This feature can be activated by setting the ‘int_half’ bit in Word 7 of the channel’s TCD. Note that in this case it is impossible to determine whether an interrupt is signalling the completion of a channel or only the halfway completion of a channel. The Interrupt Service Routine (ISR) that handles DMA channel interrupts must explicitly keep track of this by itself.

To implement polling driven channel completion, following these steps:

1. Disable interrupt on major loop counter completion by clearing the ‘int_maj’ bit in Word 7 of the channel’s TCD.
2. Determine the completion of a channel by continuously reading Word 7 of the channel’s TCD. When the ‘done’ bit is set, the channel has completed.

12.1.6.3.6 Channel Activation Method

Each DMA Channel may be configured to use one of three channel activation methods available:

— Explicit software activation. This method allows the channel to be started explicitly by the software at any time.

— Initiation via a channel-to-channel linking mechanism for continuous transfers. This method allows a channel to be started once another channel has completed.

— Peripheral-paced hardware requests. The method allows the channel activation to be controlled by the peripheral itself.

Note than in all three methods, one activation per execution of the minor loop is required.

If explicit software activation is chosen for a channel, perform the following series of steps:

1. Configure the channel's TCD with the correct source and destination information.
2. Write the 'start' bit in Word 7 of the channel's TCD to activate the channel and start transfers.

If channel-to-channel linking is chosen for a channel, perform the following series of steps:

1. Configure the channel's TCD with the correct source and destination information, ensuring that the 'start' bit in Word 7 is cleared.
2. Set the 'e_link' bit in Word 7 of the channel's TCD.
3. Program the 'linkch[5:0]' bits in Word 7 of the channel's TCD.

If peripheral paced hardware activation is chosen for a channel, perform the following series of steps:

1. Ensure that the DMA channel you want to use is disabled by clearing the corresponding bit in the DMAERL register.
2. Configure the channel's TCD with the correct source and destination information, ensuring that the 'start' bit in Word 7 is cleared.
3. Configure the DMA Channel Mux. For basic operation (no triggering), you can write the CONFIG*n* register in the DMA Channel Mux with $80 | <Channel Source>.
4. Set the corresponding DMAERL register bit. If the 'd_req' bit in the TCD is set, then the bit in the DMAERL register will be automatically cleared after completion of the major loop. This guarantees that the channel will not be re-activated after completion without explicit software intervention.
5. Configure the peripheral to enable DMA transfers. Please refer to the documentation for the particular peripheral for more details.

12.1.6.4 Using the DMA

Accessing multiple registers

Use the scatter-gather feature of the eDMA to construct more complex sets of transactions in order to (for example) access peripherals with separate control and data transmit/receive registers. Assuming channel 1 is used, the following sequence illustrates this:

1. Peripheral requests a DMA transfer (also applies to software and triggered requests)
2. TCD1a transfers data to the peripheral's control register
3. TCD1a executes a scatter-gather operation, and reloads its TCD descriptor with TCD1b
4. TCD1b has the TCD.start bit already set in the descriptor
5. The transfer request is automatically issued once TCD1b is loaded
6. TCD1b transfers data to the peripheral's data register (as many loops as required)

7. TCD1b executes a scatter-gather operation, and reloads its TCD descriptor with TCD1a
8. TCD1a has the TCD.start bit cleared in the descriptor

Once the peripheral requests another transfer, the series of steps repeats.

Basically, the TCDs ping-pong between each other, with the throttling of block/frame transfers done by the peripheral (hardware request) or by software (software request).

Table 12-3. DMA Register Summary

DMA Register	Register "Mirrors"	Purpose
DMACR	--	DMA Control Register
DMAES	--	DMA Error Status
DMAERQH/L	DMASERQ DMACERQ	DMA Enable Request
DMAEEIH/L	DMASEEI DMACEEI	DMA Enable Error Interrupt
DMAINTH/L	DMACINT	DMA Interrupt Request
DMAERRH/L	DMACERR	DMA Error
DCHPRI*n*	--	DMA Channel *n* Priority
TCD*nn*	--	Transfer Control Descriptor *nn*

12.1.6.5 TCD Memory Initialization

As the TCD Memory does not have ECC, there is no requirement to initialize the memory in a specific manner.

12.2 The SPP DMA Controller Module (SPP_DMA2)

This section provides a more detailed description of the generice SPP DMA Controller Module.

The DMA (Direct Memory Access) is a second-generation platform module capable of performing complex data transfers with minimal intervention from a host processor via "*n*" programmable channels. Intended for use as part of the Standard Product Platform (SPP), the hardware microarchitecture includes a dma_engine which performs source and destination address calculations, and the actual data movement operations, along with a local memory containing the *transfer control descriptors* (TCD) for the channels. This SRAM-based implementation is utilized to minimize the overall module size.

Figure 12-1 is a block diagram of the DMA module.

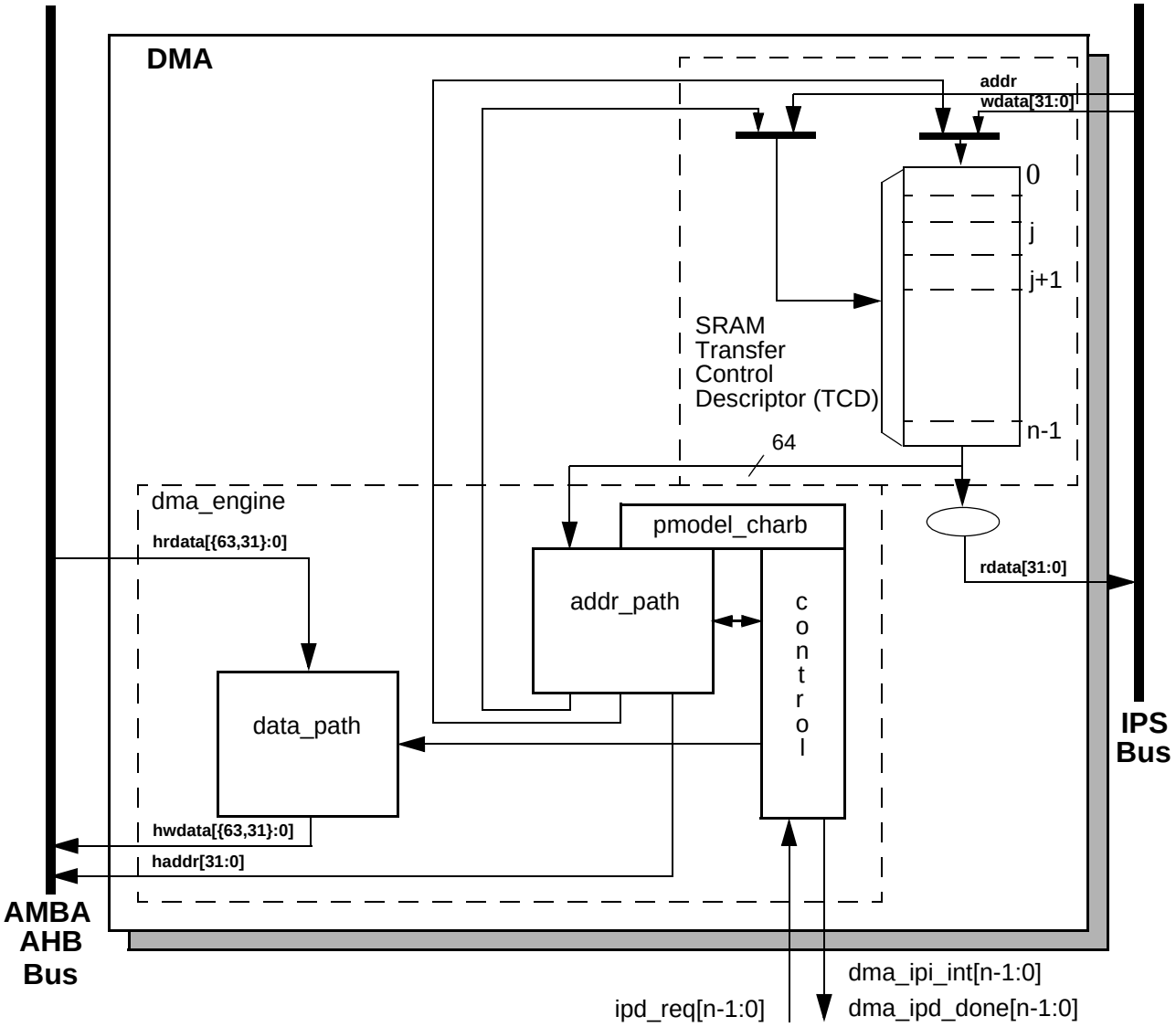


Figure 12-1. DMA Block Diagram

12.2.1 Overview

The DMA is a highly-programmable data transfer engine, which has been optimized to minimize the required intervention from the host processor. It is intended for use in applications where the data size to be transferred is statically known, and is *not* defined within the data packet itself. The DMA hardware supports:

- Single design supporting 16-, 32- and 64-channel implementations, dependent on size of the TCD memory and design parameters (the MAC72xx family of devices has 16 channels)
- Connections to the AMBA-AHB crossbar switch for bus mastering the data movement, IPS slave bus for programming the module
  - Parameterized support for 32- and 64-bit AMBA-AHB data path widths
- 32-byte transfer control descriptor per channel stored in local memory

- 32 bytes of data registers, used as temporary storage to support burst transfers

Throughout this document, *n* is used to reference the channel number. Additionally, data sizes are defined as byte (8-bit), halfword (16-bit), word (32-bit) and doubleword (64-bit).

12.2.2 Features

The DMA module supports the following features:

- All data movement via dual-address transfers: read from source, write to destination
  - Programmable source, destination addresses, transfer size, plus support for enhanced addressing modes
- Transfer control descriptor organized to support two-deep, nested transfer operations
  - An *inner* data transfer loop defined by a “minor” byte transfer count
  - An *outer* data transfer loop defined by a “major” iteration count
- Channel service request via one of three methods:
  - Explicit software initiation
  - Initiation via a channel-to-channel linking mechanism for continuous transfers
  - Independent channel linking at end of minor loop and/or major loop
  - Peripheral-paced hardware requests (one per channel)
  - For all three methods, *one service request per execution of the minor loop is required*
- Support for fixed-priority and round-robin channel arbitration
- Channel completion reported via optional interrupt requests
  - One interrupt per channel, optionally asserted at completion of major iteration count
  - Error terminations are optionally enabled per channel, and logically summed together to form a small number of error interrupt outputs
- Optional support for scatter/gather DMAprocessing

The structure of the transfer control descriptor is fundamental to the operation of the DMA module. It is defined below in a ‘C‘ pseudo-code specification:

NOTE

To compile these structures, change any periods ’.’ in the variable name to underscores ’_’.

```
typedef union {
    struct {                            /* citer.e_link = 1 */
        unsigned short citer.linkch:6;  /* link channel number, */
        unsigned short citer:9;         /* current (“major”) iteration count */
    } minor_link_enabled;               /* channel link at end of the minor loop */
    struct {                            /* citer.e_link = 0 */
        unsigned short citer:15;        /* current (“major”) iteration count */
    } minor_link_disabled;              /* no linking at end of the minor loop */
} t_minor_link_citer;

typedef union {
    struct {                            /* biter.e_link = 1 */
```

```
        unsigned short biter.linkch:6;   /* link channel number, */
        unsigned short biter:9;          /* beginning (“major”) iteration count */
    } init_minor_link_enabled;           /* channel link at end of the minor loop */
    struct {                             /* biter.e_link = 0 */
        unsigned short biter:15;         /* beginning (“major”) iteration count */
    } init_minor_link_disabled;          /* no linking at end of the minor loop */
} t_minor_link_biter;

typedef struct {
unsigned intsaddr;                       /* source address */
unsigned intsmod:5;                      /* source address modulo */
unsigned intssize:3;                     /* source transfer size */
unsigned intdmod:5;                      /* destination address modulo */
unsigned intdsize:3;                     /* destination transfer size */
short    soff;                           /* signed source address offset */
unsigned intnbytes;                      /* inner (“minor”) byte count */
int      slast;                          /* last source address adjustment */
unsigned intdaddr;                       /* destination address */
unsigned shortciter.e_link:1;            /* enable channel linking on minor loop */
t_minor_link_citerminor_link_citer;      /* conditional current iteration count */
short    doff;                           /* signed destination address offset */
int      dlast_sga;                      /* last destination address adjustment, or
                                         scatter/gather address (if e_sg = 1) */
unsigned shortbiter.e_link:1;            /* beginning channel link enable */
t_minor_link_biterminor_link_biter;      /* beginning (“major”) iteration count */
unsigned intbwc:2;                       /* bandwidth control */
unsigned intmajor.linkch:6;              /* link channel number */
unsigned intdone:1;                      /* channel done */
unsigned intactive:1;                    /* channel executing */
unsigned intmajor.e_link:1;              /* enable channel linking on major loop*/
unsigned inte_sg:1;                      /* enable scatter/gather descriptor */
unsigned intd_req:1;                     /* disable ipd_req when done */
unsigned intint_half:1;                  /* interrupt on citer = (biter >> 1) */
unsigned intint_maj:1;                   /* interrupt on major loop completion */
unsigned intstart:1;                     /* explicit channel start */
} tcd                                    /* transfer_control_descriptor */
```

where `int` refers to a 32-bit variable (unless noted otherwise) and `short` is a 16-bit variable. The basic operation of a channel is defined as:

1. The channel is initialized by software loading the transfer control descriptor into the DMA’s programming model, memory-mapped through the IPS space, and implemented as local memory.
2. The channel requests service; either explicitly by software, a peripheral request or a linkage from another channel.

   Note: The major loop executes one iteration per service request.
3. The contents of the transfer control descriptor for the activated channel is read from the local memory and loaded into the dma_engine’s registers.
4. The dma_engine executes the data transfer defined by the transfer control descriptor, reading from the source and writing to the destination. The number of iterations in the minor loop is automatically calculated by the dma_engine. The number of iterations within the minor loop is a function of the number of bytes to transfer (nbytes), the source size (ssize) and the destination size (dsize). The completion of the minor loop is equal to one iteration of the major loop.

5. At the conclusion of the minor loop's execution, certain fields of the transfer control descriptor are written back to the local TCD memory.

The process (steps 2-5) is repeated until the outer major loop's iteration count is exhausted. At that time, additional processing steps are completed, e.g., the optional assertion of an interrupt request signaling the transfer's completion, final adjustments to the source and destination addresses, etc. A more detailed description of the channel processing is listed in the pseudo-code below. This simplified example is intended to represent basic data transfers. Detailed processing associated with the error handling is omitted.

```
/* the given DMAchannel is requesting service by the software assertion of the
   tcd[channel].start bit, the assertion of an enabled ipd_req from a device, or
   the implicit assertion of a channel-to-channel link */

/* begin by reading the transfer control descriptor from the local RAM
   into the local dma_engine registers */
dma_engine        = read_from_local_memory [channel];
dma_engine.active = 1;                 /* set active flag */
dma_engine.done   = 0;                           /* clear done flag */

/* check the transfer control descriptor for consistency */
if (dma_engine.config_error == 0) {

    / * begin execution of the inner “minor” loop transfers */
    {

        /* convert the source transfer size into a byte count */
        switch (dma_engine.ssize) {
        case 0:                          /* 8-bit transfer */
            src_xfr_size = 1;
            break;
        case 1:                          /* 16-bit transfer */
            src_xfr_size = 2;
            break;
        case 2:                          /* 32-bit transfer */
            src_xfr_size = 4;
            break;
        case 3:                          /* 64-bit transfer */
            src_xfr_size = 8;
            break;
        case 4:                          /* 16-byte burst transfer */
            src_xfr_size = 16;
            break;
        case 5:                          /* 32-byte burst transfer */
            src_xfr_size = 32;
            break;
        }

        /* convert the destination transfer size into a byte count */
        switch (dma_engine.dsize) {
        case 0:                          /* 8-bit transfer */
            dest_xfr_size = 1;
            break;
        case 1:                          /* 16-bit transfer */
            dest_xfr_size = 2;
```

```
        break;
    case 2:                          /* 32-bit transfer */
        dest_xfr_size = 4;
        break;
    case 3:                          /* 64-bit transfer */
        dest_xfr_size = 8;
        break;
    case 4:                          /* 16-byte burst transfer */
        dest_xfr_size = 16;
        break;
    case 5:                          /* 32-byte burst transfer */
        dest_xfr_size = 32;
        break;
    }

    /* determine the larger of the two transfer sizes, this value reflects */
    /* the number of bytes transferred per read->write sequence. */
    /* number of iterations of the minor loop = nbytes / xfer_size */
    if (dma_engine.ssize < dma_engine.dsize)
        xfr_size = dest_xfer_size;
     else
        xfr_size = src_xfer_size;

    /* process the source address, READ data into the buffer*/

    /* read “xfr_size” bytes from the source */
    /* if the ssize < dsize, do multiple reads to equal the dsize */
    /* if the ssize => dsize, do a single read of source data */
    number_of_source_reads = xfer_size / src_xfer_size;

    for (number_of_source_reads) {
        dma_engine.data = read_from_amba-ahb (dma_engine.saddr, src_xfr_size);

        /* generate the next-state source address */
        /* sum the current saddr with the signed source offset */
        ns_addr = dma_engine.saddr + (int) dma_engine.soff; }

        /* if enabled, apply the power-of-2 modulo to the next-state addr */
        if (dma_engine.smod != 0)
            address_select = (1 << dma_engine.smod) - 1; }
        else
            address_select = 0xffff_ffff;

        dma_engine.saddr = ns_addr           &  address_select
                    | dma_engine.saddr & ~address_select; }
    }


    /* process the destination address, WRITE data from buffer */

    /* write “xfr_size” bytes to the destination */
    /* if the dsize < ssize, do multiple writes to equal the ssize */
    /* if the dsize => ssize, do a single write of dest data */
    number_of_dest_writes = xfer_size / dest_xfer_size;

    for (number_of_dest_writes) {
        write_to_amba-ahb (dma_engine.daddr, dest_xfr_size) = dma_engine.data;
```

```
            /* generate the next-state destination address */
            /* sum the current daddr with the signed destination offset */
            ns_addr = dma_engine.daddr + (int) dma_engine.doff;

            /* if enabled, apply the power-of-2 modulo to the next-state dest addr */
            if (dma_engine.dmod != 0)
                address_select = (1 << dma_engine.dmod) - 1;
            else
                address_select = 0xffff_ffff;

            dma_engine.daddr = ns_addr           &  address_select
                        | dma_engine.daddr & ~address_select;
        }

        /* check for a higher priority channel to service if: */
        /* 1) preemption is enabled */
        /* 2) in fixed arbitration mode */
        /* 3) a higher priority channel is requesting service */
        /* 4) not already servicing a preempting channel */
        if ((DCHPRIn.ecp = 1)  &  fixed_arbitration_mode
            higher_pri_request & ~current_channel_is_preempt)
            service_preempt_channel;

        /* the bandwidth control field determines when the next read/write occurs */
        if (dma_engine.bwc > 1)
            stall_dma_engine (1 << dma_engine.bwc);

        /* decrement the minor loop byte count */
        dma_engine.nbytes = dma_engine.nbytes - xfr_size;

    }while (dma_engine.nbytes > 0) /* end of minor inner loop */

    dma_engine.citer--;            /* decrement major loop iteration count */

    /* if the major loop is not yet exhausted, update certain TCD values in the RAM */
    if (dma_engine.citer != 0) {
        write_to_local_memory [channel].saddr = dma_engine.saddr;
        write_to_local_memory [channel].daddr = dma_engine.daddr;
        write_to_local_memory [channel].citer = dma_engine.citer;

                /* if minor loop linking is enabled, make the channel link */
        if (dma_engine.citer.e_link)
            TCD[citer.linkch].start = 1;     /* specified channel service req */

        /* check for interrupt assertion if half of the major iterations are done */
        if (dma_engine.int_half && (dma_engine.citer == (dma_engine.biter >> 1)))
            generate_interrupt (channel);

        dma_engine.active = 0;         /* clear the channel busy flag */

     }
     else { /* major loop is complete, dma_engine.citer == 0 */
       /* since the major loop is complete, perform the final address adjustments */

       /* sum the current {src,dst} addresses with “last” adjustment */
       write_to_local_memory [channel].saddr = dma_engine.saddr + dma_engine.slast;
```

```
        write_to_local_memory [channel].daddr = dma_engine.daddr + dma_engine.dlast;
        /* restore the major iteration count to the beginning value */
        write_to_local_memory [channel].citer = dma_engine.biter;

        /* check for interrupt assertion at completion of the major iteration */
        if (dma_engine.int_maj)
          generate_interrupt (channel);

         /* check if the ipd_req is to be disabled at completion of the major iteration */
        if (dma_engine.d_req)
            DMAERQ [channel] = 0;

        /* check for a scatter/gather transfer control descriptor */
        if (dma_engine.e_sg) {
         /* load new transfer control descriptor from the address defined by dlast_sga */
            write_to_local_memory [channel] =
                                    read_from_amba-ahb(dma_engine.dlast_sga,32);
        }
        if (dma_engine.major.e_link)
            TCD[major.linkch].start = 1;       /* specified channel service req */

        dma_engine.active = 0;                /* clear the channel busy flag */
        dma_engine.done = 1;                  /* set the channel done flag */

}
else { /* configuration error detected, abort the channel */
        dma_engine.error_status = error_type; /* record the error */
        dma_engine.active = 0;                /* clear the channel busy flag */
        /* check for interrupt assertion on error */
        if (dma_engine.int_err)
            generate_interrupt (channel);
}
```

For more details, consult Section 12.2.4.1, “Register Descriptions.”

12.2.3 External Signal Description

As shown in Figure 12-1, the DMA’s primary platform interfaces are the AMBA-AHB 2.v6 master bus and the IPS slave bus. The AHB 2.v6 nomenclature refers to the AMBA-AHB 2.0 AHB-Lite protocol, with AMBA V6 extensions for exclusive access support and extended cache control attributes. Additionally, the DMA inputs request signals (one per channel) from the peripherals and outputs interrupt signals (one per channel plus error indicators) to the platform’s interrupt controller. It also interfaces to the off-platform local memory array. The memory BIST (MBIST) controller is a separate module since test requirements for the RAM may vary by process technology.

12.2.4 Memory Map/Register Definition

The DMA’s programming model is partitioned into two sections, both mapped into the IPS space: the first region defines a number of registers providing control functions, while the second region corresponds to the local transfer control descriptor memory. Reading an *unimplemented* register bit or memory location will return the value of zero. Writes to an *unimplemented* register bit or memory location will be ignored. Any access to a *reserved* memory location will result in a bus error.

Many of the control registers have a bit width that matches the number of channels implemented in the module, i.e., 16-, 32- or 64-bits in size. Registers associated with a 64-channel design are implemented as two 32-bit registers, and include an “H” and “L” suffixes, signaling the “high” and “low” portions of the control function. The descriptions in this section define the 64-channel implementation. For 16- or 32-channel designs, the unused bits are not implemented: reads return zeroes, and writes are ignored.

The DMA module does not include any logic which provides access control. Rather, this function is supported using the standard access control logic provided by the AIPS controller.

Table 12-4 is a 32-bit view of the DMA’s memory map. Areas not applicable to the MAC72xx family, which has 16 channels, have been grayed out in theis table.

Table 12-4. DMA 32-bit Memory Map

DMA Offset	Register			
0x0000	DMA Control Register (DMACR)			
0x0004	DMA Error Status (DMAES)			
0x0008	DMA Enable Request High (DMAERQH, Channels 63-32)			
0x000c	DMA Enable Request Low (DMAERQL, Channels 31-00)			
0x0010	DMA Enable Error Interrupt High (DMAEEIH, Channels 63-32)			
0x0014	DMA Enable Error Interrupt Low (DMAEEIL, Channels 31-00)			
0x0018	DMA Set Enable Request (DMASERQ)	DMA Clear Enable Request (DMACERQ)	DMA Set Enable Error Interrupt (DMASEEI)	DMA Clear Enable Error Interrupt (DMACEEI)
0x001c	DMA Clear Interrupt Request (DMACINT)	DMA Clear Error (DMACERR)	DMA Set Start Bit (DMASSRT)	DMA Clear Done Status Bit (DMACDNE)
0x0020	DMA Interrupt Request High (DMAINTH, Channels 63-32)			
0x0024	DMA Interrupt Request Low (DMAINTL, Channels 31-00)			
0x0028	DMA Error High (DMAERRH, Channels 63-32)			
0x002c	DMA Error Low (DMAERRL, Channels 31-00)			
0x0030-0x00fc	Reserved			
0x0100	DMA Channel 0 Priority (DCHPRI0)	DMA Channel 1 Priority (DCHPRI1)	DMA Channel 2 Priority (DCHPRI2)	DMA Channel 3 Priority (DCHPRI3)
0x0104	DMA Channel 4 Priority (DCHPRI4)	DMA Channel 5 Priority (DCHPRI5)	DMA Channel 6 Priority (DCHPRI6)	DMA Channel 7 Priority (DCHPRI7)
0x0108	DMA Channel 8 Priority (DCHPRI8)	DMA Channel 9 Priority (DCHPRI9)	DMA Channel 10 Priority (DCHPRI10)	DMA Channel 11 Priority (DCHPRI11)
0x010c	DMA Channel 12 Priority (DCHPRI12)	DMA Channel 13 Priority (DCHPRI13)	DMA Channel 14 Priority (DCHPRI14)	DMA Channel 15 Priority (DCHPRI15)
0x0110	DMA Channel 16 Priority (DCHPRI16)	DMA Channel 17 Priority (DCHPRI17)	DMA Channel 18 Priority (DCHPRI18)	DMA Channel 19 Priority (DCHPRI19)
0x0114	DMA Channel 20 Priority (DCHPRI20)	DMA Channel 21 Priority (DCHPRI21)	DMA Channel 22 Priority (DCHPRI22)	DMA Channel 23 Priority (DCHPRI23)
0x0118	DMA Channel 24 Priority (DCHPRI24)	DMA Channel 25 Priority (DCHPRI25)	DMA Channel 26 Priority (DCHPRI26)	DMA Channel 27 Priority (DCHPRI27)
0x011c	DMA Channel 28 Priority (DCHPRI28)	DMA Channel 29 Priority (DCHPRI29)	DMA Channel 30 Priority (DCHPRI30)	DMA Channel 31 Priority (DCHPRI31)
0x0120	DMA Channel 32 Priority (DCHPRI32)	DMA Channel 33 Priority (DCHPRI33)	DMA Channel 34 Priority (DCHPRI34)	DMA Channel 35 Priority (DCHPRI35)

Table 12-4. DMA 32-bit Memory Map (Continued)

DMA Offset	Register			
0x0124	DMA Channel 36 Priority (DCHPRI36)	DMA Channel 37 Priority (DCHPRI37)	DMA Channel 38 Priority (DCHPRI38)	DMA Channel 39 Priority (DCHPRI39)
0x0128	DMA Channel 40 Priority (DCHPRI40)	DMA Channel 41 Priority (DCHPRI41)	DMA Channel 42 Priority (DCHPRI42)	DMA Channel 43 Priority (DCHPRI43)
0x012c	DMA Channel 44 Priority (DCHPRI44)	DMA Channel 45 Priority (DCHPRI45)	DMA Channel 46 Priority (DCHPRI46)	DMA Channel 47 Priority (DCHPRI47)
0x0130	DMA Channel 48 Priority (DCHPRI48)	DMA Channel 49 Priority (DCHPRI49)	DMA Channel 50 Priority (DCHPRI50)	DMA Channel 51 Priority (DCHPRI51)
0x0134	DMA Channel 52 Priority (DCHPRI52)	DMA Channel 53 Priority (DCHPRI53)	DMA Channel 54 Priority (DCHPRI54)	DMA Channel 55 Priority (DCHPRI55)
0x0138	DMA Channel 56 Priority (DCHPRI56)	DMA Channel 57 Priority (DCHPRI57)	DMA Channel 58 Priority (DCHPRI58)	DMA Channel 59 Priority (DCHPRI59)
0x013c	DMA Channel 60 Priority (DCHPRI60)	DMA Channel 61 Priority (DCHPRI61)	DMA Channel 62 Priority (DCHPRI62)	DMA Channel 63 Priority (DCHPRI63)
0x0200-0x0ffc	Reserved			
0x1000-0x11fc	TCD00-TCD15			
0x1200-0x13fc	TCD16-TCD31			
0x1400-0x15fc	TCD32-TCD47			
0x1600-0x17fc	TCD48-TCD63			

12.2.4.1 Register Descriptions

12.2.4.1.1 DMA Control Register (DMACR)

The 32-bit DMACR defines the basic operating configuration of the DMA.

The DMA arbitrates channel service requests in groups of 16 channels. The 16 channel configuration has only one group (0). Group 0 contains channels 15-0.

Arbitration within the group can be configured to use either a fixed priority or a round robin. In fixed priority arbitration, the highest priority channel requesting service is selected to execute. The priorities are assigned by the channel priority registers (see section Section 12.2.4.1.15, “DMA Channel n Priority (DCHPRIn), n = 0,..., {15,31,63}).” In round robin arbitration mode, the channel priorities are ignored and the channels are cycled through without regard to priority.

See Figure 12-2 and Table 12-5 for the DMACR definition.

Register address: DMA_Offset + 0x0000

	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																
Reset																

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	ERCA	EDBG	EBW
W																
Reset																

= Unimplemented

Figure 12-2. DMA Control Register (DMACR)

Table 12-5. DMACR Field Descriptions

Field	Description
31–3	Reserved, should be cleared.
2 ERCA	Enable Round Robin Channel Arbitration. 0 Fixed priority arbitration is used for channel selection. 1 Round robin arbitration is used for channel selection.
1 EDBG	Enable Debug. 0 The assertion of the ipg_debug input is ignored. 1 The assertion of the ipg_debug input causes the DMA to stall the start of a new channel. Executing channels are allowed to complete. Channel execution will resume when either the ipg_debug input is negated or the EDBG bit is cleared.
0 EBW	Enable Buffered Writes. 0 The bufferable write signal (hprot[2]) is not asserted during AMBA AHB writes. 1 The bufferable write signal (hprot[2]) is asserted on all AMBA AHB writes except for the last write sequence.

12.2.4.1.2 DMA Error Status (DMAES)

The DMAES register provides information concerning the last recorded channel error. Channel errors can be caused by a configuration error (an illegal setting in the transfer control descriptor or an illegal priority register setting in fixed arbitration mode) or an error termination to a bus master read or write cycle.

A configuration error is caused when the starting source or destination address, source or destination offsets, minor loop byte count and the transfer size represent an inconsistent state. The addresses and offsets must be aligned on 0-modulo-transfer_size boundaries, and the minor loop byte count must be a multiple of the source and destination transfer sizes. All source reads and destination writes must be configured to the natural boundary of the programmed transfer size respectively. In fixed arbitration mode, a configuration error is caused by any two channel priorities being equal. All channel priority levels must be unique when fixed arbitration mode is enabled. If a scatter/gather operation is enabled upon channel completion, a configuration error is reported if the scatter/gather address (dlast_sga) is not aligned on a

32 byte boundary. If minor loop channel linking is enabled upon channel completion, a configuration error is reported when the link is attempted if the TCD.citer.e_link bit does not equal the TCD.biter.e_link bit. All configuration error conditions except scatter/gather and minor loop link error are reported as the channel is activated and assert an error interrupt request, if enabled. A scatter/gather configuration error is reported when the scatter/gather operation begins at major loop completion when properly enabled. A minor loop channel link configuration error is reported when the link operation is serviced at minor loop completion.

If a system bus read or write is terminated with an error, the data transfer is stopped and the appropriate bus error flag set. In this case, the state of the channel's transfer control descriptor is updated by the dma_engine with the current source address, destination address and current iteration count at the point of the fault. When a system bus error occurs, the channel is terminated after the read or write transaction which is already pipelined after errant access, has completed. If a bus error occurs on the last read prior to beginning the write sequence, the write will execute using the data captured during the bus error. If a bus error occurs on the last write prior to switching to the next read sequence, the read sequence will execute before the channel is terminated due to the destination bus error.

The occurrence of any type of error causes the dma_engine to immediately stop, and the appropriate channel bit in the DMA Error register to be asserted. At the same time, the details of the error condition are loaded into the DMAES register. The major loop complete indicators, setting the transfer control descriptor done flag and the possible assertion of an interrupt request, are *not* affected when an error is detected. See Figure 12-3 and Table 12-6 for the DMAES definition.

Register address: DMA_Offset + 0x0004

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	VLD	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0															

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	CPE	ERRCHN[5:0]						SAE	SOE	DAE	DOE	NCE	SGE	SBE	DBE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented

Figure 12-3. DMA Error Status (DMAES) Register

Table 12-6. DMAES Field Descriptions

Field	Description
31 VLD	Logical OR of all DMAERRH and DMAERRL status bits. 0 No DMAERR bits are set. 1 At least one DMAERR bit is set indicating a valid error exists that has not been cleared.
30–15	Reserved, should be cleared.

Table 12-6. DMAES Field Descriptions (Continued)

Field	Description
14 CPE	Channel Priority Error. 0 No channel priority error. 1 The last recorded error was a configuration error in the channel priorities. All channel priorities are not unique.
13–8 ERRCHN [5:0]	Error Channel Number. The channel number of the last recorded error. (excluding CPE errors).
7 SAE	Source Address Error. 0 No source address configuration error. 1 The last recorded error was a configuration error detected in the TCD.saddr field. TCD.saddr is inconsistent with TCD.ssize.
6 SOE	Source Offset Error. 0 No source offset configuration error. 1 The last recorded error was a configuration error detected in the TCD.soff field. TCD.soff is inconsistent with TCD.ssize.
5 DAE	Destination Address Error. 0 No destination address configuration error. 1 The last recorded error was a configuration error detected in the TCD.daddr field. TCD.daddr is inconsistent with TCD.dsize.
4 DOE	Destination Offset Error. 0 No destination offset configuration error. 1 The last recorded error was a configuration error detected in the TCD.doff field. TCD.doff is inconsistent with TCD.dsize.
3 NCE	Nbytes/Citer Configuration Error. 0 No nbytes/citer configuration error. 1 The last recorded error was a configuration error detected in the TCD.nbytes or TCD.citer fields. TCD.nbytes is not a multiple of TCD.ssize and TCD.dsize, or TCD.citer is equal to zero, or TCD.citer.e_link is not equal to TCD.biter.e_link.
2 SGE	Scatter/Gather Configuration Error. 0 No scatter/gather configuration error. 1 The last recorded error was a configuration error detected in the TCD.dlast_sga field. This field is checked at the beginning of a scatter/gather operation after major loop completion if TCD.e_sg is enabled. TCD.dlast_sga is not on a 32 byte boundary.
1 SBE	Source Bus Error. 0 No source bus error. 1 The last recorded error was a bus error on a source read.
0 DBE	Destination Bus Error. 0 No destination bus error. 1 The last recorded error was a bus error on a destination write.

12.2.4.1.3 DMA Enable Request (DMAERQH, DMAERQL)

The DMAERQ{H,L} registers provide a bit map for the implemented channels {16,32,64} to enable the request signal for each channel. DMAERQH supports channels 63-32, while DMAEQRL covers channels 31-00. The state of any given channel enable is directly affected by writes to this register; it is also affected by writes to the DMASERQ and DMACERQ registers. The DMA{S,C}ERQ registers are provided so that

the request enable for a *single* channel can easily be modified without the need to perform a read-modify-write sequence to the DMAERQ{H,L} registers.

Both the DMA request input signal and this enable request flag must be asserted before a channel's hardware service request is accepted. The state of the DMA enable request flag does *not* affect a channel service request made explicitly through software or a linked channel request. See Figure 12-4 and Table 12-7 for the DMAERQ definition.

Register address: DMA_Offset + 0x0008 (DMAERQH) + 0x000c (DMAERQL)

	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																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	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																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	ERQ15	ERQ14	ERQ13	ERQ12	ERQ11	ERQ10	ERQ09	ERQ08	ERQ07	ERQ06	ERQ05	ERQ04	ERQ03	ERQ02	ERQ01	ERQ00
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 12-4. DMA Enable Request (DMAERQH, DMAERQL) Register

Table 12-7. DMAERQH, DMAERQL field Descriptions

Field	Description
31–16	Reserved, should be cleared.
15–0 ERQ*n* n = 0,... 15	Enable DMA Request *n*. 0 The DMA request signal for channel *n* is disabled. 1 The DMA request signal for channel *n* is enabled.

As a given channel completes the processing of its major iteration count, there is a flag in the transfer control descriptor that may affect the ending state of the DMAERQ bit for that channel. If the TCD.d_req

bit is set, then the corresponding DMAERQ bit is cleared, disabling the DMA request; else if the d_req bit is cleared, the state of the DMAERQ bit is unaffected.

12.2.4.1.4 DMA Enable Error Interrupt (DMAEEIH, DMAEEIL)

The DMAEEI{H,L} registers provide a bit map for the implemented channels {16,32,64} to enable the error interrupt signal for each channel. DMAEEIH supports channels 63-32, while DMAEEIL covers channels 31-00. The state of any given channel's error interrupt enable is directly affected by writes to this register; it is also affected by writes to the DMASEEI and DMACEEI registers. The DMA{S,C}EEI registers are provided so that the error interrupt enable for a *single* channel can easily be modified without the need to perform a read-modify-write sequence to the DMAEEI{H,L} registers.

Both the DMA error indicator and this error interrupt enable flag must be asserted before an error interrupt request for a given channel is asserted. See Figure 12-5 and Table 12-8 for the DMAEEI definition.

Register address: DMA_Offset + 0x0010 (DMAEEIH), + 0x0014 (DMAEEIL)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	EEI15	EEI14	EEI13	EEI12	EEI11	EEI10	EEI09	EEI08	EEI07	EEI06	EEI05	EEI04	EEI03	EEI02	EEI01	EEI00
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 12-5. DMA Enable Error Interrupt (DMAEEIH, DMAEEIL) Registers

Table 12-8. DMAEEIH, DMAEEIL Field Descriptions

Field	Description
31–16	Reserved, should be cleared.
15–0 EEI*n* n = 0,... 15	Enable Error Interrupt *n*. 0 The error signal for channel *n* does not generate an error interrupt. 1 The assertion of the error signal for channel *n* generate an error interrupt request.

12.2.4.1.5 DMA Set Enable Request (DMASERQ)

The DMASERQ register provides a simple memory-mapped mechanism to set a given bit in the DMAERQ{H,L} registers to enable the DMA request for a given channel. The data value on a register write causes the corresponding bit in the DMAERQ{H,L} register to be set. A data value of 64 to 127 (regardless of the number of implemented channels) provides a global set function, forcing the entire contents of DMAERQ{H,L} to be asserted. Reads of this register return all zeroes. See Figure 12-6 and Table 12-9 for the DMASERQ definition.

Register address: DMA_Offset + 0x0018

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W		SERQ[6:0]						
Reset		0	0	0	0	0	0	0

= Unimplemented

Figure 12-6. DMA Set Enable Request (DMASERQ) Register

Table 12-9. DMASERQ Field Descriptions

Field	Description
7	Reserved, should be cleared.
6–0 SERQ[6:0]	Set Enable Request. 0-63 Set the corresponding bit in DMAERQ{H,L} 64-127 Set all bits in DMAERQ{H,L}

12.2.4.1.6 DMA Clear Enable Request (DMACERQ)

The DMACERQ register provides a simple memory-mapped mechanism to clear a given bit in the DMAERQ{H,L} registers to disable the DMA request for a given channel. The data value on a register write causes the corresponding bit in the DMAERQ{H,L} register to be cleared. A data value of 64 to 127 (regardless of the number of implemented channels) provides a global clear function, forcing the entire contents of the DMAERQ{H,L} to be zeroed, disabling all DMA request inputs. Reads of this register return all zeroes. See Figure 12-7 and Figure 12-10 for the DMACERQ definition.

Register address: DMA_Offset + 0x0019

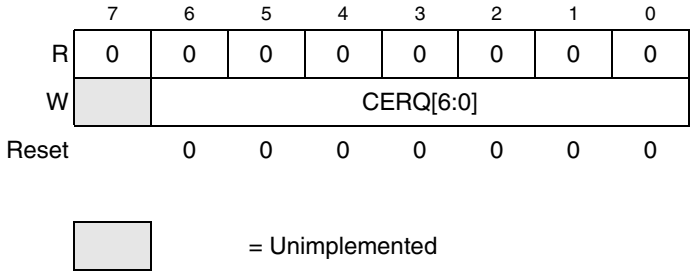


Figure 12-7. DMA Clear Enable Request (DMACERQ) Register

Table 12-10. DMACERQ Field Descriptions

Field	Description
7	Reserved, should be cleared.
6–0 CERQ[6:0]	Clear Enable Request 0-63 Clear corresponding bit in DMAERQ{H,L} 64-127 Clear all bits in DMAERQ{H,L}

12.2.4.1.7 DMA Set Enable Error Interrupt (DMASEEI)

The DMASEEI register provides a simple memory-mapped mechanism to set a given bit in the DMAEEI{H,L} registers to enable the error interrupt for a given channel. The data value on a register write causes the corresponding bit in the DMAEEI{H,L} register to be set. A data value of 64 to 127 (regardless of the number of implemented channels) provides a global set function, forcing the entire contents of DMAEEI{H,L} to be asserted. Reads of this register return all zeroes. See Figure 12-8 and Table 12-11 for the DMASEEI definition.

Register address: DMA_Offset + 0x001a

Figure 12-8. DMA Set Enable Error Interrupt (DMASEEI) Register

Table 12-11. DMASEEI Field Descriptions

Field	Description
7	Reserved, should be cleared.
6–0 SEEI[6:0]	Set Enable Error Interrupt. 0-63 Set the corresponding bit in DMAEEI{H,L} 64-127 Set all bits in DMAEEI{H,L}

12.2.4.1.8 DMA Clear Enable Error Interrupt (DMACEEI)

The DMACEEI register provides a simple memory-mapped mechanism to clear a given bit in the DMAEEI{H,L} registers to disable the error interrupt for a given channel. The data value on a register write causes the corresponding bit in the DMAEEI{H,L} register to be cleared. A data value of 64 to 127 (regardless of the number of implemented channels) provides a global clear function, forcing the entire contents of the DMAEEI{H,L} to be zeroed, disabling all DMA request inputs. Reads of this register return all zeroes. See Figure 12-7 and Table 12-12 for the DMACEEI definition.

Register address: DMA_Offset + 0x001b

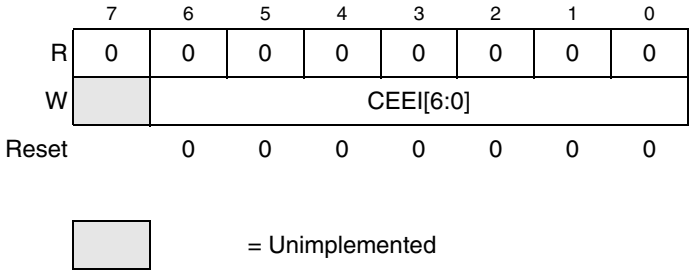


Figure 12-9. DMA Clear Enable Error Interrupt (DMACEEI) Register

Table 12-12. DMACEEI Field Descriptions

Field	Description
7	Reserved, should be cleared.
6–0 CEEI[6:0]	Clear Enable Error Interrupt. 0-63 Clear corresponding bit in DMAEEI{H,L} 64-127 Clear all bits in DMAEEI{H,L}

12.2.4.1.9 DMA Clear Interrupt Request (DMACINT)

The DMACINT register provides a simple memory-mapped mechanism to clear a given bit in the DMAINT{H,L} registers to disable the interrupt request for a given channel. The given value on a register write causes the corresponding bit in the DMAINT{H,L} register to be cleared. A data value of 64 to 127 (regardless of the number of implemented channels) provides a global clear function, forcing the entire contents of the DMAINT{H,L} to be zeroed, disabling all DMA interrupt requests. Reads of this register return all zeroes. See Figure 12-10 and Table 12-13 for the DMACINT definition.

Register address: DMA_Offset + 0x001c

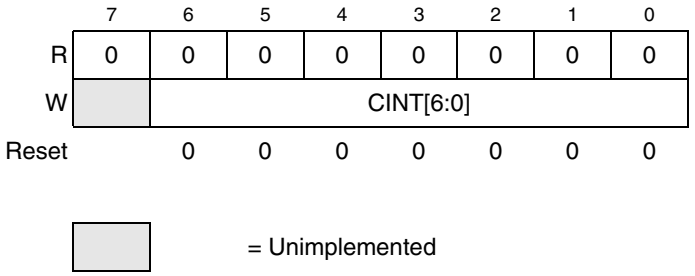


Figure 12-10. DMA Clear Interrupt Request (DMACINT) Register

Table 12-13. DMACINT Field Descriptions

Field	Description
7	Reserved, should be cleared.
6–0 CINT[6:0]	Clear Interrupt Request. 0-63 Clear the corresponding bit in DMAINT{H,L} 64-127 Clear all bits in DMAINT{H,L}

12.2.4.1.10 DMA Clear Error (DMACERR)

The DMACEER register provides a simple memory-mapped mechanism to clear a given bit in the DMAERR{H,L} registers to disable the error condition flag for a given channel. The given value on a register write causes the corresponding bit in the DMAERR{H,L} register to be cleared. A data value of 64 to 127 (regardless of the number of implemented channels) provides a global clear function, forcing the entire contents of the DMAERR{H,L} to be zeroed, clearing all channel error indicators. Reads of this register return all zeroes. See Figure 12-11 and Table 12-14 for the DMACERR definition.

Register address: DMA_Offset + 0x001d

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W		CERR[6:0]						
Reset		0	0	0	0	0	0	0

= Unimplemented

Figure 12-11. DMA Clear Error (DMACERR) Register

Table 12-14. DMACERR Field Descriptions

Field	Description
7	Reserved, should be cleared.
6–0 CERR[6:0]	Clear Error Indicator. 0-63 Clear corresponding bit in DMAERR{H,L} 64-127 Clear all bits in DMAERR{H,L}

12.2.4.1.11 DMA Set START Bit (DMASSRT)

The DMASSRT register provides a simple memory-mapped mechanism to set the START bit in the TCD of the given channel. The data value on a register write causes the START bit in the corresponding Transfer Control Descriptor to be set. A data value of 64 to 127 (regardless of the number of implemented channels) provides a global set function, forcing all START bits to be set. Reads of this register return all zeroes. See Table 12-28 for the TCD START bit definition.

Register address: DMA_Offset + 0x001e

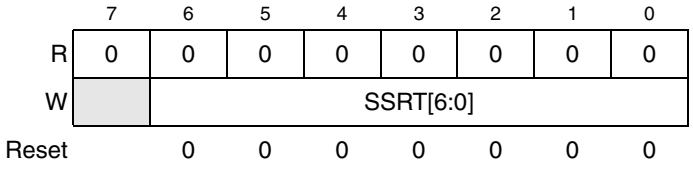


Figure 12-12. DMA Set START Bit (DMASSRT) Register

Table 12-15. DMASSRT Field Descriptions

Field	Descriptions
7	Reserved, should be cleared.
6–0 SSRT[6:0]	Set START Bit (Channel Service Request). 0-63 Set the corresponding channel's TCD.start 64-127 Set all TCD.start bits

12.2.4.1.12 DMA Clear DONE Status (DMACDNE)

The DMACDNE register provides a simple memory-mapped mechanism to clear the DONE bit in the TCD of the given channel. The data value on a register write causes the DONE bit in the corresponding Transfer Control Descriptor to be cleared. A data value of 64 to 127 (regardless of the number of implemented channels) provides a global clear function, forcing all DONE bits to be cleared. Reads of this register return all zeroes. See Table 12-28 for the TCD DONE bit definition.

Register address: DMA_Offset + 0x001f

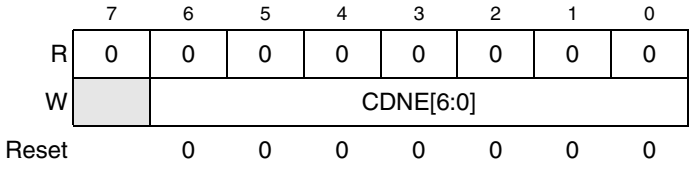


Figure 12-13. DMA Clear DONE Status (DMACDNE) Register

Table 12-16. DMACDNE Field Descriptions

Field	Description
7	Reserved, should be cleared.
6–0 CDNE[6:0]	Clear DONE Status Bit. 0-63 Clear the corresponding channel's DONE bit 64-127 Clear all TCD DONE bits

12.2.4.1.13 DMA Interrupt Request (DMAINTH, DMAINTL)

The DMAINT{H,L} registers provide a bit map for the implemented channels {16,32,64} signaling the presence of an interrupt request for each channel. DMAINTH supports channels 63-32, while DMAINTL covers channels 31-00. The dma_engine signals the occurrence of a programmed interrupt upon the completion of a data transfer as defined in the transfer control descriptor by setting the appropriate bit in this register. The outputs of this register are directly routed to the platform's interrupt controller. During the execution of the interrupt service routine associated with any given channel, it is software's responsibility to clear the appropriate bit, negating the interrupt request. Typically, a write to the DMACINT register in the interrupt service routine is used for this purpose.

The state of any given channel's interrupt request is directly affected by writes to this register; it is also affected by writes to the DMACINT register. On writes to the DMAINT, a one in any bit position clears the corresponding channel's interrupt request. A zero in any bit position has no affect on the corresponding channel's current interrupt status. The DMACINT register is provided so the interrupt request for a *single* channel can easily be cleared without the need to perform a read-modify-write sequence to the DMAINT{H,L} registers. See Figure 12-14 and Table 12-17 for the DMAINT definition.

Register address: DMA_Offset + 0x0020 (DMAINTH), +0x0024 (DMAINTL)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	INT15	INT14	INT13	INT12	INT11	INT10	INT09	INT08	INT07	INT06	INT05	INT04	INT03	INT02	INT01	INT00
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented

Figure 12-14. DMA Interrupt Request (DMAINTH, DMAINTL) Registers

Table 12-17. DMAINTH, DMAINTL Field Descriptions

Field	Description
31–16	Reserved, should be cleared.
15–0 INT*n*, n = 0,... 15	DMA Interrupt Request *n.* 0 The interrupt request for channel *n* is cleared. 1 The interrupt request for channel *n* is active.

12.2.4.1.14 DMA Error (DMAERRH, DMAERRL)

The DMAERR{H,L} registers provide a bit map for the implemented channels {16,32,64} signaling the presence of an error for each channel. DMAERRH supports channels 63-32, while DMAERRL covers channels 31-00. The dma_engine signals the occurrence of a error condition by setting the appropriate bit in this register. The outputs of this register are enabled by the contents of the DMAEEI register, then logically summed across groups of 16, 32 and 64 channels to form several group error interrupt requests which is then routed to the platform's interrupt controller. During the execution of the interrupt service routine associated with any DMA errors, it is software's responsibility to clear the appropriate bit, negating the error interrupt request. Typically, a write to the DMACERR register in the interrupt service routine is used for this purpose. Recall the normal DMA channel completion indicators, setting the transfer control descriptor done flag and the possible assertion of an interrupt request, are *not* affected when an error is detected.

The contents of this register can also be polled and a non-zero value indicates the presence of a channel error, regardless of the state of the DMAEEI register. The state of any given channel's error indicators is affected by writes to this register; it is also affected by writes to the DMACERR register. On writes to the DMAERR, a one in any bit position clears the corresponding channel's error status. A zero in any bit position has no affect on the corresponding channel's current error status. The DMACERR register is provided so the error indicator for a *single* channel can easily be cleared. See Figure 12-15 and Table 12-18 for the DMAERR definition.

Register address: DMA_Offset + 0x0028 (DMAERRH), +0x002c (DMAERRL)

	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																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	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																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	ERR15	ERR14	ERR13	ERR12	ERR11	ERR10	ERR09	ERR08	ERR07	ERR06	ERR05	ERR04	ERR03	ERR02	ERR01	ERR00
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented

Figure 12-15. DMA Error (DMAERRH, DMAERRL) Registers

Table 12-18. DMAERRH, DMAERRL Field Descriptions

Field	Description
31–16	DMA Error *n*. 0 An error in channel *n* has not occurred. 1 An error in channel *n* has occurred.
15–0 ERRn, n = 0,... 15	DMA Error *n*. 0 An error in channel *n* has not occurred. 1 An error in channel *n* has occurred.

12.2.4.1.15 DMA Channel n Priority (DCHPRIn), n = 0,..., {15,31,63}

When the fixed-priority channel arbitration mode is enabled (DMACR[ERCA] = 0), the contents of these registers define the unique priorities associated with each channel. The channel priorities are evaluated by numeric value, i.e., 0 is the lowest priority, 1 is the next higher priority, then 2, 3, etc. Software must program the channel priorities with unique values, otherwise a configuration error will be reported. The

range of the priority value is limited to the values of 0 through 15. See Figure 12-2 and Table 12-5 for the DMACR definition.

Channel preemption is enabled on a per channel basis by setting the ECP bit in the DCHPRIn register. Channel preemption allows the executing channel's data transfers to be temporarily suspended in favor of starting a higher priority channel. Once the preempting channel has completed all of its minor loop data transfers, the preempted channel is restored and resumes execution. After the restored channel completes one read/write sequence, it is again eligible for preemption. If any higher priority channel is requesting service, the restored channel will be suspended and the higher priority channel will be serviced. Nested preemption (attempting to preempt a preempting channel) is not supported. Once a preempting channel begins execution, it cannot be preempted. Preemption is only available when fixed arbitration is selected for channel arbitration modes. See Figure 12-16 and Table 12-19 for the DCHPRIn definition.

Register address: DMA_Offset + 0x100 + n

	7	6	5	4	3	2	1	0
R	ECP	0	0	0	CHPRI[3:0]			
W								
Reset	0		*	*	*	*	*	*

= Unimplemented,

* = defaults to channel number (n) after reset

Figure 12-16. DMA Channel *n* Priority (DCHPRI*n*) Register

Table 12-19. DCHPRI*n* Field Descriptions

Field	Description
7 ECP	Enable Channel Preemption. 0 Channel n cannot be suspended by a higher priority channel's service request. 1 Channel n can be temporarily suspended by the service request of a higher priority channel.
6-4	Reserved, should be cleared.
3–0 CHPRI[3:0]	Channel *n* Arbitration Priority. Channel priority when fixed-priority arbitration is enabled.

12.2.4.1.16 Transfer Control Descriptor (TCD)

Each channel requires a 32-byte transfer control descriptor for defining the desired data movement operation. The TCD structure was previously discussed in detail in Section 12.2.2, "Features." The channel descriptors are stored in the local memory in sequential order: channel 0, channel 1, ... channel [n-1]. The definitions of the TCD are presented as eight 32-bit values. Table 12-20 is a 32-bit view of the basic TCD structure.

Table 12-20. TCDn 32-bit Memory Structure

DMA Offset	TCDn Field
0x1000 + (32 x n) + 0x00	Source Address (saddr)

Table 12-20. TCDn 32-bit Memory Structure

0x1000 + (32 x n) + 0x04	Transfer Attributes (smod, ssize, dmod, dsize)	Signed Source Address Offset (soff)
0x1000 + (32 x n) + 0x08	Inner “Minor” Byte Count (nbytes)	
0x1000 + (32 x n) + 0x0c	Last Source Address Adjustment (slast)	
0x1000 + (32 x n) + 0x10	Destination Address (daddr)	
0x1000 + (32 x n) + 0x14	Current “Major” Iteration Count (citer)	Signed Destination Address Offset (doff)
0x1000 + (32 x n) + 0x18	Last Destination Address Adjustment/Scatter Gather Address (dlast_sga)	
0x1000 + (32 x n) + 0x1c	Beginning “Major” Iteration Count (biter)	Channel Control/Status (bwc, major.linkch, done, active, major.e_link, e_sg, d_req, int_half, int_maj, start)

Figure 12-17 and Table 12-21 define word 0 of the TCDn structure, the saddr field.

Register address: DMA_Offset + 0x1000 + (32 x n) + 0x00

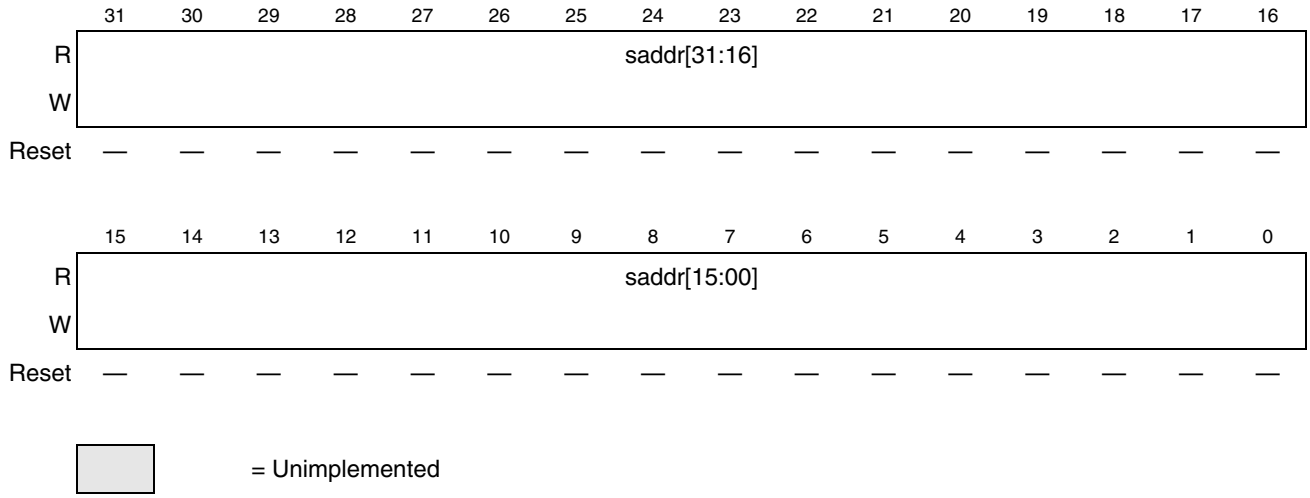


Figure 12-17. TCDn Word 0 (TCDn.saddr) Field

Table 12-21. TCDn Word 0 (TCDn.saddr) Field Description

Field	Description
31–0 saddr[31:0]	Source address. Memory address pointing to the source data.

Figure 12-18 and Table 12-22 define word 1 of the TCDn structure, the soff and transfer attribute fields.

Register address: DMA_Offset + 0x1000 + (32 x n) + 0x04

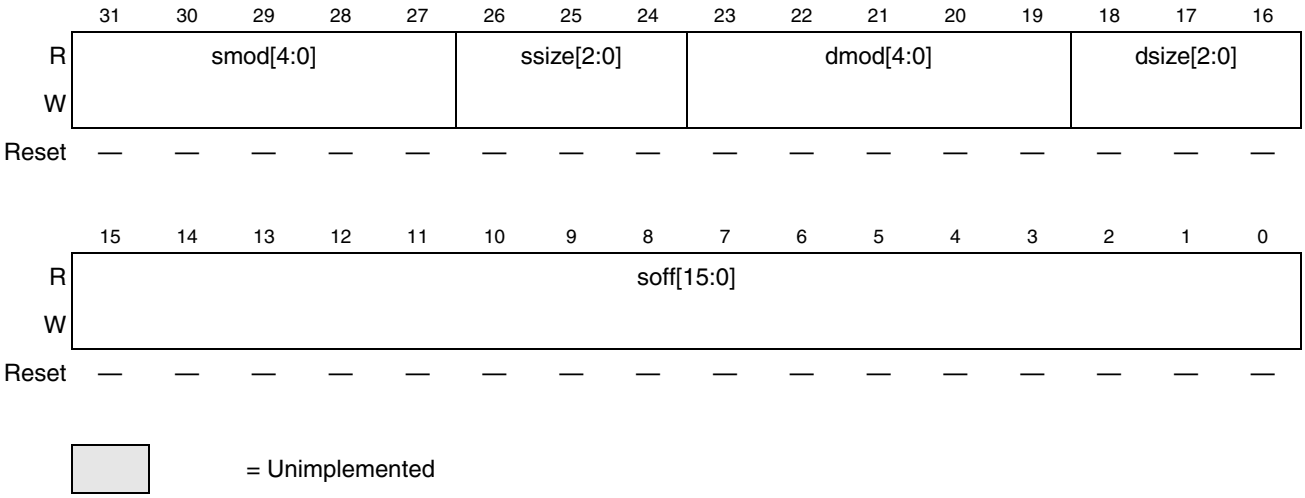


Figure 12-18. TCDn Word 1 (TCDn.{soff,smod,ssize,dmod,dsize}) Fields

Table 12-22. TCDn Word 1 (TCDn.{smod,ssize,dmod,dsize,soff}) Field Descriptions

Field	Description
31–27 smod[4:0]	Source address modulo. 0 Source address modulo feature is disabled. non-0 The value defines a specific address bit which is selected to be either the value after saddr + soff calculation is performed or the original register value. This feature provides the ability to easily implement a circular data queue. For data queues requiring power-of-2 “size” bytes, the queue should be based at a 0-modulo-size address and the smod field set to the appropriate value to freeze the upper address bits. The bit select is defined as ((1 << smod[4:0]) - 1) where a resulting 1 in a bit location selects the next state address for the corresponding address bit location and a 0 selects the original register value for the corresponding address bit location. For this application, the soff is typically set to the transfer size to implement post-increment addressing with the smod function constraining the addresses to a 0-modulo-size range.
26–24 ssize[2:0]	Source data transfer size. 000 8-bit 001 16-bit 010 32-bit 011 64-bit 100 16-byte (32-bit implementations only) 101 32-byte (if supported by the platform) 110 Reserved 111 Reserved The attempted specification of a 64-bit source size in a 32-bit AMBA AHB bus implementation produces a configuration error. Likewise, the attempted specification of a 16-byte source size in a 64-bit AMBA AHB bus implementation generates a configuration error. The attempted specification of a 32-byte burst on platforms that do not support such a transfer type will result in a configuration error.
23–19 dmod[4:0]	Destination address modulo. See the smod[5:0] definition.
18–16 dsize[2:0]	Destination data transfer size. See the ssize[2:0] definition.
15–0 soff[15:0]	Source address signed offse. Sign-extended offset applied to the current source address to form the next-state value as each source read is completed.

Figure 12-19 and Table 12-23 define word 2 of the TCDn structure, the nbytes field.

Register address: DMA_Offset + 0x1000 + (32 x n) + 0x08

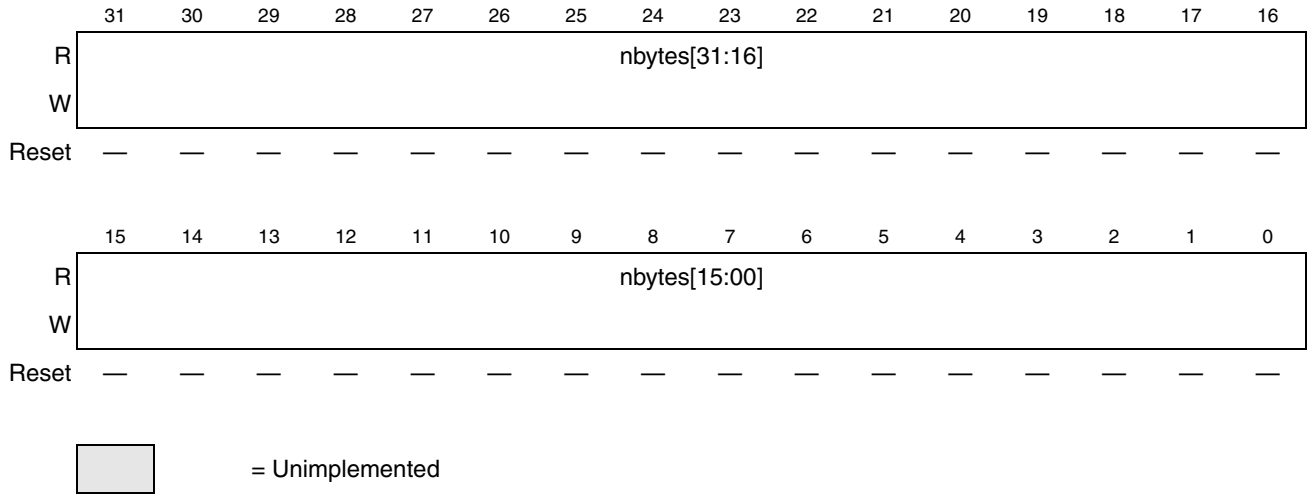


Figure 12-19. TCDn Word 2 (TCDn.nbytes) Field

Table 12-23. TCDn Word 2 (TCDn.nbytes) Field Description

Field	Description
31–0 nbytes[31:0]	Inner “minor” byte transfer count. Number of bytes to be transferred in each service request of the channel. As a channel is activated, the contents of the appropriate TCD is loaded into the dma_engine, and the appropriate reads and writes performed until the complete byte transfer count has been transferred. This is an indivisible operation and cannot be stalled or halted. Once the minor count is exhausted, the current values of the saddr and daddr are written back into the local memory, the major iteration count is decremented and restored to the local memory. If the major iteration count is completed, additional processing is performed. The nbytes value 0x0000_0000 is interpreted as 0x1_0000_0000, thus specifying a 4GB transfer.

Figure 12-20 and Table 12-24 define word 3 of the TCDn structure, the slast field.

Register address: DMA_Offset + 0x1000 + (32 x n) + 0x0c

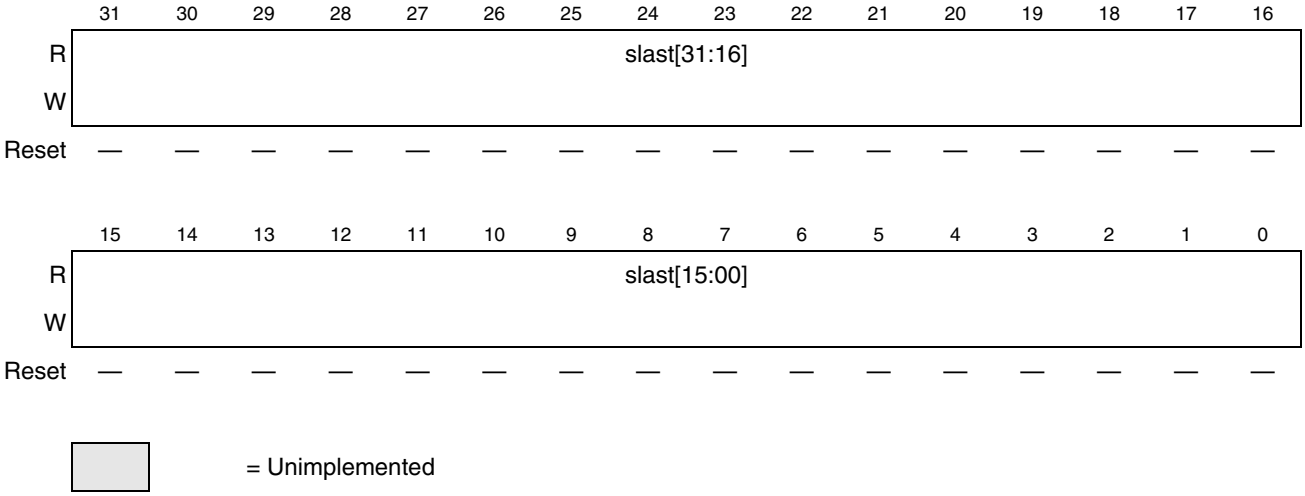


Figure 12-20. TCDn Word 3 (TCDn.slast) Field

Table 12-24. TCDn Word 3 (TCDn.slast) Field Descriptions

Field	Description
31–0 slast[31:0]	Last source address adjustment. Adjustment value added to the source address at the completion of the outer major iteration count. This value can be applied to “restore” the source address to the initial value, or adjust the address to reference the next data structure.

Figure 12-21 and Table 12-25 define word 4 of the TCDn structure, the daddr field.

Register address: DMA_Offset + 0x1000 + (32 x n) + 0x10

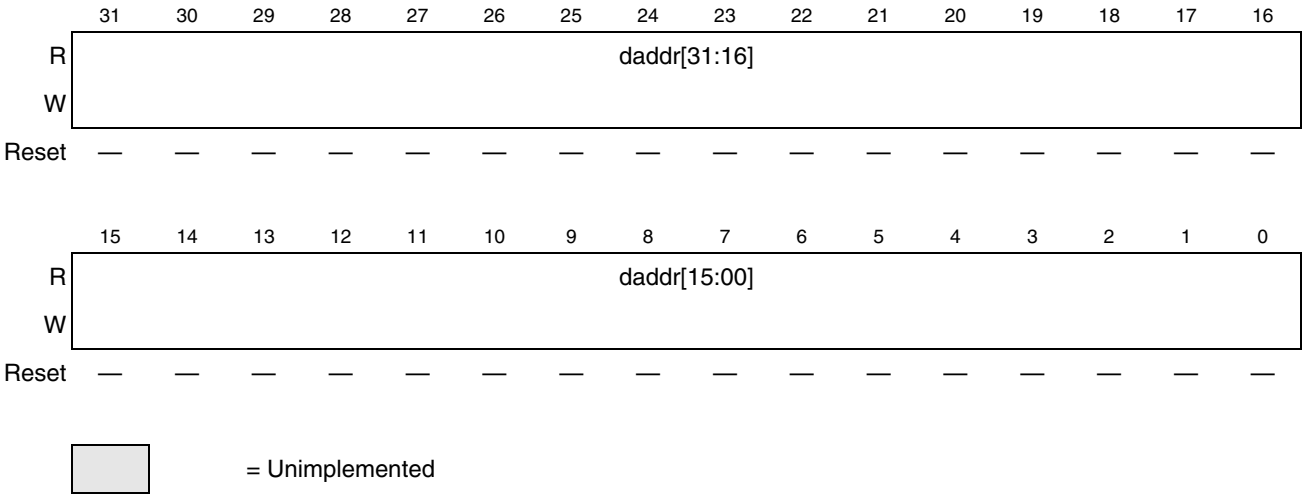


Figure 12-21. TCDn Word 4 (TCDn.daddr) Field

Table 12-25. TCDn Word 4 (TCDn.daddr) Field Descriptions

Field	Description
31–0 daddr[31:0]	Destination address. Memory address pointing to the destination data.

Figure 12-22 and Table 12-26 define word 5 of the TCDn structure, the citer and doff fields.

Register address: DMA_Offset + 0x1000 + (32 x n) + 0x14

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	citer. e_link	citer[14:9] or citer.linkch[5:0]						citer[8:0]								
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	doff[15:0]															
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

= Unimplemented

Figure 12-22. TCDn Word 5 (TCDn.{citer,doff}) Fields

Table 12-26. TCDn Word 5 (TCDn.{doff,citer}) Field Descriptions

Field	Description
31 citer.e_link	Enable channel-to-channel linking on minor loop complete. As the channel completes the inner minor loop, this flag enables the linking to another channel, defined by citer.linkch[5:0]. The link target channel initiates a channel service request via an internal mechanism that sets the TCD.start bit of the specified channel. If channel linking is disabled, the citer value is extended to 15 bits in place of a link channel number. If the "major" loop is exhausted, this link mechanism is suppressed in favor of the major.e_link channel linking. *This bit must be equal to the biter.e_link bit otherwise a configuration error will be reported.* 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled.
30–25 citer[14:9] or citer.linkch [5:0]	Current “major” iteration count or Link channel number. If (TCD.citer.e_link = 0) then No channel-to-channel linking (or chaining) is performed after the inner "minor" loop is exhausted. TCD word 5, bits [30:25] are used to form a 15 bit citer field. else After the "minor" loop is exhausted, the dma_engine initiates a channel service request at the channel defined by citer.linkch[5:0] by setting that channel’s TCD.start bit. The value contained in citer.linkch[5:0] must not exceed the number of implemented channels.

Table 12-26. TCDn Word 5 (TCDn.{doff,citer}) Field Descriptions (Continued)

Field	Description
24–16 citer[8:0]	Current “major” iteration count. This 9 or 15-bit count represents the current major loop count for the channel. It is decremented each time the minor loop is completed and updated in the transfer control descriptor memory. Once the major iteration count is exhausted, the channel performs a number of operations (e.g., final source and destination address calculations), optionally generating an interrupt to signal channel completion before reloading the citer field from the beginning iteration count (biter) field. When the citer field is initially loaded by software, it must be set to the same value as that contained in the biter field. If the channel is configured to execute a single service request, the initial values of biter and citer should be 0x0001.
15–0 doff[15:0]	Destination address signed offset. Sign-extended offset applied to the current destination address to form the next-state value as each destination write is completed.

Figure 12-23 and Table 12-27 define word 6 of the TCDn structure, the dlast_sga field.

Register address: DMA_Offset + 0x1000 + (32 x n) + 0x18

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	dlast_sga[31:16]															
W																
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	dlast_sga[15:00]															
W																
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

= Unimplemented

Figure 12-23. TCDn Word 6 (TCDn.dlast_sga) Field

Table 12-27. TCDn Word 6 (TCDn.dlast_sga) Field Descriptions

Field	Description
31–0 dlast_sga [31:0]	Last destination address adjustment or the memory address for the next transfer control descriptor to be loaded into this channel (scatter/gather). If (TCD.e_sg = 0) then Adjustment value added to the destination address at the completion of the outer major iteration count. This value can be applied to “restore” the destination address to the initial value, or adjust the address to reference the next data structure. else This address points to the beginning of a 0-modulo-32 region containing the next transfer control descriptor to be loaded into this channel. This channel reload is performed as the major iteration count completes. The scatter/gather address must be 0-modulo-32, else a configuration error is reported.

Figure 12-24 and Table 12-28 define word 7 of the TCDn structure, the biter and control/status fields.

Register address: DMA_Offset + 0x1000 + (32 x n) + 0x1c

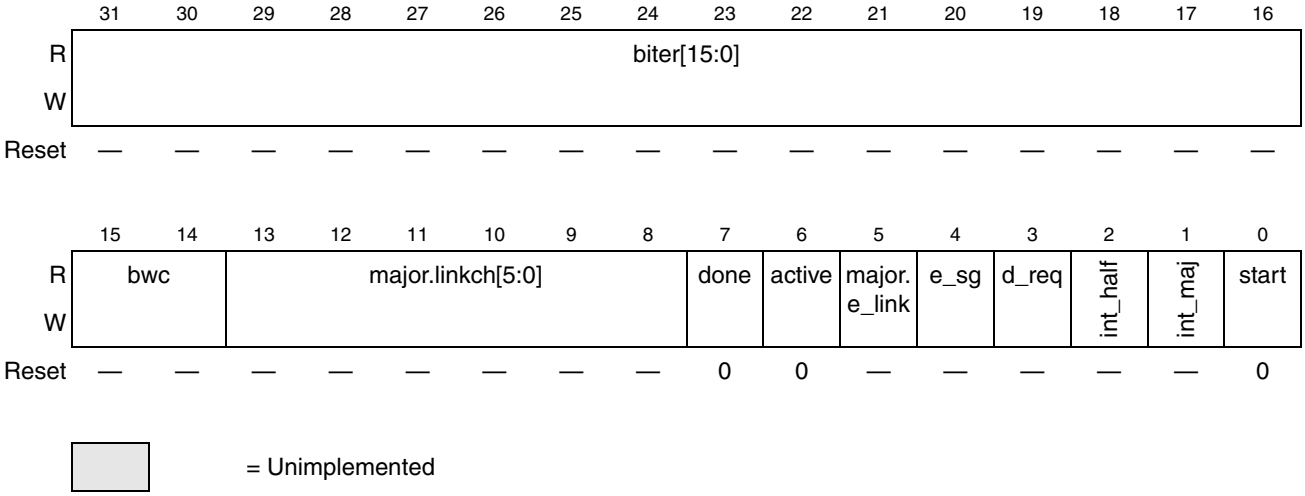


Figure 12-24. TCDn Word 7 (TCDn.{biter,control/status}) Fields

Table 12-28. TCDn Word 7 (TCDn.{biter, control/status}) Field Descriptions

Field	Description
biter.e_link	Enable channel-to-channel linking on minor loop complete. This is the initial value copied into the citer.e_link field when the major loop is completed. The citer.e_link field controls channel linking during channel execution. *This bit must be equal to the citer.e_link bit otherwise a configuration error will be reported.* 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled.

biter[14:9] or biter.linkch[5:0]	Beginning “major” iteration count or Beginning Link channel number. This is the initial value copied into the citer field or citer.linkch field when the major loop is completed. The citer fields controls the interation count and link ing during channel execution.
	if (TCD.biter.e_link = 0) then
	No channel-to-channel linking (or chaining) is performed after the inner "minor" loop is exhausted. TCD word 5, bits [30:25] are used to form a 15 bit biter field.
	else
	After the "minor" loop is exhausted, the dma_engine initiates a channel service request at the channel defined by biter.linkch[5:0] by setting that channel’s TCD.start bit.
	The value contained in biter.linkch[5:0] must not exceed the number of implemented channels

biter[8:0]	Beginning “major” iteration count. This is the initial value copied into the citer field or citer.linkch field when the major loop is completed. The citer fields controls the interation count and link ing during channel execution. This 9 or 15-bit count represents the beginning major loop count for the channel. As the major iteration count is exhausted, the contents of the entire 16 bit biter entry is reloaded into the 16 bit citer entry. When the biter field is initially loaded by software, it must be set to the same value as that contained in the citer field. If the channel is configured to execute a single service request, the initial values of biter and citer should be 0x0001.

Table 12-28. TCDn Word 7 (TCDn.{biter, control/status}) Field Descriptions (Continued)

Field	Description
bwc[1:0]	Bandwidth control. This two-bit field provides a mechanism to effectively throttle the amount of bus bandwidth consumed by the DMA. In general, as the DMA processes the inner minor loop, it continuously generates read/write, read/write, ... sequences until the minor count is exhausted. This field forces the DMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the platform's cross-bar arbitration switch. To minimize start-up latency, bandwidth control stalls are suppressed for the first two AHB bus cycles and after the last write of each minor loop. The dynamic priority elevation setting elevates the priority of the DMA as seen by the cross-bar arbitration switch for the executing channel. Dynamic priority elevation is suppressed during the first two AHB bus cycles. 00 No dma_engine stalls 01 Dynamic priority elevation 10 dma_engine stalls for 4 cycles after each r/w 11 dma_engine stalls for 8 cycles after each r/w
major.linkch[5:0]	Link channel number. If TCD.major.e_link = 0) then No channel-to-channel linking (or chaining) is performed after the outer "major" loop counter is exhausted. else After the "major" loop counter is exhausted, the dma_engine initiates a channel service request at the channel defined by major.linkch[5:0] by setting that channel's TCD.start bit. The value contained in major.linkch[5:0] must not exceed the number of implemented channels.
done	Channel done. This flag indicates the DMA has completed the outer major loop. It is set by the dma_engine as the citer count reaches zero; it is cleared by software, or the hardware when the channel is activated. This bit must be cleared in order to write the major.e_link or e_sg bits.
active	Channel active. This flag signals the channel is currently in execution. It is set when channel service begins, and is cleared by the dma_engine as the inner minor loop completes or if any error condition is detected.
major.e_link	Enable channel-to-channel linking on major loop complete. As the channel completes the outer major loop, this flag enables the linking to another channel, defined by major.linkch[5:0]. The link target channel initiates a channel service request via an internal mechanism that sets the TCD.start bit of the specified channel. *To support the dynamic linking coherency model, this field is forced to zero when written to while the TCD.done bit is set.* 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled.
e_sg	Enable scatter/gather processing. As the channel completes the outer major loop, this flag enables scatter/gather processing in the current channel. If enabled, the dma_engine uses dlast_sga as a memory pointer to a 0-modulo-32 address containing a 32-byte data structure which is loaded as the transfer control descriptor into the local memory. *To support the dynamic scatter/gather coherency model, this field is forced to zero when written to while the TCD.done bit is set.* 0 The current channel's TCD is "normal" format. 1 The current channel's TCD specifies a scatter gather format. The dlast_sga field provides a memory pointer to the next TCD to be loaded into this channel after the outer major loop completes its execution.
d_req	Disable request. If this flag is set, the DMA hardware automatically clears the corresponding DMAERQ bit when the current major iteration count reaches zero. 0 The channel's DMAERQ bit is not affected. 1 The channel's DMAERQ bit is cleared when the outer major loop is complete.

Table 12-28. TCDn Word 7 (TCDn.{biter, control/status}) Field Descriptions (Continued)

Field	Description
int_half	Enable an interrupt when major counter is half complete. If this flag is set, the channel generates an interrupt request by setting the appropriate bit in the DMAINT register when the current major iteration count reaches the halfway point. Specifically, the comparison performed by the dma_engine is (citer == (biter >> 1)). This halfway point interrupt request is provided to support double-buffered schemes or other types of data movement where the processor needs an early indication of the transfer's progress. The halfway complete interrupt is disabled when biter values are less than two. 0 The half-point interrupt is disabled. 1 The half-point interrupt is enabled.
int_maj	Enable an interrupt when major iteration count completes. If this flag is set, the channel generates an interrupt request by setting the appropriate bit in the DMAINT register when the current major iteration count reaches zero. 0 The end-of-major loop interrupt is disabled. 1 The end-of-major loop interrupt is enabled.
start	Channel start. If this flag is set, the channel is requesting service. The DMA hardware automatically clears this flag after the channel begins execution. 0 The channel is not explicitly started. 1 The channel is explicitly started via a software initiated service request.

12.2.5 DMA Performance

This section addresses the performance of the DMA module, focusing on two separate metrics. In the traditional data movement context, performance is best expressed as the peak data transfer rates achieved using the DMA. In most implementations, this transfer rate is limited by the speed of the source and destination address spaces. In a second context where device-paced movement of single data values to/from peripherals is dominant, a measure of the requests which can be serviced in a fixed time is a more interesting metric. In this environment, the speed of the source and destination address spaces remains important, but the microarchitecture of the DMA also factors significantly into the resulting metric.

The peak transfer rates for several different source and destination transfers are shown in Figure 12-29. The following assumptions apply to Figure 12-29 and Figure 12-30:

- Platform SRAM can be accessed with zero wait-states when viewed from the AMBA-AHB data phase
- All IPS reads require two wait-states, and IPS writes three wait-states, again viewed from the system bus data phase
- All IPS accesses are 32-bits in size

Table 12-29 presents a peak transfer rate comparison, measured in MBytes per sec.

Table 12-29. DMA Peak Transfer Rates [MBytes/sec]

Platform Speed, Width	Platform SRAM-to-Platform SRAM	32b IPS-to-Platform SRAM	Platform SRAM-to-32b IPS
66.7 MHz, 32b	133.3	66.7	53.3

where the Platform_SRAM-to-Platform_SRAM transfers occur at the native platform datapath width, i.e., either 32- or 64-bits per access. For all transfers involving the IPS bus, 32-bit transfer sizes are used. In all cases, the transfer rate includes the time to read the source plus the time to write the destination.

The second performance metric is a measure of the number of DMA requests which can be serviced in a given amount of time. For this metric, it is assumed the peripheral request causes the channel to move a single IPS-mapped operand to/from the platform SRAM. The same timing assumptions used in the previous example apply to this calculation. In particular, this metric also reflects the time required to activate the channel. The DMA design supports the following hardware service request sequence:

- Cycle 1: ipd_req[n] is asserted.
- Cycle 2: The ipd_req[n] is registered locally in the DMA module and qualified. (TCD.start bit initiated requests start at this point with the registering of the IPS write to TCD word7).
- Cycle 3: Channel arbitration begins.
- Cycle 4: Channel arbitration completes. The transfer control descriptor local memory read is initiated.
- Cycle 5 - 6: The first two parts of the activated channel's TCD is read from the local memory. The memory width to the dma_engine is 64 bits, so the entire descriptor can be accessed in four cycles.
- Cycle 7: The first AMBA-AHB read cycle is initiated, as the third part of the channel's TCD is read from the local memory. Depending on the state of the platform's crossbar switch, arbitration at the system bus may insert an additional cycle of delay here.
- Cycle 8 - ?: The last part of the TCD is read in. This cycle represents the 1st data phase for the read, and the address phase for the destination write.

  The exact timing from this point is a function of the response times for the channel's read and write accesses. In this case of an IPS read and a platform SRAM write, the combined data phase time is 4 cycles. For an SRAM read and IPS write, it is 5 cycles.
- Cycle ?+1: This cycle represents the data phase of the last destination write.
- Cycle ?+2: The dma_engine completes the execution of the inner minor loop and prepares to write back the required TCDn fields into the local memory. TCD word7 is read and checked for channel linking or scatter/gather requests.
- Cycle ?+3: The appropriate fields in the first part of the TCDn are written back into the local memory.
- Cycle ?+4: The fields in the second part of the TCDn are written back into the local memory. This cycle coincides with the next channel arbitration cycle start.
- Cycle ?+5: The next channel to be activated performs the read of the first part of its TCD from the local memory. This is equivalent to Cycle 4 for the first channel's service request.

Assuming zero wait states on the AHB system bus, DMA requests can be processed every 9 cycles. Assuming an average of the access times associated with IPS-to-SRAM (4 cycles) and SRAM-to-IPS (5 cycles), DMA requests can be processed every 11.5 cycles (4 + (4+5)/2 + 3). This is the time from Cycle 4 to Cycle "?+5". The resulting peak request rate, as a function of the platform frequency, is shown in Table 12-30. This metric represents millions of requests per second.

Table 12-30. DMA Peak Request Rate [MReq/sec]

Platform Speed	Request Rate (zero wait state)	Request Rate (with wait states)
66.6 MHz	7.4	5.8

A general formula to compute the peak request rate (with overlapping requests) is:

```
PEAKreq = freq / [ entry + (1 + read_ws) + (1 + write_ws) + exit ]
where:
PEAKreq - peak request rate
freq - platform frequency
entry - channel startup (4 cycles)
read_ws - wait states seen during the system bus read data phase
write_ws - wait states seen during the system bus write data phase
exit - channel shutdown (3 cycles)
```

For example: consider a platform with the following characteristics:

- Platform SRAM can be accessed with one wait-state when viewed from the AMBA-AHB data phase
- All IPS reads require two wait-states, and IPS writes three wait-states, again viewed from the system bus data phase
- Platform operates at 150 MHz

For an SRAM to IPS transfer,

$$PEAKreq = \frac{150\ MHz}{[4 + (1 + 1) + (1 + 3) + 3]cycles} = 11.5\ M\ req/sec \qquad \textit{Eqn. 12-1}$$

For an IPS to SRAM transfer,

$$PEAKreq = \frac{150\ MHz}{[4 + (1 + 2) + (1 + 1) + 3]cycles} = 12.5\ M\ req/sec \qquad \textit{Eqn. 12-2}$$

Assuming an even distribution of the two transfer types, the average Peak Request Rate would be:

$$PEAKreq = \frac{11.5\ M\ req/sec + 12.5\ M\ req/sec}{2} = 12.0\ M\ req/sec \qquad \textit{Eqn. 12-3}$$

The minimum number of cycles to perform a single read/write, zero wait states on the system bus, from a cold start (where no channel is executing, DMA is idle) are:

- 11 cycles for a software (TCD.start bit) request
- 12 cycles for a hardware (ipd_req signal) request

Two cycles account for the arbitration pipeline and one extra cycle on the hardware request resulting from the internal registering of the ipd_req signals. For the peak request rate calculations above, the arbitration and request registering is absorbed in or overlap the previous executing channel.

Note: When channel linking or scatter/gather is enabled, a two cycle delay is imposed on the next channel selection and startup. This allows the link channel or the scatter/gather channel to be eligible and considered in the arbitration pool for next channel selection.

12.2.6 Initialization/Application Information

12.2.6.1 DMA Initialization

A typical initialization of the DMA would be:

1. write the DMACR register if a configuration other than the default is desired,
2. write the channel priority levels into the DCHPRIn registers if a configuration other than the default is desired,
3. enable error interrupts in the DMAEEI registers if so desired,
4. write the 32 byte TCD for each channel that may request service,
5. enable any hardware service requests via the DMAERQ register,
6. request channel service by either software (setting the TCD.start bit) or by hardware (slave device asserting its ipd_req signal).

Once any channel requests service, a channel is selected for execution based on the arbitration and priority levels written into the programmer's model. The dma_engine will read the entire TCD for the selected channel into its internal address path module. As the TCD is being read, the first transfer is initiated on the AHB bus unless a configuration error is detected. Transfers from the source (as defined by the source address, TCD.saddr) to the destination (as defined by the destination address, TCD.daddr) continue until the specified number of bytes (TCD.nbytes) have been transferred. When the transfer is complete, the dma_engine's local TCD.saddr, TCD.daddr, and TCD.citer are written back to the main TCD memory and any minor loop channel linking is performed, if enabled. If the major loop is exhausted, further post processing is executed, i.e. interrupts, major loop channel linking, and scatter/gather operations, if enabled.

12.2.6.2 DMA Programming Errors

The DMA performs various tests on the Transfer Control Descriptor to verify consitency in the descriptor data. Most programming errors are reported on a per channel basis with the exception of Channel Priority Error, CPE in the DMAES register respectively.

For all error types other than Channel Priority Errors, the channel number causing the error is recorded in the DMAES register. If the error source is not removed before the next activation of the problem channel, the error will be detected and recorded again.

In general, if priority levels are not unique, the highest channel priority that has an active request will be selected, but the lowest numbered channel with that priority will be selected by arbitration and executed by the dma_engine. The hardware service request handshake signals, error interrupts and error reporting will be associated with the selected channel.

12.2.6.3 DMA Arbitration Mode Considerations

12.2.6.3.1 Fixed Channel Arbitration

In this mode, the channel service request from the highest priority channel will be selected to execute.

Preemption is available in this scenario only.

12.2.6.3.2 Round Robin Channel Arbitration

Channels are serviced starting with the highest channel number and rotating through to the lowest channel number without regard to channel priority levels.

All channels are treated equally. Priority levels are not used in round robin mode.

12.2.6.4 DMA Transfer

12.2.6.4.1 Single request

To perform a simply transfer of ’n’ bytes of data with one activation, set the major loop to one (TCD.citer = TCD.biter = 1). The data transfer will begin after the channel service request is acknowledged and the channel is selected to execute. Once the transfer is complete, the TCD.done bit will be set and an interrupt will be generated if properly enabled.

For example, the following TCD entry is configured to transfer 16 bytes of data. The DMA is programmed for one iteration of the major loop transferring 16 bytes per iteration. The source memory has a byte wide memory port located at 0x1000. The destination memory has a word wide port located at 0x2000. The address offsets are programmed in increments to match the size of the transfer; one byte for the source and four bytes for the destination. The final source and destination addresses are adjusted to return to their beginning values.

TCD.citer = TCD.biter = 1
TCD.nbytes = 16
TCD.saddr = 0x1000
TCD.soff = 1
TCD.ssize = 0
TCD.slast = -16
TCD.daddr = 0x2000
TCD.doff = 4
TCD.dsize = 2
TCD.dlast_sga = -16
TCD.int_maj = 1
TCD.start = 1 (TCD.word7 should be written last after all other fields have been initialized)
All other TCD fields = 0

This would generate the following sequence of events:

1. IPS write to the TCD.start bit requests channel service,
2. the channel is selected by arbitration for servicing,
3. dma_engine writes: TCD.done = 0, TCD.start = 0, TCD.active = 1,
4. dma_engine reads: channel TCD data from local memory to internal register file,

5. the source to destination transfers are executed as follows:
   a) read_byte(0x1000), read_byte(0x1001), read_byte(0x1002), read_byte(0x1003),
   b) write_word(0x2000) -> *first iteration of the minor loop*
   c) read_byte(0x1004), read_byte(0x1005), read_byte(0x1006), read_byte(0x1007),
   d) write_word(0x2004) -> *second iteration of the minor loop*
   e) read_byte(0x1008), read_byte(0x1009), read_byte(0x100a), read_byte(0x100b),
   f) write_word(0x2008) -> *third iteration of the minor loop*
   g) read_byte(0x100c), read_byte(0x100d), read_byte(0x100e), read_byte(0x100f),
   h) write_word(0x200c) -> *last iteration of the minor loop -> major loop complete*
6. dma_engine writes: TCD.saddr = 0x1000, TCD.daddr = 0x2000, TCD.citer = 1 (TCD.biter),
7. dma_engine writes: TCD.active = 0, TCD.done = 1, DMAINT[n] = 1,
8. the channel retires.

The DMAgoes idle or services next channel.

12.2.6.4.2 Multiple requests

The next example is the same as previous with the exception of transferring 32 bytes via two hardware requests. The only fields that change are the major loop iteration count and the final address offsets. The DMAis programmed for two iterations of the major loop transferring 16 bytes per iteration. After the channel's hardware requests is enabled in the DMAERQ register, channel service requests are initiated by the slave device.

TCD.citer = TCD.biter = 2

TCD.slast = -32

TCD.dlast_sga = -32

This would generate the following sequence of events:

1. first hardware (ipd_req) request for channel service,
2. the channel is selected by arbitration for servicing,
3. dma_engine writes: TCD.done = 0, TCD.start = 0, TCD.active = 1,
4. dma_engine reads: channel TCD data from local memory to internal register file,
5. the source to destination transfers are executed as follows:
   a) read_byte(0x1000), read_byte(0x1001), read_byte(0x1002), read_byte(0x1003),
   b) write_word(0x2000) -> *first iteration of the minor loop*
   c) read_byte(0x1004), read_byte(0x1005), read_byte(0x1006), read_byte(0x1007),
   d) write_word(0x2004) -> *second iteration of the minor loop*
   e) read_byte(0x1008), read_byte(0x1009), read_byte(0x100a), read_byte(0x100b),
   f) write_word(0x2008) -> *third iteration of the minor loop*
   g) read_byte(0x100c), read_byte(0x100d), read_byte(0x100e), read_byte(0x100f),
   h) write_word(0x200c) -> *last iteration of the minor loop*

6. dma_engine writes: TCD.saddr = 0x1010, TCD.daddr = 0x2010, TCD.citer = 1,
7. dma_engine writes: TCD.active = 0,
8. the channel retires -> *one iteration of the major loop*

The DMA goes idle or services next channel.

9. second hardware (ipd_req) requests channel service,
10. the channel is selected by arbitration for servicing,
11. dma_engine writes: TCD.done = 0, TCD.start = 0, TCD.active = 1,
12. dma_engine reads: channel TCD data from local memory to internal register file,
13. the source to destination transfers are executed as follows:
    a) read_byte(0x1010), read_byte(0x1011), read_byte(0x1012), read_byte(0x1013),
    b) write_word(0x2010) -> *first iteration of the minor loop*
    c) read_byte(0x1014), read_byte(0x1015), read_byte(0x1016), read_byte(0x1017),
    d) write_word(0x2014) -> *second iteration of the minor loop*
    e) read_byte(0x1018), read_byte(0x1019), read_byte(0x101a), read_byte(0x101b),
    f) write_word(0x2018) -> *third iteration of the minor loop*
    g) read_byte(0x101c), read_byte(0x101d), read_byte(0x101e), read_byte(0x101f),
    h) write_word(0x201c) -> *last iteration of the minor loop -> major loop complete*
14. dma_engine writes: TCD.saddr = 0x1000, TCD.daddr = 0x2000, TCD.citer = 2 (TCD.biter),
15. dma_engine writes: TCD.active = 0, TCD.done = 1, DMAINT[n] = 1,
16. the channel retires -> *major loop complete*

The DMAgoes idle or services the next channel.

12.2.6.5 TCD Status

12.2.6.5.1 Minor loop complete

There are two methods to test for minor loop completion when using software initiated service requests. The first method is to read the TCD.citer field and test for a change. Another method may be extracted from the sequence shown below. The second method is to test the TCD.start bit AND the TCD.active bit. The minor loop complete condition is indicated by both bits reading zero after the TCD.start was written to a one. Polling the TCD.active bit may be inconclusive because the active status may be missed if the channel execution is short in duration.

The TCD status bits execute the following sequence for a software activated channel:

1. TCD.start = 1, TCD.active = 0, TCD.done = 0 (channel service request via software)
2. TCD.start = 0, TCD.active = 1, TCD.done = 0 (channel is executing)
3. TCD.start = 0, TCD.active = 0, TCD.done = 0 (channel has completed the minor loop and is idle) or
4. TCD.start = 0, TCD.active = 0, TCD.done = 1 (channel has completed the major loop and is idle)

The best method to test for minor loop completion when using hardware initiated service requests is to read the TCD.citer field and test for a change. The hardware request and ackowledge handshakes signals are not visible in the programmer's model.

The TCD status bits execute the following sequence for a hardware activated channel:

1. ipd_req asserts (channel service request via hardware)
2. TCD.start = 0, TCD.active = 1, TCD.done = 0 (channel is executing)
3. TCD.start = 0, TCD.active = 0, TCD.done = 0 (channel has completed the minor loop and is idle) or
4. TCD.start = 0, TCD.active = 0, TCD.done = 1 (channel has completed the major loop and is idle)

For both activation types, the major loop complete status is explicitly indicated via the TCD.done bit.

The TCD.start bit is cleared automatically when the channel begins execution regardless of how the channel was activated.

12.2.6.5.2 Active channel TCD reads

The DMA will read back the 'true' TCD.saddr, TCD.daddr, and TCD.nbytes values if read while a channel is executing. The 'true' values of the saddr, daddr, and nbytes are the values the dma_engine is currently using in its internal register file and not the values in the TCD local memory for that channel. The addresses (saddr and daddr) and nbytes (decrements to zero as the transfer progresses) can give an indication of the progress of the transfer. All other values are read back from the TCD local memory.

12.2.6.5.3 Preemption status

Preemption is only available when *fixed* arbitration is selected for channel arbitration modes. A preempt-able situation is one in which a preempt-enabled channel is running and a higher priority request becomes active. When the dma_engine is not operating in fixed group, fixed channel arbitration mode, the determination of the relative priority of the actively running and the outstanding requests become undefined. Channel and/or group priorities are treated as equal (or more exactly, constantly rotating) when round-robin arbitration mode is selected.

The TCD.active bit for the preempted channel remains asserted throughout the preemption. The preempted channel is temporarily suspended while the preempting channel executes one iteration of the major loop. Two TCD.active bits set at the same time in the overall TCD map indicates a higher priority channel is actively preempting a lower priority channel.

The worst case latency when switching to a preempt channel is the summation of:

- arbitration latency (2 cycles)
- bandwidth control stalls (if enabled)
- the time to execute two read/write sequences (including AHB bus holds; a system dependency driven by the slave devices or the crossbar)

12.2.6.6 Channel Linking

Channel linking (or chaining) is a mechanism where one channel sets the TCD.start bit of another channel (or itself) thus initiating a service request for that channel. This operation is automatically performed by the dma_engine at the conclusion of the major or minor loop when properly enabled.

The minor loop channel linking occurs at the completion of the minor loop (or one iteration of the major loop). The TCD.citer.e_link field are used to determine whether a minor loop link is requested. When enabled, the channel link is made after each iteration of the major loop except for the last. When the major loop is exhausted, only the major loop channel link fields are used to determine if a channel link should be made. For example, with the initial fields of:

TCD.citer.e_link = 1

TCD.citer.linkch = 0xC

TCD.citer value = 0x4

TCD.major.e_link = 1

TCD.major.linkch = 0x7

will execute as:

1. minor loop done -> set channel 12 TCD.start bit
2. minor loop done -> set channel 12 TCD.start bit
3. minor loop done -> set channel 12 TCD.start bit
4. minor loop done, major loop done -> set channel 7 TCD.start bit

When minor loop linking is enabled (TCD.citer.e_link = 1), the TCD.citer field uses a nine bit vector to form the current iteration count.

When minor loop linking is disabled (TCD.citer.e_link = 0), the TCD.citer field uses a 15 bit vector to form the current iteration count. The bits associated with the TCD.citer.linkch field are concatenated onto the citer value to increase the range of the citer.

Note: the TCD.citer.e_link bit and the TCD.biter.e_link bit must equal or a configuration error will be reported. The citer and biter vector widths must be equal in order to calculate the major loop, half-way done interrupt point.

12.2.6.7 Dynamic Programming

This section provides recommended methods to change the programming model during channel execution.

12.2.6.7.1 Dynamic priority changing

The following two options are recommended for dynamically changing ***channel*** priority levels:

1. switch to round-robin channel arbitration mode, change the channel priorities, then switch back to fixed arbitration mode,
2. disable all the channels, then change the channel priorities, then enable the appropriate channels.

12.2.6.7.2 Dynamic channel linking and dynamic scatter/gather

Dynamic channel linking and dynamic scatter/gather is the process of changing the TCD.major.e_link or TCD.e_sg bits during channel execution. These bits are read from the TCD local memory at the ***end*** of channel execution thus allowing the user to enable either feature during channel execution.

Because the user is allowed to change the configuration during execution, a coherency model is needed. Consider the scenario where the user attempts to execute a dynamic channel link by enabling the TCD.major.e_link bit at the same time the dma_engine is retiring the channel. The TCD.major.e_link would be set in the programmer's model, but it would be unclear whether the actual link was made before the channel retired.

The following coherency model is recommended when executing a dynamic channel link or dynamic scatter/gather request:

1. set the TCD.major.e_link bit,
2. read back the TCD.major.e_link bit,
3. test the TCD.major.e_link request status:
   a) if the bit is set, the dynamic link attempt was successful,
   b) if the bit is cleared, the attempted dynamic link did not succeed, the channel was already retiring.

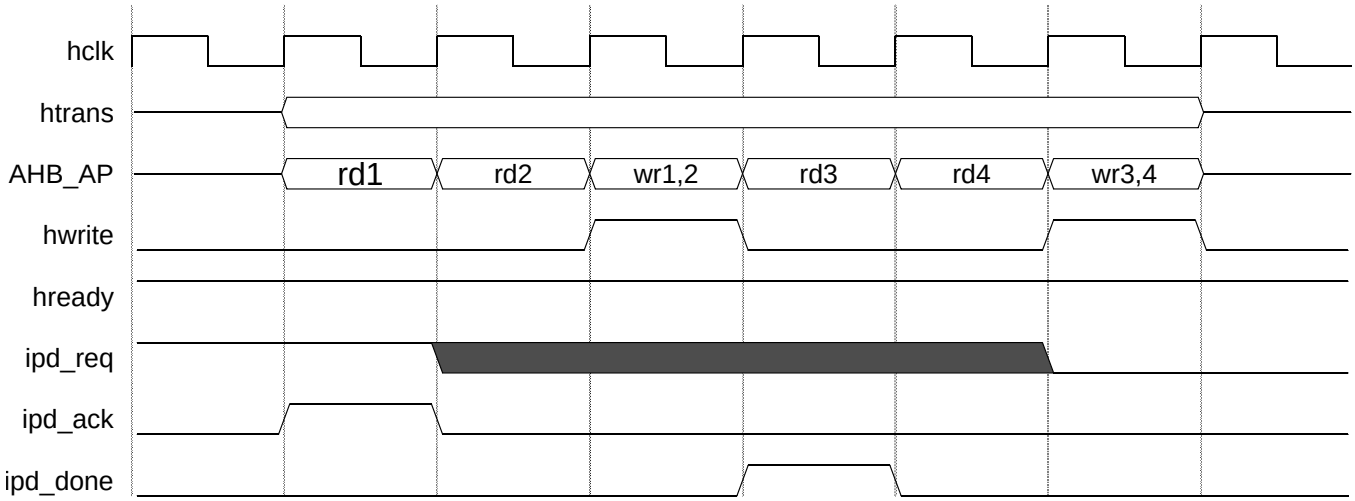
This same coherency model is true for dynamic scatter/gather operations. For both dynamic requests, the TCD local memory controller forces the TCD.major.e_link and TCD.e_sg bits to zero on any writes to a channel's TCD.word7 once that channel's TCD.done bit is set indicating the major loop is complete.

Note: The user must clear the TCD.done bit before writing the TCD.major.e_link or TCD.e_sg bits. The TCD.done bit is cleared automatically by the dma_engine once a channel begins execution.

12.2.6.8 Hardware Request Release Timing

This section provides a timing diagram for deasserting the ipd_req hardware request signal. Figure 12-25 shows an encapsulating write (i.e. 2 word reads -> 1 longword write) with grey indicating the release of the ipd_req hardware request signal.

Figure 12-25. ipd_req removal



Chapter 13
Miscellaneous Control Module (MCM)

13.1 Introduction

The Miscellaneous Control Module (MCM), formerly known as the System Control Module (SCM), provides a myriad of miscellaneous control functions for the device including program-visible information about configuration and revision levels, a reset status register, a software watchdog timer, wakeup control for exiting sleep modes, and optional features such as an address map for the device's crossbar switch, information on memory errors reported by error-correcting codes and/or generic access error information for certain processor cores.

13.1.1 Overview

The Miscellaneous Control Module is mapped into the IPS space and supports a number of miscellaneous control functions for the device.

13.1.2 Features

The MCM includes these features:

- Program-visible information on the device configuration and revision
- Reset status register (MRSR)
- Software watchdog timer (SWT) with programmable system reset or interrupt response which runs on a separate, asynchronous clock
- Wakeup control for exiting sleep modes
- Optional address map for device's crossbar switch (AXBS)
- Optional registers for capturing information on memory errors if error-correcting codes (ECC) are implemented
- Optional registers to specify the generation of single- and double-bit memory data inversions for test purposes if error-correcting codes are implemented
- Optional access address information for faulted memory accesses for certain processor core micro-architectures, e.g. ARM7TDMI-S
- Non-Maskable Interrupt with programmable polarity.

13.2 Memory Map/Register Definition

This section details the programming model for the Miscellaneous Control Module. This is a 128-byte space mapped to the region serviced by an IPS bus controller.

13.2.1 Memory Map

The Miscellaneous Control Module does not include any logic which provides access control. Rather, this function is supported using the standard access control logic provided by the IPS controller.

Table 13-1. MCM 32-bit Memory Map

MCM Offset	Register			
0x00	Processor Core Type (PCT)		Revision (REV)	
0x04	AXBS Master Configuration (AMC)		AXBS Slave Configuration (ASC)	
0x08	IPS Module Configuration (IMC)			
0x0c	Reserved			Misc Reset Status (MRSR)
0x10	Reserved			Misc Wakeup Control (MWCR)
0x14	Reserved		Misc Software Watchdog Timer Control (MSWTCR)	
0x18	Reserved			Misc SWT Service (MSWTSR)
0x1c	Reserved			Misc Interrupt (MIR)
0x20	AXBS Address Map Register (AAMR)			
0x24	Miscellaneous User-Defined Control Register (MUDCR)			
0x28	Reserved			NMI Control (NMICR)
0x28	Reserved			
0x2c	PPMRS	PPMRC	PPMRS1	PPMRC1
0x30	PPMRH[63:32]			
0x34	PPMRL[31:0]			
0x38	PPMR1H[63:32]			
0x3c	PPMR1L[31:0]			
0x40	Reserved			ECC Configuration (ECR)
0x44	Reserved			ECC Status (ESR)
0x48	Reserved		ECC Error Generation (EEGR)	
0x4c	Reserved			
0x50	Flash ECC Address (FEAR)			
0x54	Reserved		Flash ECC Master (FEMR)	Flash ECC Attributes (FEAT)
0x58	Reserved			
0x5c	Flash ECC Data (FEDR)			
0x60	RAM ECC Address (REAR)			
0x64	Reserved	RAM ECC Syndrome (RESR)	RAM ECC Master (REMR)	RAM ECC Attributes (REAT)
0x68	Reserved			
0x6c	RAM ECC Data (REDR)			
0x40 - 0x6c	Reserved			

Table 13-1. MCM 32-bit Memory Map (Continued)

0x70 - 0x7c	Reserved			
0x70	Core Fault Address (CFADR)			
0x74	Reserved	Core Fault Location 1 (CFLOC1)	Core Fault Location (CFLOC)	Core Fault Attributes (CFATR)
0x78	Reserved			
0x7c	Core Fault Data (CFDTR)			

13.2.2 Register Descriptions

Attempted accesses to reserved addresses result in an error termination, while attempted writes to read-only registers are ignored and do not terminate with an error. Unless noted otherwise, *writes to the programming model must match the size of the register*, e.g., an *n*-bit register only supports *n*-bit writes, etc. Attempted writes of a different size than the register width produce an error termination of the bus cycle and no change to the targeted register.

13.2.2.1 Processor Core Type (PCT)

The PCT is a 16-bit read-only register specifying the architecture of the processor core in the device. The state of this register is defined by a module input signal; it can only be read from the IPS programming model. Any attempted write is ignored.

See Figure 13-1 and Table 13-2 for the Processor Core Type definition.

Register address: MCM Base + 0x00

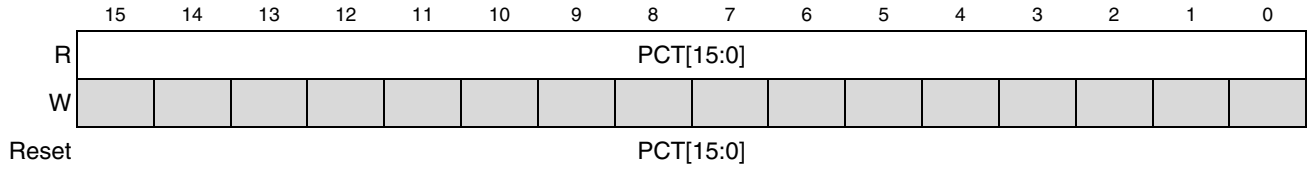


Figure 13-1. Processor Core Type (PCT) Register

Table 13-2. PCT Field Descriptions

Field	Description
15–0 PCT[15:0	Processor Core Type. The device supports ARM, ColdFire and PowerPC cores. The following values identify the specific core complexes: A700 ARM7 A900 ARM9 A110 ARM11 CF20 V2 ColdFire CF30 V3 ColdFire CF40 V4 ColdFire CF50 V5 ColdFire E650 Z650 PowerPC

13.2.2.2 Revision (REV)

The REV is a 16-bit read-only register specifying a revision number. The state of this register is defined by an input signal; it can only be read from the IPS programming model. Any attempted write is ignored.

See Figure 13-2 and Table 13-3 for the Revision definition.

Register address: MCM Base + 0x02

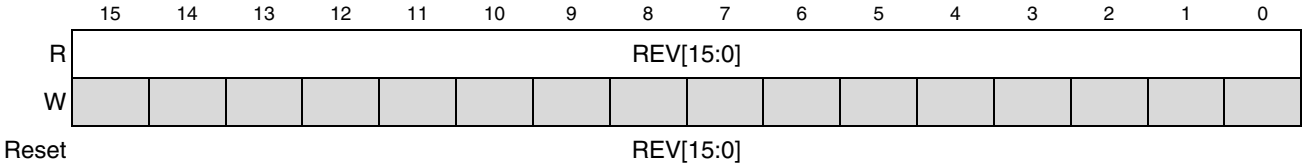


Figure 13-2. Revision (REV) Register

Table 13-3. REV Field Descriptions

Field	Description
15–0 REV[15:0]	Revision. The REV[15:0] field is specified by an input signal to define a software-visible revision number.

13.2.2.3 AXBS Master Configuration (AMC)

The AMC is a 16-bit read-only register identifying the presence/absence of bus master connections to the device's AMBA-AHB Crossbar Switch (AXBS). The state of this register is defined by a module input signal; it can only be read from the IPS programming model. Any attempted write is ignored.

See Figure 13-3 and Table 13-4 for the AXBS Master Configuration definition.

Register address: MCM Base + 0x04

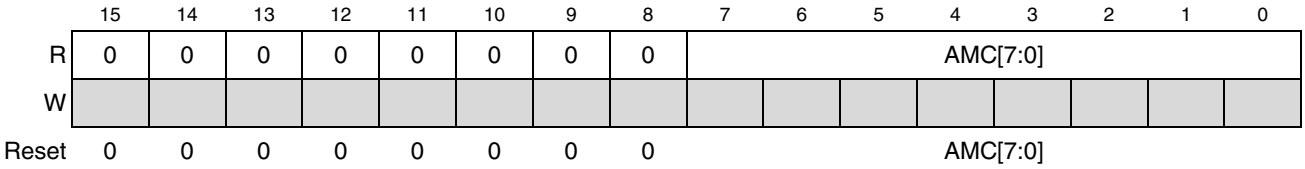


Figure 13-3. AXBS Master Configuration (AMC) Register

Table 13-4. AMC Field Descriptions

Field	Description
15–8	Reserved, should be cleared.
7–0 AMC[7:0]	AXBS Master Configuration. AMC*n* = 0 if a bus master connection to AXBS input port "n" is absent AMC*n* = 1 if a bus master connection to AXBS input port "n" is present

13.2.2.4 AXBS Slave Configuration (ASC)

The ASC is a 16-bit read-only register identifying the presence/absence of bus slave connections to the device's AMBA-AHB Crossbar Switch (AXBS), plus a 1-bit flag defining the datapath width (DP64). The state of this register is defined by a module input signal; it can only be read from the IPS programming model. Any attempted write is ignored.

See Figure 13-4 and Table 13-5 for the AXBS Slave Configuration definition.

Register address: MCM Base + 0x06

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	DP64	0	0	0	0	0	0	0	ASC[7:0]							
W																
Reset	DP64	0	0	0	0	0	0	0	ASC[7:0]							

Figure 13-4. AXBS Slave Configuration (ASC) Register

Table 13-5. ASC Field Descriptions

Field	Description
15 DP64	64-bit Datapath. 0 The datapath width is 32 bits 1 The datapath width is 64 bits
14–8	Reserved, should be cleared.
7–0 ASC[7:0]	AXBS Slave Configuration. ASC*n* = 0 if a bus slave connection to AXBS output port "n" is absent ASC*n* = 1 if a bus slave connection to AXBS output port "n" is present

13.2.2.5 IPS Module Configuration (IMC)

The IMC is a 32-bit read-only register identifying the presence/absence of the 32 low-order IPS peripheral modules connected to the primary IPS bus controller. The state of this register is defined by a module input signal; it can only be read from the IPS programming model. Any attempted write is ignored.

See Figure 13-5 and Table 13-6 for the IPS Module Configuration definition.

Register address: MCM Base + 0x08

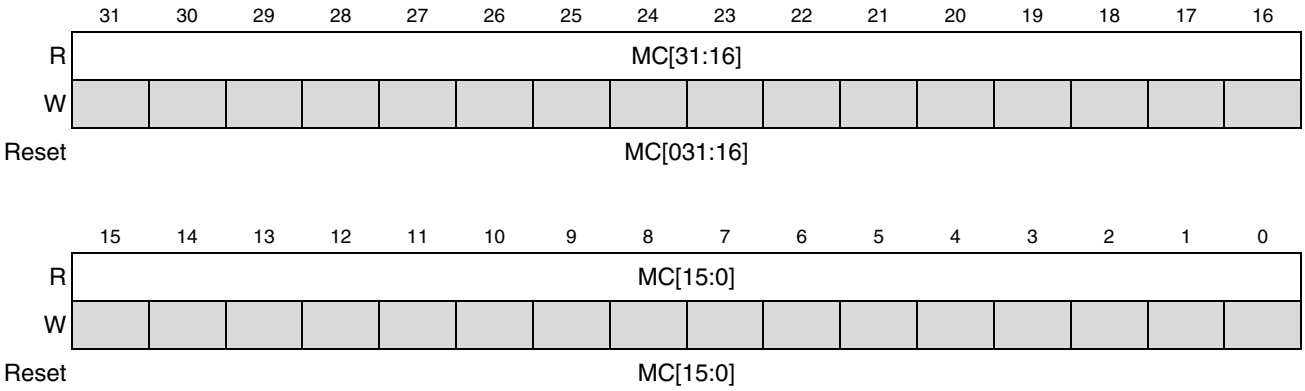


Figure 13-5. IPS Module Configuration (IMC) Register

Table 13-6. IPS IMC Field Descriptions

Field	Description
31–0 MC[31:0]	IPS Module Configuration, MC*n* = 0 if an IPS module connection to decoded slot “n” is absent MC*n* = 1 if an IPS module connection to decoded slot “n” is present

13.2.2.6 Miscellaneous Reset Status Register (MRSR)

The MRSR contains a bit for each of the reset sources to the device. An asserted bit indicates the last type of reset that occurred. Only one bit is set at any time in the MRSR, reflecting the cause of the most recent reset as signalled by device reset input signals. The MRSR can only be read from the IPS programming model. Any attempted write is ignored. See Figure 13-6 and Table 13-7 for the Miscellaneous Reset Status Register definition.

Register address: MCM Base + 0x0f

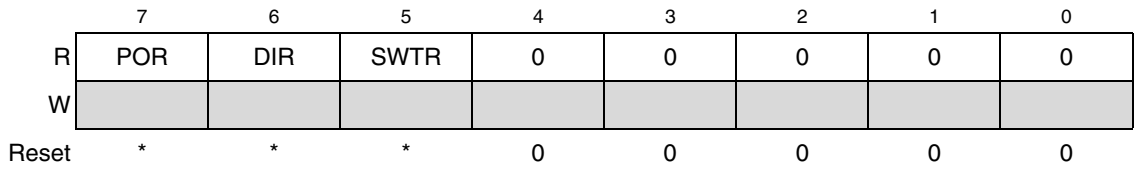


Figure 13-6. Miscellaneous Reset Status (MRSR) Register

Table 13-7. MRSR Field Descriptions

Field	Description
7 POR	Power-On Reset. 1 Last recorded event was caused by a power-on reset (based on a device input signal)
6 DIR	Device Input Reset. 1 Last recorded event was a reset caused by a device input reset.

Table 13-7. MRSR Field Descriptions (Continued)

Field	Description
5 SWTR	Software Watchdog Timer Reset. 1 Last recorded event was a reset caused by the MCM's software watchdog timer.
4–0	Reserved, should be cleared.

13.2.2.7 Miscellaneous Wakeup Control Register (MWCR)

Implementation of low-power sleep modes and exit from these modes via an interrupt require communication between the MCM, the interrupt controller and external logic typically associated with phase-locked loop clock generation circuitry. The Miscellaneous Wakeup Control Register (MWCR) provides an 8-bit register controlling entry into these types of low-power modes as well as definition of the interrupt level needed to exit the mode.

The following sequence of operations is generally needed to enable this functionality. Note that the exact details are likely to be system-specific.

1. The processor core loads the appropriate data value into the MWCR, setting the ENBWCR bit and the desired interrupt priority level.
2. At the appropriate time, the processor ceases execution. The exact mechanism varies by processor core. In some cases, a *processor-is-stopped* status is signaled to the MCM and external logic. This assertion, if properly enabled by MWCR[ENBWCR], causes the selected external, low-power mode, as specified by MWCR[LPMD], to be entered, and the appropriate clock signals disabled. In most implementations, there are multiple low-power modes, where the exact clocks to be disabled vary across the different modes.
3. After entering the low-power mode, the interrupt controller enables a special combinational logic path which evaluates all unmasked interrupt requests. The device remains in this mode until an event which generates an unmasked interrupt request with a priority level *greater than* the value programmed in the MWCR[PRILVL] occurs.
4. Once the appropriately-high interrupt request level arrives, the interrupt controller signals its presence, and the MCM responds by asserting an "exit_low_power_mode" signal.
5. The external logic senses the assertion of the "exit" signal, and re-enables the appropriate clock signals.
6. With the processor core clocks enabled, the core handles the pending interrupt request.

See Figure 13-7 and Table 13-8 for the Miscellaneous Wakeup Control Register definition.

Register address: MCM Base + 0x13

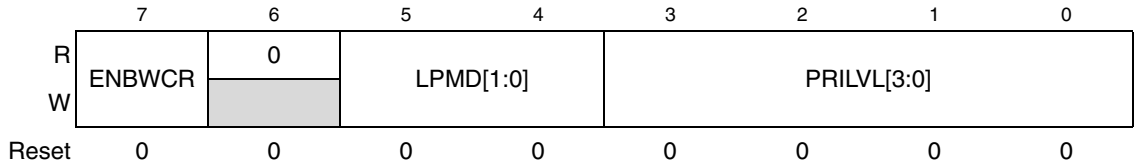


Figure 13-7. Miscellaneous Wakeup Control (MWCR) Register

Table 13-8. MWCR Field Descriptions

Field	Description
7 ENBWCR	Enable WCR. 0 MWCR is disabled. 1 MWCR is enabled.
6	Reserved, should be cleared.
5–4 LPMD[1:0]	Low Power Mode. Used to select the low-power mode the chip enters once the ColdFire core executes the STOP instruction. These bits must be written prior to instruction execution for them to take effect. The LPMD bits are readable and writable in all modes. 0b00 Run 0b01 Doze 0b10 Wait 0b11 Stop **Note:** If MWCR[LPMD] is cleared, then the device will stop executing code upon issue of a STOP instruction. However, no clocks will be disabled.
3–0 PRILVL[3:0]	Interrupt Priority Level. The interrupt priority level is a core-specific definition. It specifies the interrupt priority level needed to exit the low-power mode. Specifically, an unmasked interrupt request of a priority level *greater than* the PRILVL value is required to exit the mode. Certain interrupt controller implementations include logic associated with this priority level that restricts the data value contained in this field to a [0, maximum - 1] range. See the specific interrupt controller module for details.

13.2.2.8 Miscellaneous Software Watchdog Timer Control Register (MSWTCR)

The software watchdog timer (SWT) prevents system lockup if the software becomes trapped in a loop with no controlled exit or if a bus transaction becomes “hung.” The software watchdog timer can be enabled or disabled through the MSWTCR[SWE]. By default, it is disabled. The SWT operates in a separate, asynchronous clock domain and contains clock domain synchronizers as the communication mechanism between the system clock domain and the software watchdog timer domain. If enabled, the watchdog timer requires the periodic execution of a software watchdog servicing sequence. If this periodic servicing action does *not* occur, the timer expires, resulting in a watchdog timer interrupt or a hardware reset, as programmed in the MSWTCR[SWRI].

There are three user-defined responses to a time-out:

1. The MSWTCR[SWRI] can specify the assertion of a watchdog timer interrupt.
2. The MSWTCR[SWRI] can specify the immediate assertion of a system reset.
3. The MSWTCR[SWRI] can specify a sequence of responses. Upon the first time-out, a watchdog timer interrupt is asserted. If this time-out condition is *not serviced* before a second time-out occurs, the watchdog timer asserts the system reset signal in response to the second time-out. This configuration supports a more graceful response to watchdog time-outs: first attempting an interrupt to notify the system, and if that fails, generating a system reset.

In addition to these three basic modes of operation, the watchdog timer also supports a “windowed” mode of operation. In this mode, the time-out period is divided into 4 equal segments and the actual servicing of the watchdog timer must occur during the last segment, i.e., during the last 25% of the time-out period. If

the watchdog timer is serviced anytime in the first 75% of the time-out period, an immediate system reset occurs.

Throughout Section 13.2.2.8, “Miscellaneous Software Watchdog Timer Control Register (MSWTCR), there are numerous references to the generation of a system reset. MCM’s behavior during this process is detailed below. When the watchdog timer expires and MSWTCR[SWRI] is programmed for a reset (either as the initial or secondary response), the MCM generates a watchdog timer reset output signal which is driven off the device where it is typically combined with other reset signals and driven throughout the system. The combined reset input signal is driven back to the device and MCM, where MRSR[SWTR] can be set and the appropriate action taken by the core and device logic.

The watchdog timer logic also sends an interrupt request to the device’s interrupt controller.

To prevent the watchdog timer from interrupting or resetting, the MSWTSR must be serviced by performing the following sequence:

- Write 0x55 to the MSWTSR.
- Write 0xAA to the MSWTSR.

Both writes must occur in this order before the time-out, but any number of instructions can be executed between the two writes. This definition allows interrupts and exceptions to occur, if necessary, between the two writes. The timer value is constantly compared with the time-out period specified by MSWTCR[SWT]. *Any write to the MSWTCR resets the watchdog timer.* In addition, there is a read-only control flag included in the MSWTCR to prevent accidental updates to this control register from changing the desired system configuration.

If the second write occurs at the exact same cycle as the time-out condition is reached, the clear takes precedence and the time-out response suppressed.

The MSWTCR controls the software watchdog timer, time-out periods, and time-out response. The register can be read or written at any time. At system reset, the software watchdog timer is disabled. See Figure 13-8 and Table 13-9 for the Miscellaneous Software Watchdog Timer Control Register definition.

Register address: MCM Base + 0x16

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	RO	0	0	0	0	0	SWCIN16	SWRWH	SWE	SWRI[1:0]		SWT[4:0]				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 13-8. Miscellaneous Software Watchdog Timer Control (MSWTCR) Register

Table 13-9. MSWTCR Field Definitions

Field	Description
15 RO	Read-Only. 0 MSWTCR can be read or written. 1 MSWTCR can only be read. A system reset is required before this register can again be written. The setting of this bit is intended to prevent accidental writes of the MSWTCR from changing the defined system watchdog configuration.
14–10	Reserved, should be cleared.
9 SWCIN16	Force SWT CarryIn16. This control bit is intended for use only when testing the operation of the SWT. When asserted, it forces the actual timer to increment by 65537 (2^{16} + 1) each cycle rather than simply by 1. It allows testing of the large SWT time-out values without actually incrementing through the entire dynamic range.
8 SWRWH	Software Watchdog Run While Halted. 0 SWT stops counting if the processor core is halted. 1 SWT continues to count even while the processor core is halted.
7 SWE	Software Watchdog Enable 0 SWT is disabled. 1 SWT is enabled.
6–5 SWRI[1:0]	Software Watchdog Reset/Interrupt. 00 If a time-out occurs, the SWT generates an interrupt to the processor core. The programming of the interrupt level for the SWT is system-specific. Typically, the highest priority interrupt level is used to signal the SWT. 01 The first time-out generates an interrupt to the processor, and if not serviced, then a second time-out generates a system reset and sets the MRSR[SWTR] flag. 10 If a time-out occurs, the SWT generates a system reset. MRSR[SWTR] is set. 11 The SWT functions in a “window” mode of operation. For this mode, the servicing of the MSWSR must occur during the last 25% of the time-out period. Any writes to the MSWSR during the first 75% of the time-out period generate an immediate system reset. Likewise, if the MSWSR is not serviced during the last 25% of the time-out period, then a system reset is generated. For any type of reset response, the MRSR[SWTR] flag is set. As noted previously, the generation of a system reset actually causes the MCM module to assert an output signal which is driven out of the device where it is combined with other reset signals and then driven throughout the system.
4–0 SWT[4:0]	Software Watchdog Time-Out Period. This field selects the time-out period for the SWT. At reset, this field is cleared selecting the minimum time-out period, but the SWT is disabled since MSWTCR[SWE] = 0. In general, the value in this field defines the bit position within the 32-bit counter that specifies the time-out period. Thus, if SWT[4:0] = n, then the time-out period is 2^n system clock cycles. Since it is not practical to operate the software watchdog timer with very short time-out periods, data values of [0-7] are forced to a value of 8, defining a minimum time-out period of 256 system clock cycles. The logic which forces the minimum value to 8 does **not** affect the contents of this field in the register. For SWT[4:0] = n, then time-out period = 2^n system clock cycles, n = 8,9,..., 31.

13.2.2.9 Miscellaneous Software Watchdog Timer Service Register (MSWTSR)

The software service sequence must be performed using the MSWTSR as a data register to prevent a SWT time-out. The service sequence requires two writes to this data register: first a write of 0x55 followed by a write of 0xAA. Both writes must be performed in this order prior to the SWT time-out, but any number of instructions or accesses to the SWSR may occur between the two writes. If the SWT has already timed

out, writing to this register has no effect in negating the SWT interrupt or reset. Figure 13-9 illustrates the MSWTSR.

Register address: MCM Base + 0x1b

	7	6	5	4	3	2	1	0
R	SWSR[7:0]							
W								
Reset	—	—	—	—	—	—	—	—

Figure 13-9. Miscellaneous Software Watchdog Timer Service (MSWTSR) Register

If the software watchdog timer is enabled (MSWTCR[SWE] = 1), then any write of a data value **other than** 0x55 or 0xAA generates an immediate system reset, *regardless of the value in the MSWTCR[SWRI] field.*

13.2.2.10 Miscellaneous Interrupt Register (MIR)

All interrupt requests associated with MCM are collected in the MIR register. This includes the software watchdog timer interrupt and the processor core system bus fault interrupt.

During the appropriate interrupt service routine handling these requests, the interrupt source contained in the MCMIR must be explicitly cleared. See Figure 13-10 and Table 13-10.

Register address: MCM Base + 0x1f

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	CFEI	SWTIC
W								
Reset	0	0	0	0	0	0	0	0

Figure 13-10. Miscellaneous Interrupt (MIR) Register

Table 13-10. MIR Field Descriptions

Field	Description
7–2	Reserved, should be cleeared.
1 CFEI	Code Fault Error Interrupt Flag. 0 An enabled processor core system bus fault has not occurred.. 1 An enabled processor core system bus fault has occurred. The faulting address, attributes (and possibly write data) are captured in the CFADR, CFATR, and CFDTR registers. The error interrupt is only enabled if CFLOC[ECFEI] is set. The interrupt request is negated by writing a 1 to this bit. Writing a 0 has no effect..
0 SWTIC	Software Watchdog Interrupt Flag. 0 An SWT interrupt has not occurred. 1 An SWT interrupt has occurred. The interrupt request is negated by *writing a 1 to this bit*. Writing a 0 has no effect.

13.2.2.11 AXBS Address Map Register (AAMR)

For certain designs, the static decoding of the upper bits of the AMBA-AHB address bus to steer requests to the appropriate crossbar switch slave may not be sufficient. For those designs, the AAMR provides an optional register which implements additional capability for AXBS request steering. This register provides a program-visible address steering mechanism by supporting the mapping of any 512 MByte address space to any of the AXBS slave connections.

The AAMR is divided into eight 4-bit fields providing an enable bit and a 3-bit slave number. Thus, the upper 3 bits of the AMBA-AHB address (HADDR[31:29]) are used to index into the AAMR to select a corresponding 3-bit vector which defines the targeted AXBS slave.

The AAMR is initially loaded at reset with a 32-bit value provided as an input. For subsequent writes to the AAMR, the hardware invokes the following synchronization mechanism:

1. The 32-bit IPS write to the AAMR loads the pending address map operand into a temporary register.
2. The pending AAMR write causes the MCM to assert a control signal which forces the AXBS into a “halted” state.
3. The IPS write cycle is terminated normally.
4. Once the AXBS module indicates it has stalled all bus master transactions and entered the halted state, then MCM loads the pending address map operand into the AAMR, effectively enabling the new mapping.
5. MCM then releases the halt request signal to the AXBS, allowing it to resume operation with the just-loaded address map.

The device is guaranteed to function properly after changing the value of the AAMR if the write to the register is followed by a sufficient number of NOP instructions necessary to flush out the depth of the processor pipeline.

The AAMR can only be written in its entirety, i.e., only 32-bit writes are supported. Any attempted write of a smaller data size (e.g., 8- or 16-bit value) generates an error response. See Figure 13-11 and Table 13-11 for the AXBS Address Map Register definition.

Register address: MCM Offset + 0x20

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	EA7	A7Slave			EA6	A6Slave			EA5	A5Slave			EA4	A4Slave		
Reset	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	EA3	A3Slave			EA2	A2Slave			EA1	A1Slave			EA0	A0Slave		
Reset	0	0	0	0	1	0	1	1	aamr_init[7:0]							

Figure 13-11. AXBS Address Map (AAMR) Register

where *aamr_init[7:0]* is the value at reset controlled by the reset mode. The MCM only allows the following values to be written to the AAMR[31:0] 0xF000_00{00, 08, 09, 0B, 80, 90, B0, 89, 8B, 98, 9B, B8, B9, 99, and BB}. If the flash is not valid, 0x8 cannot be written to either of the two least-significant nibbles of the AAMR. Likewise, if the EIM is not valid, 0x9 cannot be written to AAMR[7:4] or AAMR[3:0]. The register will not update if an attempt is made to write an illegal value to the AAMR register.

Table 13-11. AAMR Field Descriptions

Field	Description
EA*n*	Enable Address Region *n.* 0 The 512 MByte address region associated with HADDR[31:29] = n is disabled, and not mapped to an AXBS slave. Accesses to this memory region terminate with an error response. 1 The 512 MByte address region associated with HADDR[31:29] = n is enabled, and mapped to the AXBS slave defined by AnSlave.
A*n*Slave	Address *n* Slave Number. 1 A*n*Slave defines the AXBS slave mapped to the memory region defined by HADDR[31:29] = n

13.2.2.12 Miscellaneous User-Defined Control Register (MUDCR)

The MUDCR provides a program-visible register for user-defined control functions. It typically is used as configuration control for miscellaneous SoC-level modules. The contents of this register is simply output from MCM to other modules where the user-defined control functions are implemented. See Figure 13-12 and Table 13-12 for the Miscellaneous User-Defined Control Register definition.

Register address: MCM Base + 0x24

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	MUDCR[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	MUDCR[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 13-12. Miscellaneous User-Defined Control (MUDCR) Register

Table 13-12. MUDCR Field Descriptions

Field	Description
31–0 MUDCR	User-Defined control Register. 0 The control associated with this MUDCR bit is disabled. 1 The control associated with this MUDCR bit is enabled.

13.2.2.13 NMI Control Register (NMICR)

The NMICR is an 8-bit register controlling the operation of a non-maskable interrupt. This functionality is specifically provided to compensate for processor cores which do not directly support this type of interrupt request. This logic operates by forcing error terminations (or aborts) on instruction fetches once a properly-enabled NMI request is asserted. The resulting exception processing of the prefetch abort then provides the required NMI service routine. The NMI input signal is treated as an edge-sensitive event, and the request must be negated before a subsequent NMI is recognized. The required sequence of operations is detailed below:

1. The NMICR is written to enable the non-maskable request and specify the polarity of the input signal. The NMICR is a write-once register and any subsequent writes to this register are ignored.
2. Once an enabled NMI request is detected, the MCM logic begins forcing error terminations on all instruction prefetches from the processor. At the same time, both FIQ and IRQ interrupt requests are disabled by this logic.
3. The error tagging of instruction prefetches continues until the processor recognizes the prefetch abort exception and accesses the exception vector at address 0x0000_000c.
4. Once the prefetch abort exception handler begins execution, the state of the NMICR[NMI_PEND] flag is checked by the handler. If NMI_PEND = 1, then the prefetch abort was generated by a pending NMI request, and execution continues in the appropriate NMI service routine.
5. Eventually, the source of the NMI request is negated. The service routine must properly adjust the value of the CPSR to enable/disable FIQ and IRQ interrupts as required. The MCM logic disables both the FIQ and IRQ inputs to the processor core while a properly-enabled NMI request is asserted.

See Table 13-13 and Table 13-13 for the NMI Control Register definition.

Register address: MCM Base + 0x2b

	7	6	5	4	3	2	1	0
R	NMI_PEND	0	0	0	0	0	NMI_POL	NMI_ENB
W								
Reset	—	—	—	—	—	0	0	0

Figure 13-13. NMI Control (NMICR) Register

Table 13-13. NMICR Field Descriptions

Field	Description
7 NMI_PEND	Non-Maskable Interrupt Pending. 0 Non-Maskable Interrupt is not pending 1 Non-Maskable Interrupt is pendin
6–2	Reserved, should be cleared.

Table 13-13. NMICR Field Descriptions (Continued)

Field	Description
1 NMI_POL	Non-Maskable Interrupt Polarity. 0 Non-Maskable Interrupt is active low 1 Non-Maskable Interrupt is active high
0 NMI_ENB	Non-Maskable Interrupt Enable. 0 Non-Maskable Interrupt is disabled 1 Non-Maskable Interrupt is enabled

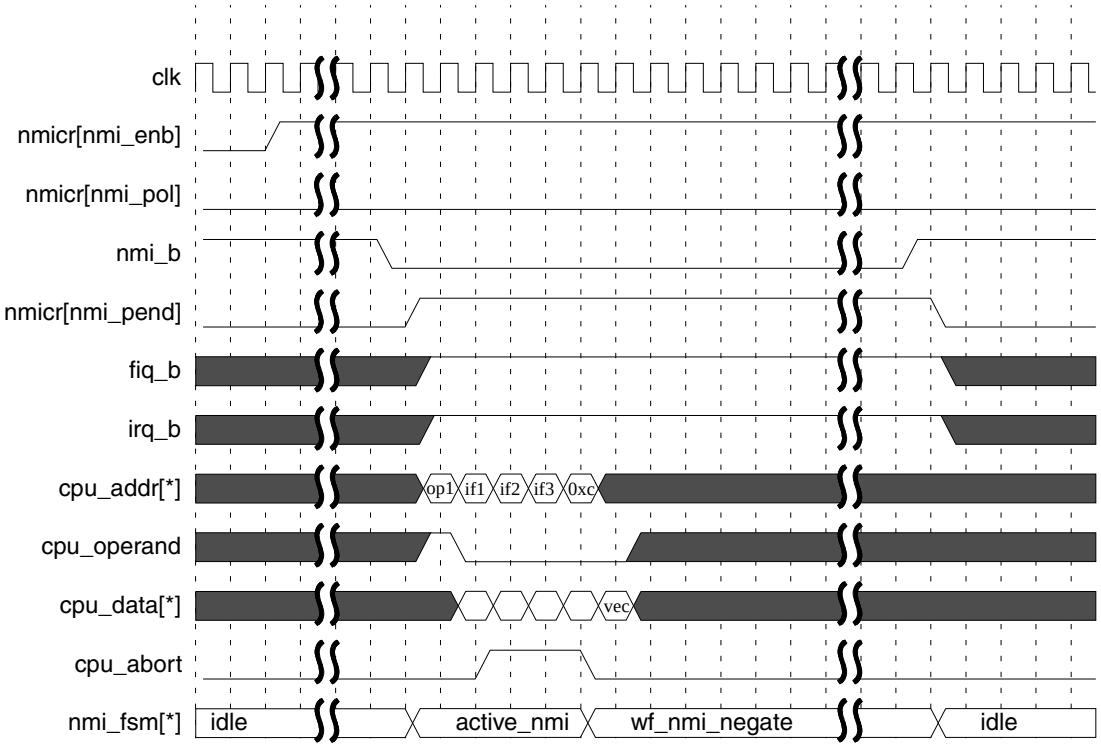


Figure 13-14. Non-Maskable Interrupt Operation Timing

13.2.2.14 Peripheral Power Management Registers (PPMR)

The Miscellaneous Control Module implements two 64-bit IPS peripheral power management control registers along with 4 additional registers for easy manipulation of these configuration registers.

13.2.2.14.1 Peripheral Power Management Set Register (PPMRS)

The PPMRS register provides a simple memory-mapped mechanism to set a given bit in the PPMR{H,L} registers to *disable the clock* for a given IPS module without the need to perform a read-modify-write on the PPMR. The data value on a register write causes the corresponding bit in the PPMR{H,L} register to be set. A data value of 64 to 127 provides a global set function, forcing the entire contents of the PPMR{H,L} to be set, disabling all IPS module clocks. Reads of this register return all zeroes. See Figure 13-15 and Figure 13-14 for the PPMRS definition.

Register address: MCM Base + 0x2c

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W		PPMRS[6:0]						
Reset	—	0	0	0	0	0	0	0

Figure 13-15. Peripheral Power Management Set (PPMRS) Register

Table 13-14. Peripheral Power Management Set (PPMRS) Field Descriptions

Field	Description
7	Reserved, should be cleared.
6–0 PPMRS[6:0]	Set Module Clock Disable. 0-63 Set corresponding bit in PPMR{H,L}, disabling the module clock. 64-127 Set all bits in PPMR{H,L}, disabling all the module clocks.

13.2.2.14.2 Peripheral Power Management Clear Register (PPMRC)

The PPMRC register provides a simple memory-mapped mechanism to clear a given bit in the PPMR{H,L} registers to *enable the clock* for a given IPS module without the need to perform a read-modify-write on the PPMR. The data value on a register write causes the corresponding bit in the PPMR{H,L} register to be cleared. A data value of 64 to 127 provides a global clear function, forcing the entire contents of the PPMR{H,L} to be zeroed, enabling all IPS module clocks. In the event on simultaneous writes of the PPMRS and PPMRC, the write to the PPMRC takes priority. Reads of this register return all zeroes. See Figure 13-16 and Figure 13-15 for the PPMRC definition.

Register address: MCM Base + 0x2d

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W		PPMRC[6:0]						
Reset	—	0	0	0	0	0	0	0

Figure 13-16. Peripheral Power Management Clear (PPMRC) Register

Table 13-15. Peripheral Power Management Clear (PPMRC) Field Descriptions

Field	Description
7	Reserved, should be cleared.
6–0 PPMRC[6:0]	Clear Module Clock Disable. 0-63 Clear corresponding bit in PPMR{H,L}, enabling the module clock. 64-127 Clear all bits in PPMR{H,L}, enabling all the module clocks.

13.2.2.14.3 Peripheral Power Management Set Register 1 (PPMRS1)

The PPMRS1 register provides a simple memory-mapped mechanism to set a given bit in the PPMR1{H,L} registers to *disable the clock* for a given IPS module without the need to perform a read-modify-write on the PPMR1. The data value on a register write causes the corresponding bit in the PPMR1{H,L} register to be set. A data value of 64 to 127 provides a global set function, forcing the entire contents of the PPMR1{H,L} to be set, disabling all IPS module clocks. Reads of this register return all zeroes. See Figure 13-17 and Figure 13-16 for the PPMRS1 definition.

Register address: MCM Base + 0x2e

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W		PPMRS1[6:0]						
Reset	—	0	0	0	0	0	0	0

Figure 13-17. Peripheral Power Management Set 1 (PPMRS1) Register

Table 13-16. Peripheral Power Management Set 1 (PPMRS1) Field Descriptions

Field	Description
7	Reserved, should be cleared.
6–0 PPMRS1[6:0]	Set Module Clock Disable. 0-63 Set corresponding bit in PPMR1{H,L}, disabling the module clock. 64-127 Set all bits in PPMR1{H,L}, disabling all the module clocks.

13.2.2.14.4 Peripheral Power Management Clear Register 1 (PPMRC1)

The PPMRC1 register provides a simple memory-mapped mechanism to clear a given bit in the PPMR1{H,L} registers to *enable the clock* for a given IPS module without the need to perform a read-modify-write on the PPMR1. The data value on a register write causes the corresponding bit in the PPMR1{H,L} register to be cleared. A data value of 64 to 127 provides a global clear function, forcing the entire contents of the PPMR1{H,L} to be zeroed, enabling all IPS module clocks. In the event on simultaneous writes to the PPMRS1 and PPMRC1, the write to the PPMRC1 takes priority. Reads of this register return all zeroes. See Figure 13-18 and Figure 13-17 for the PPMRC1 definition.

Register address: MCM Base + 0x2f

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W		PPMRC1[6:0]						
Reset	—	0	0	0	0	0	0	0

Figure 13-18. Peripheral Power Management Clear 1 (PPMRC1) Register

Table 13-17. Peripheral Power Management Clear 1 (PPMRC1) Field Descriptions

Field	Description
7	Reserved, should be cleared.
6–0 PPMRC1[6:0]	Clear Module Clock Disable. 0-63 Clear corresponding bit in PPMR1{H,L}, enabling the module clock. 64-127 Clear all bits in PPMR1{H,L}, enabling all the module clocks.

13.2.2.14.5 Peripheral Power Management Register (PPMR{H,L})

The PPMR{H,L} registers provide a bit map for controlling the generation of the module clocks for each decoded address space associated with the AIPS controller. Each AIPS controller decodes the mapped address space into sixty-four 16 KByte “slots” (32 on-platform and 32 off-platform) plus a 63 MByte off-platform global region. The PPMR provides a unique control bit for each of these address spaces that defines whether the module clock for the given space is enabled or disabled. *It is software’s responsibility to appropriately disable module clocks using the PPMR only when a module is completely unused or quiescent.*

Since the operation of the crossbar switch, the AIPS controller and the Miscellaneous Control Module (MCM) are fundamental to the operation of the core platform, the clocks for these three modules cannot be disabled.

The individual bits of the PPMR can be modified using a read-modify-write to this register directly or indirectly through writes to the PPMRS and PPMRC registers to set/clear individual bits. If the static module enable is negated, the appropriate PPMR bit is automatically set by MCM after the negation of the platform reset and cannot be altered by any subsequent IPS writes.

See Figure 13-19, Figure 13-20, and Figure 13-18 for the PPMR definition.

Register address: MCM Base + 0x0030 (PPMRH)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	CD63	CD62	CD61	CD60	CD59	CD58	CD57	CD56	CD55	CD54	CD53	CD52	CD51	CD50	CD49	CD48
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	CD47	CD46	CD45	CD44	CD43	CD42	CD41	CD40	CD39	CD38	CD37	CD36	CD35	CD34	CD33	CD32
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 13-19. Peripheral Power Management Register High (PPMRH)

Register address: MCM Base + 0x0034 (PPMRL)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	CD31	CD30	CD29	CD28	CD27	CD26	CD25	CD24	CD23	CD22	0	CD20	CD19	CD18	CD17	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	CD15	CD14	CD13	CD12	CD11	CD10	CD09	CD08	CD07	CD06	CD05	CD04	CD03	CD02	CDG	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 13-20. Peripheral Power Management Register Low (PPMRL)

Table 13-18. Peripheral Power Management (PPMRH, PPMRL) Field Description

Field	Description
63–2 CD[63:2]	Clock Disable, Module Slot n. 0 The clock for this IPS module is enabled. 1 The clock for this IPS module is disabled.
1 CDG	0 The clock for the global IPS space is enabled. 1 The clock for the global IPS space is disabled.
0	Reserved, should be cleared.

13.2.2.14.6 Peripheral Power Management Register 1 (PPMR1{H,L})

The PPMR1{H,L} registers provide a bit map for controlling the generation of the module clocks for each decoded address space associated with the *optional AIPS1 controller*. Recall each AIPS controller decodes the mapped address space into sixty-four 16 KByte “slots” (32 on-platform and 32 off-platform) and a 63 MByte off-platform global region. The PPMR1 provides a unique control bit for each of these address spaces that defines whether the module clock for the given space is enabled or disabled. *It is software’s responsibility to appropriately disable module clocks using the PPMR1 only when a module is completely unused or quiescent.*

The individual bits of the PPMR1 can be modified using a read-modify-write to this register directly or indirectly through writes to the PPMRS1 and PPMRC1 registers to set/clear individual bits. If the static module enable is negated, the appropriate PPMR1 bit is automatically set by MCM after the negation of the platform reset and cannot be altered by any subsequent IPS writes.

See Figure 13-21, Figure 13-22, and Figure 13-19 for the PPMR1 definition.

Register address: MCM Base + 0x0038 (PPMR1H)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	CD63	CD62	CD61	CD60	CD59	CD58	CD57	CD56	CD55	CD54	CD53	CD52	CD51	CD50	CD49	CD48
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	CD47	CD46	CD45	CD44	CD43	CD42	CD41	CD40	CD39	CD38	CD37	CD36	CD35	CD34	CD33	CD32
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 13-21. Peripheral Power Management Register 1 High (PPMR1H)

Register address: MCM Base + 0x003c (PPMR1L)

	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																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	CDG	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 13-22. Peripheral Power Management Register 1 Low (PPMR1L)

Table 13-19. Peripheral Power Management (PPMR1H, PPMR1L) Field Description

Field	Description
63–32 CD[63:32]	Clock Disable, Module Slot n. 0 The clock for this IPS module is enabled. 1 The clock for this IPS module is disabled.
31–2	Reserved, should be cleared.
1 CDG	0 The clock for the global IPS space is enabled. 1 The clock for the global IPS space is disabled.
0	Reserved, should be cleared.S

13.2.2.15 ECC Registers

For designs including error-correcting code (ECC) implementations to improve the quality and reliability of memories, there are a number of program-visible registers for the sole purpose of reporting and logging of memory failures. These optional registers include:

- ECC Configuration Register (ECR)
- ECC Status Register (ESR)
- ECC Error Generation Register (EEGR)
- Flash ECC Address Register (FEAR)
- Flash ECC Master Number Register (FEMR)
- Flash ECC Attributes Register (FEAT)
- Flash ECC Data Register (FEDR)
- RAM ECC Address Register (REAR)
- RAM ECC Syndrome Register (RESR)
- RAM ECC Master Number Register (REMR)
- RAM ECC Attributes Register (REAT)
- RAM ECC Data Register (REDR)

The details on the ECC registers are provided in the subsequent sections. If the design does not include ECC on the memories, these addresses are reserved locations within the MCM's programming model.

13.2.2.15.1 ECC Configuration Register (ECR)

The ECC Configuration Register is an 8-bit control register for specifying which types of memory errors are reported. In all systems with ECC, the occurrence of a non-correctable error causes the current access to be terminated with an error condition. In many cases, this error termination is reported directly by the initiating bus master. However, there are certain situations where the occurrence of this type of non-correctable error is *not* reported by the master. Examples include speculative instruction fetches which are discarded due to a change-of-flow operation, and buffered operand writes. The ECC reporting logic in the MCM provides an optional error interrupt mechanism to signal all non-correctable memory errors. In addition to the interrupt generation, the MCM captures specific information (memory address, attributes and data, bus master number, etc.) which may be useful for subsequent failure analysis.

The reporting of single-bit memory corrections can only be enabled via a an SoC-configurable module input signal. While not directly accessible to a user, this capability is viewed as important for error logging and failure analysis.

See Figure 13-23 and Table 13-20 for the ECC Configuration Register definition.

Register address: MCM Base + 0x43

	7	6	5	4	3	2	1	0
R	0	0	ER1BR	EF1BR	0	0	ERNCR	EFNCR
W								
Reset	0	0	0	0	0	0	0	0

Figure 13-23. ECC Configuration (ECR) Register

Table 13-20. ECR Field Descriptions

Field	Description
7–6	Reserved, should be cleared.
5 ER1BR	Enable RAM 1-bit Reporting. 0 Reporting of single-bit RAM corrections is disabled. 1 Reporting of single-bit RAM corrections is enabled. This bit can only be set if the SoC-configurable input enable signal is asserted. The occurrence of a single-bit RAM correction generates a MCM ECC interrupt request as signalled by the assertion of ESR[R1BC]. The address, attributes and data are also captured in the REAR, RESR, REMR, REAT and REDR registers.
4 EF1BR	Enable Flash 1-bit Reporting. 0 Reporting of single-bit flash corrections is disabled. 1 Reporting of single-bit flash corrections is enabled. This bit can only be set if the SoC-configurable input enable signal is asserted. The occurrence of a single-bit flash correction generates a MCM ECC interrupt request as signalled by the assertion of ESR[F1BC]. The address, attributes and data are also captured in the FEAR, FEMR, FEAT and FEDR registers.
3–2	Reserved, should be cleared.
1 ERNCR	Enable RAM Non-Correctable Reporting. 0 Reporting of non-correctable RAM errors is disabled. 1 Reporting of non-correctable RAM errors is enabled. The occurrence of a non-correctable multi-bit RAM error generates a MCM ECC interrupt request as signalled by the assertion of ESR[RNCE]. The faulting address, attributes and data are also captured in the REAR, RESR, REMR, REAT and REDR registers.
0 EFNCR	Enable Flash Non-Correctable Reporting. 0 Reporting of non-correctable flash errors is disabled. 1 Reporting of non-correctable flash errors is enabled. The occurrence of a non-correctable multi-bit flash error generates a MCM ECC interrupt request as signalled by the assertion of ESR[FNCE]. The faulting address, attributes and data are also captured in the FEAR, FEMR, FEAT and FEDR registers.

13.2.2.15.2 ECC Status Register (ESR)

The ECC Status Register is an 8-bit control register for signaling which types of properly-enabled ECC events have been detected. The ESR signals the last, properly-enabled memory event to be detected. The generation of the MCM ECC interrupt request is defined by the boolean equation:

```
MCM_ECC_IRQ
    =  ECR[ER1BR] &  ESR[R1BC]// ram,   1-bit correction
    |  ECR[EF1BR] &  ESR[F1BC]// flash, 1-bit correction
    |  ECR[ERNCR] &  ESR[RNCE]// ram,   noncorrectable error
    |  ECR[EFNCR] &  ESR[FNCE]// flash, noncorrectable error
```

where the combination of a properly-enabled category in the ECR and the detection of the corresponding condition in the ESR produces the interrupt request.

The MCM allows a maximum of one bit of the ESR to be asserted at any given time. This preserves the association between the ESR and the corresponding address and attribute registers, which are loaded on each occurrence of an properly-enabled ECC event. If there is a pending ECC interrupt and another properly-enabled ECC event occurs, the MCM hardware automatically handles the ESR reporting, clearing the previous data and loading the new state and thus guaranteeing that only a single flag is asserted.

To maintain the coherent software view of the reported event, the following sequence in the MCM error interrupt service routine is suggested:

1. Read the ESR and save it.
2. Read and save all the address and attribute reporting registers.
3. Re-read the ESR and verify the current contents matches the original contents. If the two values are different, go back to step 1 and repeat.
4. When the values are identical, write a 1 to the asserted ESR flag to negate the interrupt request.

See Figure 13-24 and Table 13-21 for the ECC Status Register definition.

Register address: MCM Base + 0x47

	7	6	5	4	3	2	1	0
R	0	0	R1BC	F1BC	0	0	RNCE	FNCE
W								
Reset	0	0	0	0	0	0	0	0

Figure 13-24. ECC Status (ESR) Register

In the event that multiple status flags are signaled *simultaneously,* MCM records the event with the R1BC as highest priority, then F1BC, then RNCE, and finally FNCE.

Table 13-21. ESR Field Descriptions

Field	Description
7–6	Reserved, should be cleared.
5 R1BC	RAM 1-bit Correction. 0 No reportable single-bit RAM correction has been detected. 1 A reportable single-bit RAM correction has been detected. This bit can only be set if ECR[EPR1BR] is asserted. The occurrence of a properly-enabled single-bit RAM correction generates a MCM ECC interrupt request. The address, attributes and data are also captured in the REAR, RESR, REMR, REAT and REDR registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect.
4 F1BC	Flash 1-bit Correction. 0 No reportable single-bit flash correction has been detected. 1 A reportable single-bit flash correction has been detected. This bit can only be set if ECR[EPF1BR] is asserted. The occurrence of a properly-enabled single-bit flash correction generates a MCM ECC interrupt request. The address, attributes and data are also captured in the FEAR, FEMR, FEAT and FEDR registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect.
3–2	Reserved, should be cleared.
1 RNCE	RAM Non-Correctable Error. 0 No reportable non-correctable RAM error has been detected. 1 = A reportable non-correctable RAM error has been detected. The occurrence of a properly-enabled non-correctable RAM error generates a MCM ECC interrupt request. The faulting address, attributes and data are also captured in the REAR, RESR, REMR, REAT and REDR registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect.
0 FNCE	Flash Non-Correctable Error. 0 No reportable non-correctable flash error has been detected. 1 A reportable non-correctable flash error has been detected. The occurrence of a properly-enabled non-correctable flash error generates a MCM ECC interrupt request. The faulting address, attributes and data are also captured in the FEAR, FEMR, FEAT and FEDR registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect.

13.2.2.15.3 ECC Error Generation Register (EEGR)

The ECC Error Generation Register is a 16-bit control register used to force the generation of single- and double-bit data inversions in the memories with ECC, most notably the RAM. This capability is provided for two purposes:

- It provides a software-controlled mechanism for “injecting” errors into the memories during data writes to verify the integrity of the ECC logic.
- It provides a mechanism to allow testing of the software service routines associated with memory error logging.

It should be noted that while the EEGR is associated with the RAM, similar capabilities exist for the flash, i.e., the ability to program the non-volatile memory with single- or double-bit errors is supported for the same two reasons previously identified.

For both types of memories (RAM and flash), the intent is to generate errors during data write cycles, such that subsequent reads of the corrupted address locations generate ECC events, either single-bit corrections or double-bit non-correctable errors that are terminated with an error response.

The enabling of these error generation modes requires the same SoC-configurable input enable signal (as that used to enable single-bit correction reporting) be asserted.

See Figure 13-25 and Table 13-22 for the ECC Configuration Register definition.

Register address: MCM Base + 0x4a

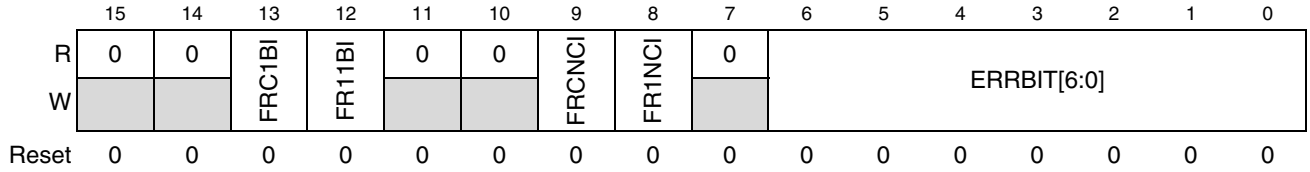


Figure 13-25. ECC Error Generation (EEGR) Register

If an attempt to force a non-correctable inversion (by asserting EEGR[FRCNCI] or EEGR[FRC1NCI]) and EEGR[ERRBIT] equals 64, then no data inversion will be generated.

Table 13-22. EEGR Field Descriptions

Field	Description
15–14	Reserved, should be cleared.
13 FRC1BI	Force RAM Continuous 1-Bit Data Inversions. 0 No RAM continuous 1-bit data inversions are generated. 1 1-bit data inversions in the RAM are continuously generated. The assertion of this bit forces the RAM controller to create 1-bit data inversions, as defined by the bit position specified in ERRBIT[6:0], continuously on every write operation. The normal ECC generation takes place in the RAM controller, but then the polarity of the bit position defined by ERRBIT is inverted to introduce a 1-bit ECC event in the RAM. After this bit has been enabled to generate another continuous 1-bit data inversion, it must be cleared before being set again to properly re-enable the error generation logic. This bit can only be set if the same SoC configurable input enable signal (as that used to enable single-bit correction reporting) is asserted.

Table 13-22. EEGR Field Descriptions (Continued)

Field	Description
12 FR11BI	Force RAM One 1-bit Data Inversion. 0 No RAM single 1-bit data inversion is generated. 1 One 1-bit data inversion in the RAM is generated. The assertion of this bit forces the RAM controller to create one 1-bit data inversion, as defined by the bit position specified in ERRBIT[6:0], on the first write operation after this bit is set. The normal ECC generation takes place in the RAM controller, but then the polarity of the bit position defined by ERRBIT is inverted to introduce a 1-bit ECC event in the RAM. After this bit has been enabled to generate a single 1-bit data inversion, it must be cleared before being set again to properly re-enable the error generation logic. This bit can only be set if the same SoC configurable input enable signal (as that used to enable single-bit correction reporting) is asserted.
11–10	Reserved, should be cleared.
9 FRCNCI	Force RAM Continuous Non-correctable Data Inversions. 0 No RAM continuous 2-bit data inversions are generated. 1 2-bit data inversions in the RAM are continuously generated. The assertion of this bit forces the RAM controller to create 2-bit data inversions, as defined by the bit position specified in ERRBIT[6:0] and the overall odd parity bit, continuously on every write operation. After this bit has been enabled to generate another continuous non-correctable data inversion, it must be cleared before being set again to properly re-enable the error generation logic. The normal ECC generation takes place in the RAM controller, but then the polarity of the bit position defined by ERRBIT and the overall odd parity bit are inverted to introduce a 2-bit ECC error in the RAM.
8 FR1NCI	Force RAM One Non-correctable Data Inversions. 0 No RAM single 2-bit data inversions are generated. 1 One 2-bit data inversion in the RAM is generated. The assertion of this bit forces the RAM controller to create one 2-bit data inversion, as defined by the bit position specified in ERRBIT[6:0] and the overall odd parity bit, on the first write operation after this bit is set. The normal ECC generation takes place in the RAM controller, but then the polarity of the bit position defined by ERRBIT and the overall odd parity bit are inverted to introduce a 2-bit ECC error in the RAM. After this bit has been enabled to generate a single 2-bit error, it must be cleared before being set again to properly re-enable the error generation logic.

Table 13-22. EEGR Field Descriptions (Continued)

Field	Description
7	Reserved, should be cleared.
6–0 ERRBIT[6:0]	Error Bit Position. The vector defines the bit position which is complemented to create the data inversion on the write operation. For the creation of 2-bit data inversions, the bit specified by this field plus the odd parity bit of the ECC code are inverted. The RAM controller follows a vector bit ordering scheme where LSB=0. Errors in the ECC syndrome bits can be generated by setting this field to a value greater than the RAM width. For example, consider a 32-bit RAM implementation. The 32-bit ECC approach requires 7 code bits for a 32-bit word. For PRAM data width of 32 bits, the actual SRAM (32b data + 7b for ECC) = 39 bits. The following association between the ERRBIT field and the corrupted memory bit is defined: if ERRBIT = 0, then RAM[0] of the odd bank is inverted if ERRBIT = 1, then RAM[1] of the odd bank is inverted ... if ERRBIT = 31, then RAM[31] of the odd bank is inverted if ERRBIT = 64,then ECC Parity[0] of the odd bank is inverted if ERRBIT = 65,then ECC Parity[1] of the odd bank is inverted ... if ERRBIT = 70,then ECC Parity[6] of the odd bank is inverted For ERRBIT values of 32 to 63 and greater than 70, no bit position is inverted.

The only allowable values for the 4 control bit enables {FR11BI, FRC1BI, FRCNCI, FR1NCI} are {0,0,0,0}, {1,0,0,0}, {0,1,0,0}, {0,0,1,0} and {0,0,0,1}. All other values result in undefined behavior.

13.2.2.15.4 Flash ECC Address Register (FEAR)

The FEAR is a 32-bit register for capturing the address of the last, properly-enabled ECC event in the flash memory. Depending on the state of the ECC Configuration Register, an ECC event in the flash causes the address, attributes and data associated with the access to be loaded into the FEAR, FEMR, FEAT and FEDR registers, and the appropriate flag (F1BC or FNCE) in the ECC Status Register to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored. See Figure 13-26 and Table 13-23 for the Flash ECC Address Register definition.

Register address: MCM Base + 0x50

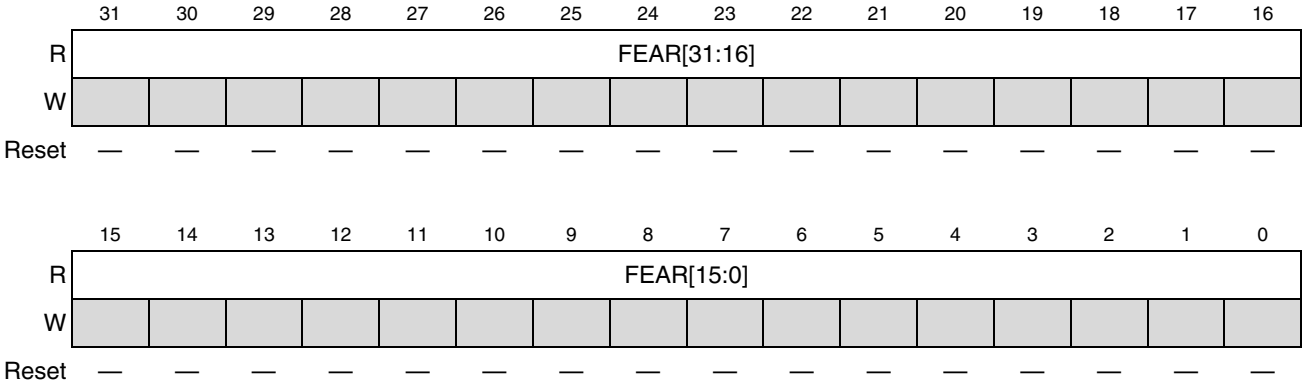


Figure 13-26. Flash ECC Address (FEAR) Register

Table 13-23. FEAR Field Descriptions

Field	Description
31–0 FEAR[31:0]	Flash ECC Address Register. This 32-bit register contains the faulting access address of the last, properly-enabled flash ECC event.

13.2.2.15.5 Flash ECC Master Number Register (FEMR)

The FEMR is a 4-bit register for capturing the AXBS bus master number of the last, properly-enabled ECC event in the flash memory. Depending on the state of the ECC Configuration Register, an ECC event in the flash causes the address, attributes and data associated with the access to be loaded into the FEAR, FEMR, FEAT and FEDR registers, and the appropriate flag (F1BC or FNCE) in the ECC Status Register to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored. See Figure 13-27 and Table 13-24 for the Flash ECC Master Number Register definition.

Register address: MCM Base + 0x56

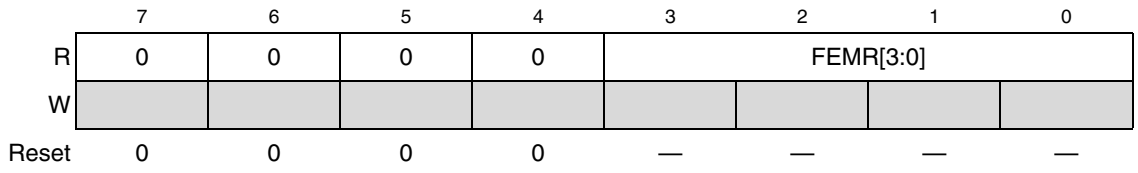


Figure 13-27. Flash ECC Master Number (FEMR) Register

Table 13-24. PFEMR Field Descriptions

Field	Description
7–4	Reserved, should be cleared.
3–0 FEMR[3:0]	Flash ECC Master Number Register. This 4-bit register contains the AXBS bus master number of the faulting access of the last, properly-enabled flash ECC event.

13.2.2.15.6 Flash ECC Attributes Register (FEAT)

The FEAT is an 8-bit register for capturing the AXBS bus master attributes of the last, properly-enabled ECC event in the flash memory. Depending on the state of the ECC Configuration Register, an ECC event in the flash causes the address, attributes and data associated with the access to be loaded into the FEAR, FEMR, FEAT and FEDR registers, and the appropriate flag (F1BC or FNCE) in the ECC Status Register to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored. See Figure 13-28 and Table 13-25 for the Flash ECC Attributes Register definition.

Register address: MCM Base + 0x57

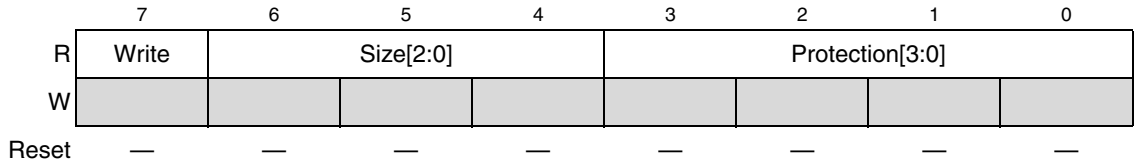


Figure 13-28. Flash ECC Attributes (FEAT) Register

Table 13-25. PFEAT Field Descriptions

Field	Description
7 Write	AMBA-AHB HWRITE. 0 AMBA-AHB read access 1 AMBA-AHB write access
6–4 Size[2:0]	AMBA-AHB HSIZE[2:0]. 000 8-bit AMBA-AHB access 001 16-bit AMBA-AHB access 010 32-bit AMBA-AHB access 011 Reserved 1xx Reserved
3–0 Protection[3:0]	AMBA-AHB HPROT[3:0]. Protection[3]: Cacheable 0 Non-cacheable 1 Cacheable Protection[2]: Bufferable 0 Non-bufferable 1 Bufferable Protection[1]: Mode 0 User mode 1 Supervisor mode Protection[0]: Type 0 I-Fetch 1 Data

13.2.2.15.7 Flash ECC Data Register (FEDR)

The FEDR is a 32-bit register for capturing the data associated with the last, properly-enabled ECC event in the flash memory. Depending on the state of the ECC Configuration Register, an ECC event in the flash causes the address, attributes and data associated with the access to be loaded into the FEAR, FEMR, FEAT and FEDR registers, and the appropriate flag (F1BC or FNCE) in the ECC Status Register to be asserted.

The data captured on a multi-bit non-correctable ECC error is undefined.

This register can only be read from the IPS programming model; any attempted write is ignored. See Figure 13-29 and Table 13-26 for the Flash ECC Data Register definition.

Figure 13-29. Flash ECC Data (FEDR) Register

Register address: MCM Base +0x5c

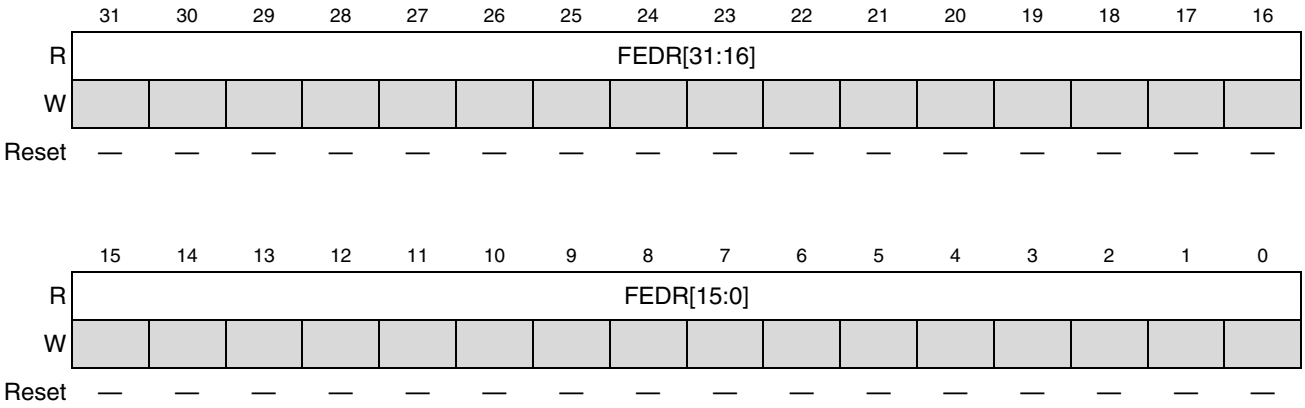


Figure 13-30. Platform Flash ECC Data (PFEDR) Register

Table 13-26. FEDR Field Descriptions

Field	Description
31–0 FEDR	Flash ECC Data Register. This 64-bit register contains the data associated with the faulting access of the last, properly-enabled flash ECC event. The register contains the data value taken directly from the data bus.

13.2.2.15.8 RAM ECC Address Register (REAR)

The REAR is a 32-bit register for capturing the address of the last, properly-enabled ECC event in the RAM memory. Depending on the state of the ECC Configuration Register, an ECC event in the RAM causes the address, attributes and data associated with the access to be loaded into the REAR, RESR, REMR, REAT and REDR registers, and the appropriate flag (R1BC or RNCE) in the ECC Status Register to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored. See Figure 13-31 and Table 13-27 for the RAM ECC Address Register definition.

Register address: MCM Base + 0x60

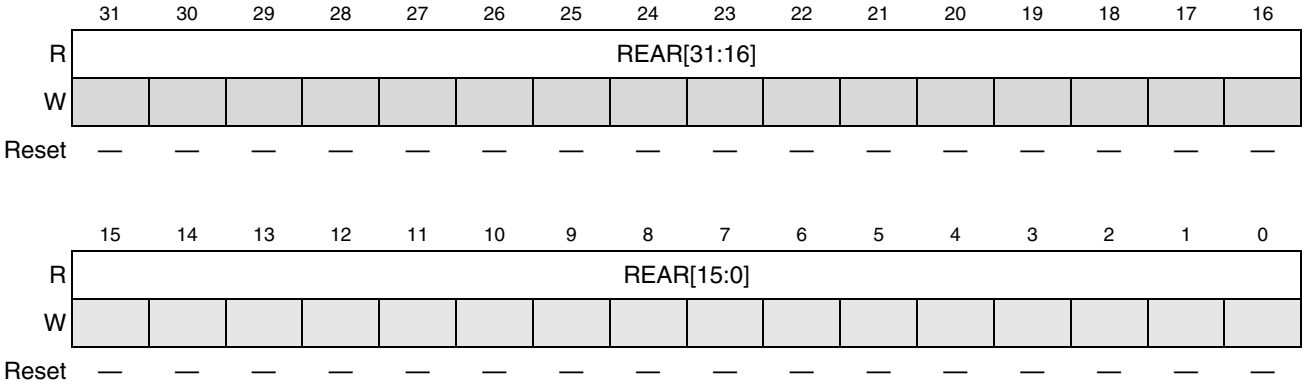


Figure 13-31. RAM ECC Address (REAR) Register

Table 13-27. REAR Field Descriptions

Field	Description
31–0 REAR[31:0]	RAM ECC Address Register. This 32-bit register contains the faulting access address of the last, properly-enabled RAM ECC event.

13.2.2.15.9 RAM ECC Syndrome Register (RESR)

The RESR is an 8-bit register for capturing the error syndrome of the last, properly-enabled ECC event in the RAM memory. Depending on the state of the ECC Configuration Register, an ECC event in the RAM causes the address, attributes and data associated with the access to be loaded into the REAR, RESR, REMR, REAT and REDR registers, and the appropriate flag (R1BC or RNCE) in the ECC Status Register to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored. See Figure 13-32 and Table 13-28 for the RAM ECC Syndrome Register definition.

Register address: MCM Base + 0x65

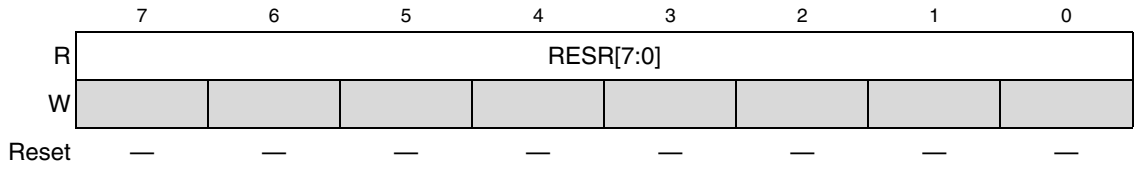


Figure 13-32. RAM ECC Syndrome (RESR) Register

Table 13-28. RESR Field Descriptions

Field	Description
7–0 RESR[7:0]	RAM ECC Syndrome Register. This 8-bit syndrome field includes 6 bits of Hamming decoded parity plus an odd-parity bit for the entire 39-bit (32-bit data + 7 ECC) code word. The upper 7 bits of the syndrome specify the exact bit position in error for single-bit correctable codewords, and the combination of a non-zero 7-bit syndrome plus overall incorrect parity bit signal a multi-bit, non-correctable error. For correctable single-bit errors, the mapping shown in Table 13-29 associates the upper 7 bits of the syndrome with the data bit in error.

Table 13-29 associates the upper 7 bits of the ECC syndrome with the exact data bit in error for single-bit correctable codewords. This table follows the bit vectoring notation where the LSB=0. Note that the syndrome value of 0x01 implies no error condition but this value is not readable when the PRESR is read for the no error case.

Table 13-29. RAM Syndrome Mapping for Single-Bit Correctable Errors

RESR[7:0]	Data Bit in Error
0x00	ECC ODD[0]
0x01	No Error
0x02	ECC ODD[1]
0x04	ECC ODD[2]
0x06	DATA ODD BANK[31]
0x08	ECC ODD[3]
0x0a	DATA ODD BANK[30]
0x0c	DATA ODD BANK[29]
0x0e	DATA ODD BANK[28]
0x10	ECC ODD[4]
0x12	DATA ODD BANK[27]
0x14	DATA ODD BANK[26]
0x16	DATA ODD BANK[25]
0x18	DATA ODD BANK[24]
0x1a	DATA ODD BANK[23]
0x1c	DATA ODD BANK[22]
0x50	DATA ODD BANK[21]
0x20	ECC ODD[5]
0x22	DATA ODD BANK[20]
0x24	DATA ODD BANK[19]
0x26	DATA ODD BANK[18]

RESR[7:0]	Data Bit in Error
0x28	DATA ODD BANK[17]
0x2a	DATA ODD BANK[16
0x2c	DATA ODD BANK[15]
0x58	DATA ODD BANK[14]
0x30	DATA ODD BANK[13]
0x32	DATA ODD BANK[12]
0x34	DATA ODD BANK[11]
0x64	DATA ODD BANK[10]
0x38	DATA ODD BANK[9]
0x62	DATA ODD BANK[8]
0x70	DATA ODD BANK[7]
0x60	DATA ODD BANK[6]
0x40	ECC ODD[6]
0x42	DATA ODD BANK[5]
0x44	DATA ODD BANK[4]
0x46	DATA ODD BANK[3]
0x48	DATA ODD BANK[2]
0x4a	DATA ODD BANK[1]
0x4c	DATA ODD BANK[0]
0x03,0x05.....0x4d	Multiple bit error
> 0x4d	Multiple bit error

13.2.2.15.10 RAM ECC Master Number Register (REMR)

The REMR is a 4-bit register for capturing the AXBS bus master number of the last, properly-enabled ECC event in the RAM memory. Depending on the state of the ECC Configuration Register, an ECC event in

the RAM causes the address, attributes and data associated with the access to be loaded into the REAR, RESR, REMR, REAT and REDR registers, and the appropriate flag (R1BC or RNCE) in the ECC Status Register to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored. See Figure 13-33 and Table 13-30 for the RAM ECC Master Number Register definition.

Register address: MCM Base + 0x66

	7	6	5	4	3	2	1	0
R	0	0	0	0	REMR[3:0]			
W								
Reset	0	0	0	0	—	—	—	—

Figure 13-33. RAM ECC Master Number (REMR) Register

Table 13-30. REMR Field Descriptions

Field	Description
7–4	Reserved, should be cleared.
3–0 REMR[3:0]	RAMECC Master Number Register. This 4-bit register contains the AXBS bus master number of the faulting access of the last, properly-enabled RAM ECC event.

13.2.2.15.11 RAM ECC Attributes Register (REAT)

The REAT is an 8-bit register for capturing the AXBS bus master attributes of the last, properly-enabled ECC event in the RAM memory. Depending on the state of the ECC Configuration Register, an ECC event in the RAM causes the address, attributes and data associated with the access to be loaded into the REAR, RESR, REMR, REAT and REDR registers, and the appropriate flag (R1BC or RNCE) in the ECC Status Register to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored. See Figure 13-34 and Table 13-31 for the RAM ECC Attributes Register definition.

Register address: MCM Base + 0x67

	7	6	5	4	3	2	1	0
R	Write	Size[2:0]			Protection[3:0]			
W								
Reset	—	—	—	—	—	—	—	—

Figure 13-34. RAM ECC Attributes (REAT) Register

Table 13-31. REAT Field Descriptions

Field	Description
7 Write	AMBA-AHB HWRITE. 0 AMBA-AHB read access 1 AMBA-AHB write access
6–4 Size[2:0]	AMBA-AHB HSIZE[2:0]. 000 8-bit AMBA-AHB access 001 16-bit AMBA-AHB access 010 32-bit AMBA-AHB access 011 Reserved 1xx Reserved
3–0 Protection[3:0]	AMBA-AHB HPROT[3:0]. Protection[3]: Cacheable 0 Non-cacheable 1 Cacheable Protection[2]: Bufferable 0 Non-bufferable 1 Bufferable Protection[1]: Mode 0 User mode 1 Supervisor mode Protection[0]: Type 0 I-Fetch 1 Data

13.2.2.15.12 RAM ECC Data Register (REDR)

The REDR is a 32-bit register for capturing the data associated with the last, properly-enabled ECC event in the RAM memory. Depending on the state of the ECC Configuration Register, an ECC event in the RAM causes the address, attributes and data associated with the access to be loaded into the REAR, RESR, REMR, REAT and REDR registers, and the appropriate flag (R1BC or RNCE) in the ECC Status Register to be asserted.

The data captured on a multi-bit non-correctable ECC error is undefined.

This register can only be read from the IPS programming model; any attempted write is ignored. See Figure 13-35 and Table 13-32 for the RAM ECC Data Register definition.

Register address: MCM Base +0x6c

Figure 13-35. RAM ECC Data (REDR) Register

Table 13-32. REDR Field Descriptions

Field	Description
31–0 REDR[31:0]	RAM ECC Data Register. This 32-bit register contains the data associated with the faulting access of the last, properly-enabled RAM ECC event. The register contains the data value taken directly from the data bus.

13.2.2.16 Core Data Fault Recovery Registers

To aid in recovery from certain types of data access errors, the MCM module optionally supports a number of registers which capture access address, attribute and data information on bus cycles terminated with an error response. These registers can then be read during the resulting exception service routine and the appropriate recovery performed.

The details on the core fault recovery registers are provided in the subsequent sections. It is important to note these registers are used to capture fault recovery information *only on data access faults*, *i.e.*, *no information is captured on instruction fetch faults*. If the design does not include the fault recovery registers, these addresses are reserved locations within the MCM's programming model.

13.2.2.16.1 Core Fault Address Register (CFADR)

The CFADR is a 32-bit register for capturing the address of the last core data access which was terminated with an error response. This register can only be read from the IPS programming model; any attempted write is ignored. See Figure 13-36 and Table 13-33 for the Core Fault Address Register definition.

Register address: MCM Base + 0x70

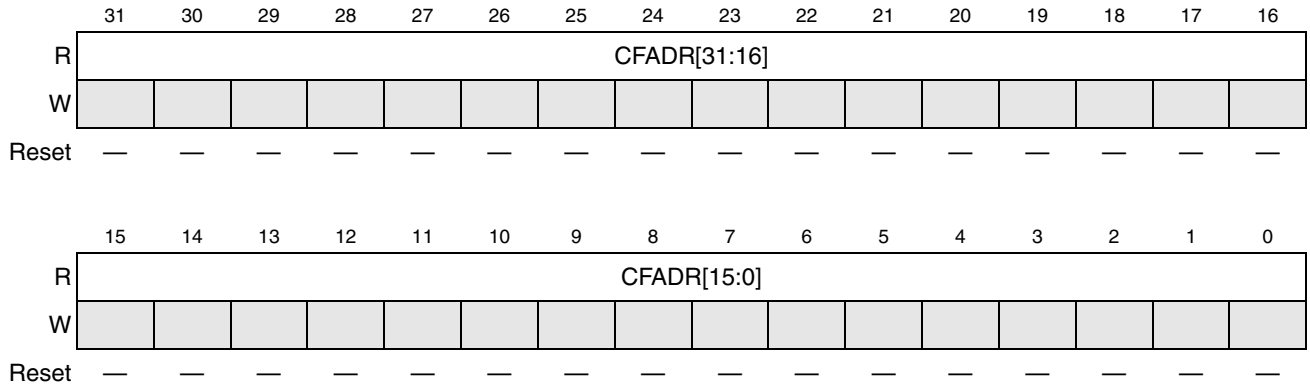


Figure 13-36. Core Fault Address (CFADR) Register

Table 13-33. CFADR Field Descriptions

Field	Description
31–0 CFADR[31:0]	Core Fault Address Register. This 32-bit register contains the faulting address of the last core data access terminated with an error response.

13.2.2.16.2 Core Fault Location/Interrupt Enable Register (CFLOC1)

The CFLOC1 is an 8-bit register with multiple functions for SPP implementations. For SPP_Ref_CFx designs, this register contains a 1-bit enable to generate an error interrupt on the detection of a faulted processor AHB system bus cycle. See Table 13-37 and Table 13-34 for the Core Fault Location Register definition.

Register address: MCM Base + 0x75

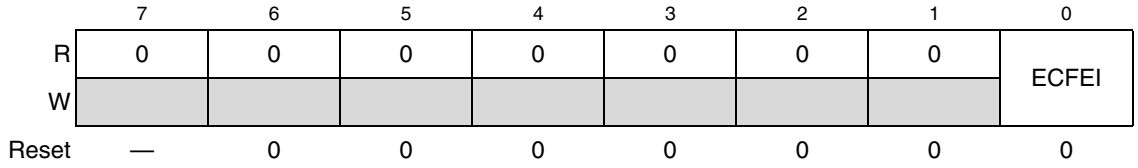


Figure 13-37. Core Fault Location 1 (CFLOC1) Register

Table 13-34. Core Fault Location 1 (CFLOC1) Field Descriptions

Field	Description
7–1	Reserved, should be cleared.
0 ECFEI	Enable Core Fault Error Interrupt. 0 Do not generate an error interrupt on a faulted processor system bus cycle. 1 Generate an error interrupt on a faulted processor system bus cycle.

13.2.2.16.3 Core Fault Location Register (CFLOC)

The CFLOC is a 1-bit register indicating the exact location of the captured fault information: a AMBA-AHB data access or a data access to a tightly-coupled core local memory. This register can only be read from the IPS programming model; any attempted write is ignored. See Table 13-38 and Table 13-35 for the Core Fault Location Register definition.

Register address: MCM Base + 0x76

	7	6	5	4	3	2	1	0
R	LocalErr	0	0	0	0	0	0	0
W								
Reset	—	0	0	0	0	0	0	0

Figure 13-38. Core Fault Location (CFLOC) Register

Table 13-35. CFLOC Field Descriptions

Field	Description
7 LocalErr	Bus Error Indicator. 0 Error occurred on the AMBA-AHB bus 1 Error occurred on the processor core's local bus

13.2.2.16.4 Core Fault Attributes Register (CFATR)

The CFATR is an 8-bit register for capturing the processor's attributes of the last faulted data access to the AMBA-AHB or core's local bus.

This register can only be read from the IPS programming model; any attempted write is ignored. See Figure 13-39 and Table 13-36 for the Core Fault Attributes Register definition.

Register address: MCM Base + 0x77

	7	6	5	4	3	2	1	0
R	Write	Size[2:0]			Protection[3:0]			
W								
Reset	—	—	—	—	—	—	—	—

Figure 13-39. Core Fault Attributes (CFATR) Register

Table 13-36. CFATR Field Descriptions

Field	Description
7 Write	WRITE. 0 Core read access 1 Core write access
6–4 Size[2:0]	SIZE[2:0]. 000 8-bit core access 001 16-bit core access 010 32-bit core access 011 64-bit core access 1xx Reserved
3–0 Protection[3:0]	PROT[3:0]. Protection[3]: Cacheable 0 Non-cacheable 1 Cacheable Protection[2]: Bufferable 0 Non-bufferable 1 Bufferable Protection[1]: Mode 0 User mode 1 Supervisor mode Protection[0]: Type 1 Data

13.2.2.16.5 Core Fault Data Register (CFDTR)

The CFDTR is a 32-bit register for capturing the data associated with the last faulted processor write data access from the device's AMBA-AHB bus. The CFDTR is valid only for faulted AMBA-AHB write accesses. *The contents of this register are **not** valid if the last fault occurred on the core's local bus (CFLOC[LocalErr] = 1). This register is not updated on AMBA-AHB read access faults.*

This register can only be read from the IPS programming model; any attempted write is ignored. See Figure 13-40 and Table 13-37 for the Core Fault Data Register definition.

Register address: MCM Base + 0x7c

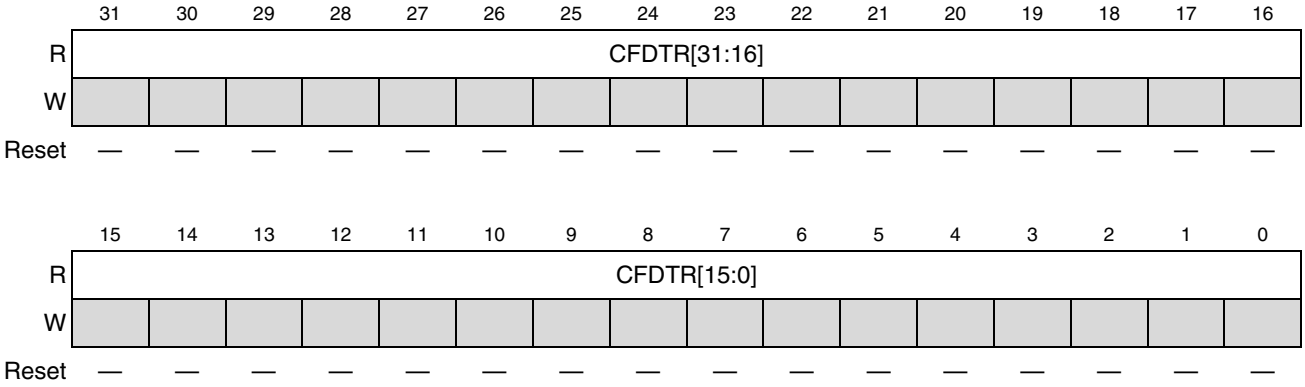


Figure 13-40. Core Fault Data (CFDTR) Register

Table 13-37. CFDTR Field Descriptions

Field	Description
31–0 CFDTR[31:0]	Core Fault Data Register. This 32-bit register contains the data associated with the faulting access of the last AMBA-AHB write access. The register contains the data value taken directly from the device write data bus.

13.3 MCM as Implemented on MAC7200

13.3.1 MCM Introduction

The Miscellaneous Control Module implements a number of system level features, including the following:

- Program-visible information on the device configuration and revision.
- Software watchdog timer (SWT) with programmable system reset or interrupt response which runs on a separate, asynchronous clock. The SWT also supports a windowed time period, requiring the writing of the watchdog key within a specified time window. The source for the SWT clock is controlled by the Clock and Reset Generator.
- Wakeup control for entering and exiting sleep modes, including an enable and an interrupt priority mask.
- Access address information for faulted memory accesses. When a bus abort occurs in the system, the exact cause of the abort can be determined by reading the Core Data Fault Recovery Registers in the MCM.
- Address map for the crossbar switch (AXBS), which is used to steer access to the Crossbar slave ports in order to provide remapping operations. Following Reset, the contents of the AAMR register are determined by the mode entered as a result of the default slave mapping, but may be configured by the application code in order to remap the Flash, External bus and RAM between AXBS Slave Port 0 and Port 1. Please refer to Chapter 9, “Device Memory Map” for further details.

For the MAC7200 family of devices, only certain crossbar switch re-mappings are valid. The reset value of the register depends on the Chip Mode selected and the security state of the device, and is discussed in detail in Chapter 9, “Device Memory Map”. After reset, the AAMR register may be re-programmed to change the memory map of the system as follows:

Table 13-38. AAMR Register Configurability [1]

AAMR Bits	Base Address	Allowed Values
31:28	$E000 0000	$f (Peripheral Space)
27:24	$C000 0000	$0 (Unused)
23:20	$A000 0000	$d (BAM)
19:16	$8000 0000	$0 (Unused)
15:12	$6000 0000	$0 (Unused)
11:8	$4000 0000	$b (SRAM)
7:4	$2000 0000	$8 (Program Flash), $9 (External Bus), $b (SRAM)
3:0	$0000 0000	$8 (Program Flash), $9 (External Bus), $b (SRAM)

1. Not all re-mappings may be available in all Chip Modes/Security states. Please refer to Chapter 9, “Device Memory Map for more details on valid memory map configurations.

NOTE

The SRAM is fixed at address $4000 0000, and may be re-mapped to either $0000 0000 or $2000 0000. When this is done, the SRAM will appear at *both* addresses in the memory map.

Consult the MCM Block Guide for further information about configuring and using the MCM.

13.3.2 MCM Features

The MCM module is an on-platform IPI module that implements several varied SoC level features, including the following:

13.3.2.1 Processor Core Type (PCT)

This 16-bit register identifies the revision ID of the platform/core pair, as follows:

Table 13-39. Processor Core Type (PCT) Values

Core	PCT	Comment
ARM7	$A700	Value for the MAC72xx
ARM9	$A900	
ARM11	$A110	
ColdFire v2	$CF20	
ColdFire v3	$CF30	

Table 13-39. Processor Core Type (PCT) Values (Continued)

Core	PCT	Comment
ColdFire v4	$CF40	
ColdFire v5	$CF50	
Z650 PowerPC	$E650	

13.3.2.2 Revision ID (REV)

This 16-bit register identifies the revision ID of the device.

Table 13-40. MAC72xx PCT and REV registers

Device	PCT	REV
MAC72x2 (0M84D Maskset)	$A700	$7220 [1]
MAC72x2 (1M84D Maskset)	$A700	$7220 [1]
MAC72x1 (0M19G Maskset)	$A700	$7230

1. Due to erratum MUCts03562, 0M84D and 1M84D masks share the same revision number

13.3.2.3 AXBS Master/Slave Configuration

The AMC and ASC registers in the MCM indicate the presence of AXBS masters and slaves. Please refer to Chapter 15, “MAC7200 Crossbar Switch (AXBS)” for a listing of which AXBS masters/slaves are present on the MAC72xx.

13.3.2.4 Misc. Reset Status Register (MRSR)

On the MAC72xx, all reset sources all consolidated into the CRG module. Therefore, this feature in the MCM will not be used, and will be removed from the MCM Block Guide for the MAC72xx.

13.3.2.5 Misc. Wakeup Control Register (MWCR)

Since the MAC72xx only supports DOZE mode, this register is used to control wake-up from DOZE mode with an interrupt. STOP mode is not supported.

13.3.2.6 Software Watchdog Timer (SWT)

The SWT will interrupt the processor periodically to request servicing (in order to prevent software deadlock or runaway). The SWT includes enable/disable, interrupt/reset and programmable period functionality.

The SWT interrupt is connected to on-platform interrupt Request #17, and the SWT reset signal will be fed off-platform to the CRG to allow for soft-reset on a SWT timeout. The SWT clock may be fed from either the oscillator clock, in normal mode, or from the PLL, when the system enters Self Clock Mode.

By programming the **BDMCTL** register in the CRG module, the system can automatically stop and start the SWT counters when the system enters debug mode.

13.3.2.7 AXBS Address Map Register (AAMR)

This 32-bit register controls the address mapping for each of the 8 possible AXBS slaves. Please refer to Section 9.16.4, “Programming the AAMR register in the MCM” for more details on the AAMR functionality.

13.3.2.8 Misc. User-Defined Control Register (MUDCR)

The MUDCR register will be used on the MAC72xx as the Write Access Control (WACC) register for the main Flash array. Please refer to Section 18.7.4.2, “Flash MCM Registers” for more information on the WACC register.

13.3.2.9 ECC

Both the on-chip RAM and Flash memories include ECC with 2-bit detection/1-bit correction. The MCM ECC registers include a variety of features to utilize the ECC, including full reporting features. Because the ARM7 core supports Pre-fetch (Instruction) and Data aborts, all code for servicing both of these exceptions should make use of the ECC reporting registers in the MCM.

13.3.2.10 Fault Registers

The MCM provides several registers that can be used by the Pre-fetch Abort and Data Abort exception handlers to determine the exact location and type of an illegal instruction or data fetch. Please refer to Section 7.2.4, “Prefetch (Instruction) Abort” for a list of bus abort sources.

13.3.2.11 Non-Maskable Interrupt (NMI)

Please refer to Section 7.3.4, “Non-Maskable Interrupt (NMI)” for details on the Non-Maskable Interrupt. The MCM allows enabling/disabling of the NMI feature, as well as polarity configuration and NMI status information.

13.3.3 MCM External Pins

There are no MCM signals that drive or are driven from MCU pins.

13.3.4 MCM Bus Aborts

The MCM module supports Peripheral Bus bus aborts, and enforces the following memory map:

Table 13-41. MCM Bus Aborts

Abort	Allowed
	$0000-$000b
$000c-$000e	
	$000f
$0010-$0012	
	$0013
$0014-$0015	

Table 13-41. MCM Bus Aborts (Continued)

Abort	Allowed
	$0016-$0017
$0018-$001a	
	$001b
$001c-$001e	
	$001f-$0027
$0028-$002a	
	$002b
$002c-$0042	
	$0043
$0044-$0046	
	$0047
$0048-$0049	
	$004a-$004b
$004c-$004f	
	$0050-$0053
$0044-$0046	
	$0047
$0054-$0055	
	$0056-$0063
$0064	
	$0065-$0073
$0074-$0075	
	$0076-$0077
$0078-$007b	
	$007c-$007f
$0080-$3fff	

If any part of a read or write falls within an aborted region, the entire transfer is aborted. For example, a 32-bit read or write to address $0074 would be aborted.

In addition, the following transfers will be aborted:

- Byte write to 16-bit reg (swtcr, eegr)
- Byte or word write to 32-bit reg (paamr, ppmr*)

Supervisor Access: Unused

13.3.5 MCM Differences from MAC71xx

- Obsoleted use of the MRSR register
- Added ECC related registers and reporting
  - General ECC: Added ECR, ESR, EEGR registers
  - Flash ECC: Added FEAR, FEMR, FEAT, FEDRH, FEDRL registers
  - SRAM ECC: Added REAR, REDRH, REDRL registers

- Added NMI functionality and NMICR register
- Changed AAMR reset value from $F000_xxxx to $F0D0_xxxx
- Added MUDCR register (used as the Flash main array WACC register)
- REV register value is changed to $7210 (PCT is unchanged)
- Single bit ECC error reporting is now enabled (**enb_ecc_rpt** = 1)
- ECC interrupt (**ipi_ecc_int**) now used (with Request $0017)

13.3.6 MCM Application Usage

Unless explicitly described below, all application information can be found in the MCM Block Guide.

13.3.6.1 Enabling the MCM

It is not necessary to enable the MCM before it can be used.

13.3.6.2 ECC

For special memory initialization requirements of the SRAM, please refer to Section 19.1.6, “SRAM Application Usage”.

13.3.6.3 Flash

Please refer to Section 18.7.4.2, “Flash MCM Registers”.

13.3.6.4 AAMR

For information on how to use the AAMR register, please refer to Section 9.16.4, “Programming the AAMR register in the MCM”.

13.3.6.5 NMI

For more information on NMI implementation and usage, please refer to Section 7.3.4, “Non-Maskable Interrupt (NMI)”. In particular, the following usage points should be noted:

- Existing Instruction Abort service routines must be modified in order to use the NMI feature.
- The NMI may be used either in a re-entrant or non re-entrant manner, selectable in software.
- Once the NMI is enabled, and the polarity set, it can not be disabled or the polarity changed without resetting the device.

13.3.6.6 REV Register

The lower 4 bits of the REV register are used to indicate the mask revision for a particular device, and are also readable directly through the JTAG interface.

Chapter 14
SPP Interrupt Controller Module for ARM (SPP_INTC_ARM)

14.1 Introduction

14.1.1 Overview

The ARM_INTC is a highly-programmable interrupt controller, collecting both on-platform and off-platform interrupt requests, mapping the requests into 16 priority levels, and then signalling the ARM processor when a properly enabled non-masked request is active. In response to the service routine's memory mapped interrupt acknowledge read cycle, the controller returns a unique vector for each interrupt request and automatically manages masking of lower level requests. As any interrupt source can be assigned to any of the sixteen priority levels, and the interrupt controller enables the definition of which priority levels are assigned to Fast or Normal interrupts, each interrupt source can be selected to generate either a Fast or Normal interrupt to the core by its assigned level.

Normal or Fast interrupts can enable the cores bus master priority to be elevated on the crossbar switch to assist with accelerating the handling of interrupts of the specified type.

The MAC7200 family of devices implement one interrupt controller which supports 64 interrupt requests. The interrupt controller is accessed via the peripheral bus for both configuration and for fetching the interrupt vector.

Consult Section 14.7, "The Interrupt Controller Module (INTC)" for information about the vector interrupt controller implemented in the core Platform and Chapter 7, "Exceptions" for a list of the possible interrupt sources in the system.

14.2 INTC Features

- 64 vectored interrupt sources.
- Interrupt sources available from internal peripherals, eDMA controller, software watchdog timer and external sources.
- Supports 16 interrupt levels (0-15) with 64 priorities per level (1-64), to allow maximum flexibility in configuring the system. Every interrupt source can be programmed to any interrupt level. Priorities within a level are hard-coded in the MCU.
- Multiple level interrupt nesting.
- Hardware support for first nesting level.
- Normal and Fast interrupts supported with software programmability of sources for both Fast and Normal Interrupts
- Memory Map: 32-bit peripheral with 256 bytes, byte addressable

- Has purely combinatorial “wakeup” signal for system wakeup

The purely combinatorial “wakeup” signal will be fed directly to the system wakeup input on the CRG (OR’d with other wakeup sources, of course). For a full list of wakeup sources, please see Section 3.5, “System Wakeup”.

14.3 INTC External Pins

Table 14-1. INTC Signals

Signals	Description
XIRQ	This external pin, when placed in Primary Peripheral Mode, drives the Interrupt Request #63 directly. This interrupt source was chosen, because it has the highest priority (given that all interrupts are specified with the same priority level) of any interrupt source.
IRQ	This external pin, when placed in Primary Peripheral Mode, drives the Interrupt Request #62 directly.

14.4 INTC Bus Aborts

The INTC module supports Peripheral Bus bus aborts, and enforces the following memory map:

Table 14-2. INTC Bus Aborts

Abort	Allowed
	$0000-$0017
$0018-$001a	
	$001b-$001f
$0020-$003c	
	$0040-$007f
$0080-$00eb	
	$00ec
$00ed-$00ef	
	$00f0
$0080-$00eb	
	$00ec
$00ed-$00ef	
	$00f0
$00f1-$3fff	

If any part of a read or write falls within an aborted region, the entire transfer is aborted. For example, a 32-bit read or write to address $00ec would be aborted.

Supervisor Access: Unused

14.5 INTC Differences from MAC71xx

- Different mapping of interrupt sources.

Table 14-3. INTC MAC71x1 versus MAC72xx Interrupt Source Assignment

Interrupt Request	MAC71x1	MAC72xx
18	CRG	—
19–21	PIT Timer Channel 1-3	—
22	PIT Timer Channel 4 / RTI	—
23	VREG	UNUSED
24	CAN_A Message Buffer 1-13,15-32	—
25	CAN_A Message Buffer 14	—
26	CAN_A	—
27	CAN_B Message Buffer 1-13,15-32	—
28	CAN_B Message Buffer 14	—
29	CAN_B	—
31	CAN_C Message Buffer 1-13,15-32	UNUSED
32	CAN_C Message Buffer 14	UNUSED
33	CAN_C	UNUSED
34	CAN_D Message Buffer 1–13,15–32	UNUSED
35	CAN_D Message Buffer 14	UNUSED
36	CAN_D	UNUSED
37	I^2C	—
38	SPI_A	—
39	SPI_B	—
40	SCI_A	—
41	SCI_B	—
42	SCI_C	—
43	SCI_D	—
44–50	eMIOS Channel 1–8	—
51–58	eMIOS Channel 9–16	UNUSED
59	ATD_A / ATD_B	ATD_A only
60	Flash	DSPI_C
61	PIM	—
62	IRQ	—
63	XIRQ	—

14.6 INTC Application Usage

Please refer to the Interrupt Controller Block Guide for specific usage. In addition, the following special notes apply:

- All interrupt source on the MAC72xx, including external sources, must be cleared at the source.
- The Non Maskable Interrupt (NMI) control registers can be found in the MCM module, not the INTC module.
- The interrupt vector number is not the same as the physical interrupt request number. The relationship is defined as **vector_number = 64 + Request#**, where Request#=0 to 63.

- If the interrupt vector number returns less than 64 (i.e.-63 or less), this indicates a spurious interrupt. A spurious interrupt is an interrupt that has been cleared before it was serviced. Since all interrupt sources on the MAC72xx require explicit clearing, this indicates either a software problem or an external interrupt that is self clearing.

14.6.1 Enabling the INTC

It is not necessary to enable the Interrupt Controller before it can be used. However, by default after reset, no interrupts are enabled. Please refer to the INTC Block Guide for information about initialization of the Interrupt Controller.

14.7 The Interrupt Controller Module (INTC)

For the required controller, the 64 fully-programmable interrupt sources service both the on-platform peripherals as well as providing interrupt support for off-platform logic. The split between off-/on-platform interrupt sources is dependent upon the exact platform peripheral configuration and the number of sources needed to support it. Typically, the optional interrupt controllers support off-platform requests.

Figure 14-1 presents a simplified block diagram for the interrupt controller.

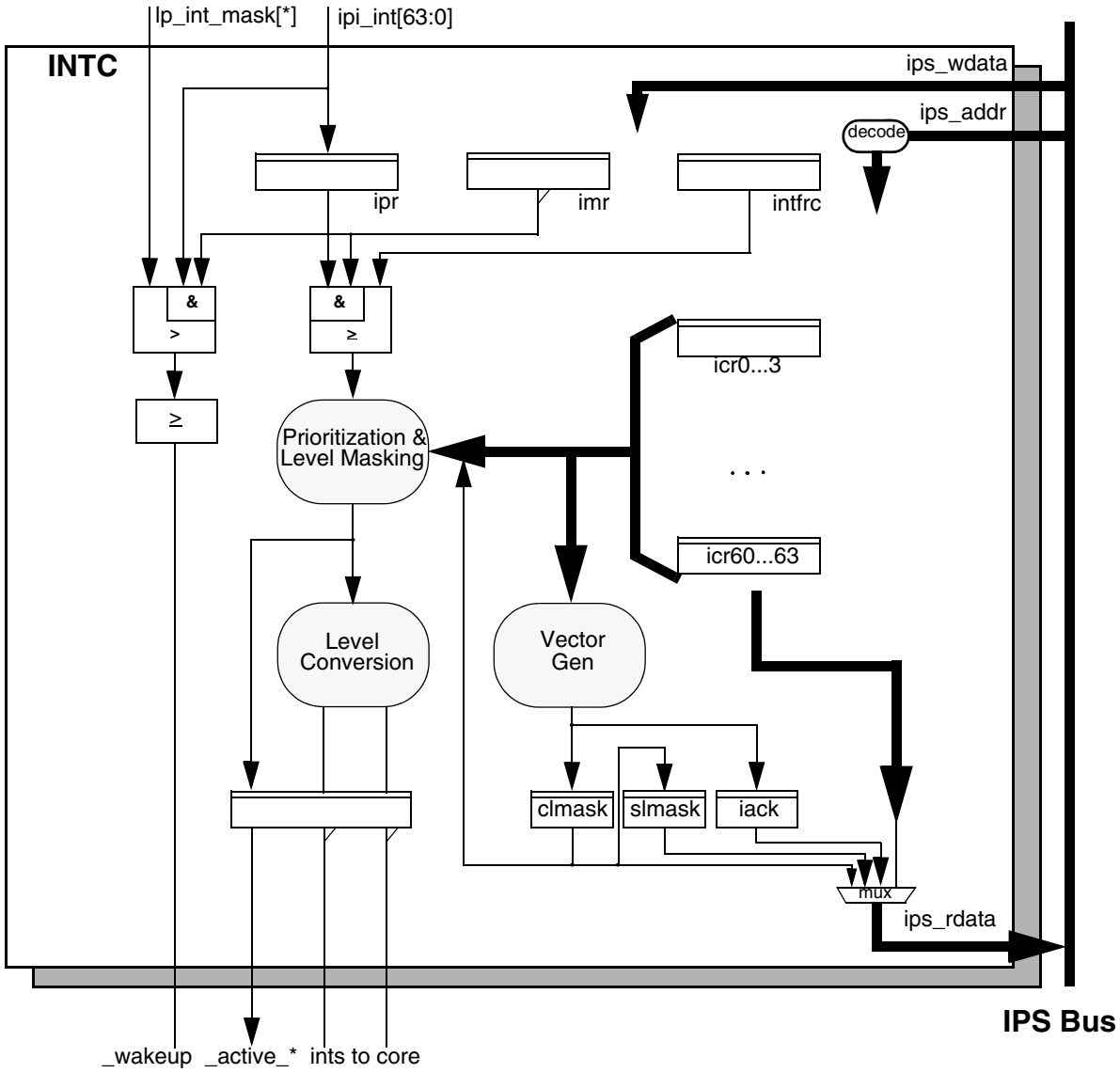


Figure 14-1. INTC Block Diagram

14.7.1 Review of ARM Interrupt Architecture

Before continuing with the specifics of the platform interrupt controller, a brief review of the interrupt architecture of the ARM core family is appropriate.

ARM cores support two direct interrupt request signals: a FIQ (fast interrupt request) and an IRQ (normal interrupt request). These two inputs are prioritized by the hardware with the FIQ being higher priority than the IRQ. Like most processor cores, interrupts are sampled once per instruction. If the FIQ input signal is asserted and enabled in the processor's status register (CPSR), then the core suspends normal execution and initiates processing of a fast interrupt exception.

During exception processing, the contents of two key registers are copied into "shadow" or banked registers. Specifically, the current program counter (R15) is copied into the banked copy of the R14 link

register named R14_fiq and the current processor status register (CPSR) value is copied into the saved processor status register (SPSR_fiq). Finally, a unique fast interrupt stack pointer, contained in R13_fiq, is activated and the core completes the exception processing by setting bits in the CPSR to disable both FIQ and IRQ interrupts, and then fetching the instruction at address 0x0000_001c. The processor continues execution in the FIQ mode, as indicated by the low-order 5 bits (the mode field) of the CPSR. While executing in this mode, banked copies of five additional general purpose registers (R[8,9,10,11,12]_fiq) are also available. The shadow copies of these registers are intended to minimize the need to save/restore the machine register state in memory using the fast interrupt stack pointer.

IRQ processing is similar with the following differences: in IRQ mode, shadow copies of only three registers are active: R13_irq (the interrupt request stack pointer), R14_irq (the PC of the interrupted instruction) and the SPSR_irq (the saved processor status register). The CPSR is configured to disable only IRQ interrupts, and the processor completes exception processing by fetching the instruction at address 0x0000_0018.

For all ARM processors, the processing of any exception forces exit from Thumb mode (if enabled), and the address loaded into the R14_{fiq,irq} link register is actually the program counter of the interrupted instruction plus 4. Thus, this instruction address must be adjusted before returning to the interrupted instruction at the completion of the service routine.

A summary of the operations associated with interrupt exception processing is shown in Figure 14-4:

Table 14-4. ARM Interrupt Exception Summary

Register	FIQ	IRQ
Interrupt Sample	Detect FIQ asserted	Detect FIQ negated, IRQ asserted
Link Register	R14_fiq = nextInst + 4	R14_irq = nextInst + 4
Saved Status Register	SPSR_fiq = CPSR	SPSR_irq = CPSR
Current Processor Status Register	CPSR[M] = 5'b10001, CPSR[T] = 1'b0, CPSR[F] = 1'b1, CPSR[I] = 1'b1	CPSR[M] = 5'b10010, CPSR[T] = 1'b0, CPSR[F] = unaffected, CPSR[I] = 1'b1
Stack Pointer	Activate R13_fiq	Activate R13_irq
Other Banked Registers	Activate R[8-12]_fiq	None
Program Counter	PC = 0x0000_001c	PC = 0x0000_0018

14.8 Memory Map/Register Definition

The register programming model for the interrupt controller is memory-mapped to a 256-byte space within the addresses serviced by the IPS controller. In the following discussion, there are a number of program-visible registers greater than 32 bits in size. For these control fields, the physical register is partitioned into two 32-bit values: a register "high" (the upper word) and a register "low" (the lower word). The nomenclature <reg_name>H and <reg_name>L is used to reference these values.

The registers and their locations are defined in Table 14-5. The INTC module does not include any logic which provides access control. Rather, this function is supported using the standard access control logic provided by the AIPS controller. Attempted accesses to the reserved locations are terminated with an error.

Table 14-5 is a 32-bit view of the INTC's memory map.

Table 14-5. INTC 32-bit Memory Map

INTC Offset	Register			
0x0000	Interrupt Pending Register High (IPRH, Requests 63-32)			
0x0004	Interrupt Pending Register Low (IPRL, Requests 31-00)			
0x0008	Interrupt Mask Register High (IMRH, Requests 63-32)			
0x000c	Interrupt Mask Register Low (IMRL, Requests 31-00)			
0x0010	Interrupt Force Register High (INTFRCH, Requests 63-32)			
0x0014	Interrupt Force Register Low (INTFRCL, Requests 31-00)			
0x0018	Reserved			Interrupt Config Register (ICONFIG)
0x001c	Set Interrupt Mask (SIMR)	Clear Interrupt Mask (CIMR)	Current Level Mask (CLMASK)	Saved Level Mask (SLMASK)
0x0020-0x003c	Reserved			
0x0040	Interrupt Control Register 0 (ICR00)	Interrupt Control Register 1 (ICR01)	Interrupt Control Register 2 (ICR02)	Interrupt Control Register 3 (ICR03)
0x0044	Interrupt Control Register 4 (ICR04)	Interrupt Control Register 5 (ICR05)	Interrupt Control Register 6 (ICR06)	Interrupt Control Register 7 (ICR07)
0x0048	Interrupt Control Register 8 (ICR08)	Interrupt Control Register 9 (ICR09)	Interrupt Control Register 10 (ICR10)	Interrupt Control Register 11 (ICR11)
0x004c	Interrupt Control Register 12 (ICR12)	Interrupt Control Register 13 (ICR13)	Interrupt Control Register 14 (ICR14)	Interrupt Control Register 15 (ICR15)
0x0050	Interrupt Control Register 16 (ICR16)	Interrupt Control Register 17 (ICR17)	Interrupt Control Register 18 (ICR18)	Interrupt Control Register 19 (ICR19)
0x0054	Interrupt Control Register 20 (ICR20)	Interrupt Control Register 21 (ICR21)	Interrupt Control Register 22 (ICR22)	Interrupt Control Register 23 (ICR23)
0x0058	Interrupt Control Register 24 (ICR24)	Interrupt Control Register 25 (ICR25)	Interrupt Control Register 26 (ICR26)	Interrupt Control Register 27 (ICR27)
0x005c	Interrupt Control Register 28 (ICR28)	Interrupt Control Register 29 (ICR29)	Interrupt Control Register 30 (ICR30)	Interrupt Control Register 31 (ICR31)
0x0060	Interrupt Control Register 32 (ICR32)	Interrupt Control Register 33 (ICR33)	Interrupt Control Register 34 (ICR34)	Interrupt Control Register 35 (ICR35)
0x0064	Interrupt Control Register 36 (ICR36)	Interrupt Control Register 37 (ICR37)	Interrupt Control Register 38 (ICR38)	Interrupt Control Register 39 (ICR39)
0x0068	Interrupt Control Register 40 (ICR40)	Interrupt Control Register 41 (ICR41)	Interrupt Control Register 42 (ICR42)	Interrupt Control Register 43 (ICR43)

Table 14-5. INTC 32-bit Memory Map (Continued)

0x006c	Interrupt Control Register 44 (ICR44)	Interrupt Control Register 45 (ICR45)	Interrupt Control Register 46 (ICR46)	Interrupt Control Register 47 (ICR47)
0x0070	Interrupt Control Register 48 (ICR48)	Interrupt Control Register 49 (ICR49)	Interrupt Control Register 50 (ICR50)	Interrupt Control Register 51 (ICR51)
0x0074	Interrupt Control Register 52 (ICR52)	Interrupt Control Register 53 (ICR53)	Interrupt Control Register 54 (ICR54)	Interrupt Control Register 55 (ICR55)
0x0078	Interrupt Control Register 56 (ICR56)	Interrupt Control Register 57 (ICR57)	Interrupt Control Register 58 (ICR58)	Interrupt Control Register 59 (ICR59)
0x007c	Interrupt Control Register 60 (ICR60)	Interrupt Control Register 61 (ICR61)	Interrupt Control Register 62 (ICR62)	Interrupt Control Register 63 (ICR63)
0x0080-0x00dc	Reserved			
0x00e0	Reserved			
0x00e4	Reserved			
0x00e8	Reserved			
0x00ec	IRQ Acknowledge {IRQIACK)	Reserved		
0x00f0	FIQ Acknowledge (FIQIACK)	Reserved		
0x00f4	Reserved			
0x00f8	Reserved			
0x00fc	Reserved			

The platform's IPS memory map reserves a contiguous 64 KByte space for the interrupt controller(s). Figure 14-5 shows the 256-byte memory map for a single interrupt controller. For designs with multiple controllers, see Figure 14-6.

Table 14-6. Multiple Interrupt Controller IPS Memory Map

Configuration	Offset for intc0[1]	Offset for intc1	Offset for intc2	IACK Addresses
1 controller	0x0000	—	—	0x00ec, 0x00f0
2 controllers	0x0000	+0x4000	—	0xc0ec, 0xc0f0
3 controllers	0x0000	+0x4000	+0x8000	0xc0ec, 0xc0f0

1. The offsets for intc1, intc2 and the IACK addresses are relative to the base of intc0.

14.8.1 Register Descriptions

14.8.1.1 IPR[63:0] - Interrupt Pending Register (IPRH, IPRL)

The IPRH and IPRL registers are each 32 bits in size, and provide a bit map for each interrupt request to indicate if there is an active request (1 = active request, 0 = no request) for the given source. *The state of*

the Interrupt Mask Register does not affect the IPR. The IPR is cleared by reset and updated each platform clock cycle. The IPR is a read-only register, so any attempted write to this register is terminated with an error.

Each bit of the IPR[n] is mapped to the corresponding input signal ipi_int[n], i.e., IPR[63:0] is mapped to ipi_int[63:0].

See Figure 14-2 and Table 14-7 for the IPR definition.

Register address: INTC_Offset + 0x00 (IPRH), + 0x04 (IPRL)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	IPR63	IPR62	IPR61	IPR60	IPR59	IPR58	IPR57	IPR56	IPR55	IPR54	IPR53	IPR52	IPR51	IPR50	IPR49	IPR48
W	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	IPR47	IPR46	IPR45	IPR44	IPR43	IPR42	IPR41	IPR40	IPR39	IPR38	IPR37	IPR36	IPR35	IPR34	IPR33	IPR32
W	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	IPR31	IPR30	IPR29	IPR28	IPR27	IPR26	IPR25	IPR24	IPR23	IPR22	IPR21	IPR20	IPR19	IPR18	IPR17	IPR16
W	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	IPR15	IPR14	IPR13	IPR12	IPR11	IPR10	IPR09	IPR08	IPR07	IPR06	IPR05	IPR04	IPR03	IPR02	IPR01	IPR00
W	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 14-2. Interrupt Pending (IPRH, IPRL) Registers

Table 14-7. Interrupt Pending (IPRH, IPRL) Field Descriptions

Name	Description
IPRn, n = 0,... 63	Interrupt Pending Register n 0 The interrupt request n is negated. 1 The interrupt request n is asserted.

14.8.1.2 IMR[63:0] - Interrupt Mask Register (IMRH, IMRL)

The IMRH and IMRL registers are each 32 bits in size, and provide a bit map for each interrupt to allow the request to be disabled or “masked” (1 = disable the request, 0 = enable the request). The IMR is set to all ones by reset, disabling all interrupt requests. The IMR can be read and written directly, or individual mask flags can be set or cleared by accesses through the SIMR (Set Interrupt Mask) or CIMR (Clear Interrupt Mask) registers.

Each bit of the IMR[n] is associated with the corresponding bit of the IPR[n].

See Figure 14-3 and Table 14-8 for the IPR definition.

Register address: INTC_Offset + 0x08 (IMRH), + 0x0c (IMRL)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	IMR63	IMR62	IMR61	IMR60	IMR59	IMR58	IMR57	IMR56	IMR55	IMR54	IMR53	IMR52	IMR51	IMR50	IMR49	IMR48
RESET:	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	IMR47	IMR46	IMR45	IMR44	IMR43	IMR42	IMR41	IMR40	IMR39	IMR38	IMR37	IMR36	IMR35	IMR34	IMR33	IMR32
RESET:	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	IMR31	IMR30	IMR29	IMR28	IMR27	IMR26	IMR25	IMR24	IMR23	IMR22	IMR21	IMR20	IMR19	IMR18	IMR17	IMR16
RESET:	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	IMR15	IMR14	IMR13	IMR12	IMR11	IMR10	IMR09	IMR08	IMR07	IMR06	IMR05	IMR04	IMR03	IMR02	IMR01	IMR00
RESET:	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

= Unimplemented or Reserved

Figure 14-3. Interrupt Mask (IMRH, IMRL) Registers

Table 14-8. Interrupt Mask (IMRH, IMRL) Field Descriptions

Name	Description
IMRn, n = 0,... 63	Interrupt Mask Register n 0 The interrupt request n is enabled. 1 The interrupt request n is disabled, i.e., masked.

14.8.1.3 INTFRC[63:0] - Force Interrupt Register (INTFRCH, INTFRCL)

The INTFRCH and INTFRCL registers are each 32 bits in size, and provide a mechanism to allow software generation of interrupts for each possible source for functional or debug purposes. The system design may reserve one or more sources to allow software to self-schedule interrupts by forcing one or more of these bits (1 = force request, 0 = negate request) in the appropriate INTFRC register. In some cases, the handling of an interrupt request may cause critical processing by the service routine along with the scheduling (using the INTFRC register) of a lower-priority interrupt request to be processed at a later

time for less-critical task handling. *The assertion of an interrupt request via the INTFRC register is not affected by the Interrupt Mask Register.* The INTFRC register is cleared by reset.

See Figure 14-4 and Table 14-9 for the INTFRC definition.

Register address: INTC_Offset + 0x10 (INTFRCH), + 0x14 (INTFRCL)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	IFRC63	IFRC62	IFRC61	IFRC60	IFRC59	IFRC58	IFRC57	IFRC56	IFRC55	IFRC54	IFRC53	IFRC52	IFRC51	IFRC50	IFRC49	IFRC48
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	IFRC47	IFRC46	IFRC45	IFRC44	IFRC43	IFRC42	IFRC41	IFRC40	IFRC39	IFRC38	IFRC37	IFRC36	IFRC35	IFRC34	IFRC33	IFRC32
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	IFRC31	IFRC30	IFRC29	IFRC28	IFRC27	IFRC26	IFRC25	IFRC24	IFRC23	IFRC22	IFRC21	IFRC20	IFRC19	IFRC18	IFRC17	IFRC16
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	IFRC15	IFRC14	IFRC13	IFRC12	IFRC11	IFRC10	IFRC09	IFRC08	IFRC07	IFRC06	IFRC05	IFRC04	IFRC03	IFRC02	IFRC01	IFRC00
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 14-4. Force Interrupt (INTFRCH, INTFRCL) Registers

Table 14-9. Force Interrupt (INTFRCH, INTFRCL) Field Descriptions

Name	Description Value
IFRCn, n = 0,... 63	Interrupt Force Register n 0 The forced interrupt request n is disabled. 1 A forced interrupt request n is enabled.

14.8.1.4 Interrupt Configuration (ICONFIG) Register

The ICONFIG register defines the operating configuration for the INTC module. See Figure 14-5 and Figure 14-10 for the ICONFIG definition.

Register address: INTC_Offset + 0x1b

Figure 14-5. Interrupt Configuration (ICONFIG) Register

Table 14-10. Interrupt Configuration (ICONFIG) Field Descriptions

Name	Description
7 EFIQPRI	Enable Core's Priority Elevation on FIQ If set, the assertion of an FIQ request to the core causes the processor's bus master priority to be temporarily elevated in the platform's crossbar switch arbitration logic. The processor's bus master arbitration priority remains elevated until the FIQ request is negated. If round-robin arbitration is enabled, this bit has no effect. If cleared, the assertion of an FIQ request does not affect the processor's bus master priority.
6 EIRQPRI	Enable Core's Priority Elevation on IRQ If set, the assertion of an IRQ request to the core causes the processor's bus master priority to be temporarily elevated in the platform's crossbar switch arbitration logic. The processor's bus master arbitration priority remains elevated until the IRQ request is negated. If round-robin arbitration is enabled, this bit has no effect. If cleared, the assertion of an IRQ request does not affect the processor's bus master priority.

Table 14-10. Interrupt Configuration (ICONFIG) Field Descriptions

Name	Description
5 EMASK	Enable Hardware Level Masking If set, the INTC automatically loads the level of an interrupt request into the CLMASK (current level mask) when an acknowledge is performed. At the exact same cycle, the value of the current interrupt level mask is saved in the SLMASK (saved level mask) register. This feature can be used to support software-managed nested interrupts. The value of SLMASK register should be read from the INTC and saved in the interrupt stack frame in memory, and restored near the service routine's exit. If cleared, the INTC does not perform any automatic masking of interrupt levels.
4–0 FIQDEF	FIQ Interrupt Level Definition This 5-bit field defines the mapping of the 16 interrupt levels into the FIQ output signal. The field is defined as: 0x00 = Levels 0 - 15 are mapped as FIQs 0x01 = Levels 1 - 15 are mapped as FIQs 0x02 = Levels 2 - 15 are mapped as FIQs 0x03 = Levels 3 - 15 are mapped as FIQs 0x04 = Levels 4 - 15 are mapped as FIQs 0x05 = Levels 5 - 15 are mapped as FIQs 0x06 = Levels 6 - 15 are mapped as FIQs 0x07 = Levels 7 - 15 are mapped as FIQs 0x08 = Levels 8 - 15 are mapped as FIQs 0x09 = Levels 9 - 15 are mapped as FIQs 0x0a = Levels 10 - 15 are mapped as FIQs 0x0b = Levels 11 - 15 are mapped as FIQs 0x0c = Levels 12 - 15 are mapped as FIQs 0x0d = Levels 13 - 15 are mapped as FIQs 0x0e = Levels 14 - 15 are mapped as FIQs 0x0f = Level 15 is mapped as FIQ 0x1- = No levels are mapped as FIQs; all are IRQs

14.8.1.5 Set Interrupt Mask (SIMR) Register

The SIMR register provides a simple memory-mapped mechanism to set a given bit in the IMR{H,L} registers to disable ("mask") a given interrupt request. The data value on a register write causes the corresponding bit in the IMR{H,L} register to be set. A data value greater than 63 provides a global set function, forcing the entire contents of IMR{H,L} to be asserted, masking all interrupts. Reads of this register return all zeroes.

This register is provided so interrupt service routines can easily mask the given interrupt request *without* the need to perform a read-modify-write sequence on the IMR{H,L}.

See Figure 14-6 and Figure 14-11 for the SIMR definition.

Register address: INTC_Offset + 0x1c

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W		SIMR[6:0]						
RESET:		0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 14-6. Set Interrupt Mask (SIMR) Register

Table 14-11. Set Interrupt Mask (SIMR) Field Descriptions

Name	Description
6–0 SIMR	Set Interrupt Mask 0-63 Set the corresponding bit in IMR{H,L}, masking the interrupt request 64-127 Set all bits in IMR{H,L}, masking all interrupt requests

14.8.1.6 Clear Interrupt Mask (CIMR) Register

The CIMR register provides a simple memory-mapped mechanism to clear a given bit in the IMR{H,L} registers to enable a given interrupt request. The data value on a register write causes the corresponding bit in the IMR{H,L} register to be cleared. A data value greater than 63 provides a global clear function, forcing the entire contents of IMR{H,L} to be cleared, enabling all interrupts. Reads of this register return all zeroes.

This register is provided so interrupt service routines can easily enable the given interrupt request *without* the need to perform a read-modify-write sequence on the IMR{H,L}.

In the event of a simultaneous write to both the CIMR and SIMR, the SIMR has priority and the resulting function would be a *set* of the interrupt mask register.

See Figure 14-7 and Figure 14-12 for the CIMR definition.

Register address: INTC_Offset + 0x1d

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W		CIMR[6:0]						
RESET:		0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 14-7. Clear Interrupt Mask (CIMR) Register

Table 14-12. Clear Interrupt Mask (CIMR) Field Descriptions

Name	Description
6–-0 CIMR	Clear Interrupt Mask 0-63 Clear the corresponding bit in IMR{H,L}, enabling the interrupt request 64-127 Clear all bits in IMR{H,L}, enabling all interrupt requests

14.8.1.7 Current Level Mask (CLMASK) Register

The Current Level Mask Register is provided so the INTC can optionally automatically manage masking of interrupt requests based on the programmed priority level. If enabled by ICONFIG[EMASK] being set, an interrupt acknowledge read cycle returns a vector number identifying the physical request source, and the CLMASK register is loaded with the level number associated with the request. Once the CLMASK register is updated, then all interrupt requests with level numbers equal to or less than this value are masked by the controller and are not allowed to cause the assertion of the interrupt signal to the processor core. As the CLMASK register is updated during the IACK cycle read, the former value is saved in the SLMASK register. Typically, once a level-*n* interrupt request is handled, the service routine restores the saved level mask value into the current level mask register to re-enable the lower priority requests. In addition, an interrupt service routine can explicitly load this register with a lower priority value to query for any pending interrupts via software interrupt acknowledge cycles. This topic is covered in more detail in **Section** Figure 14.10.3.

See Figure 14-8 and Figure 14-13 for the CLMASK definition.

Register address: INTC_Offset + 0x1e

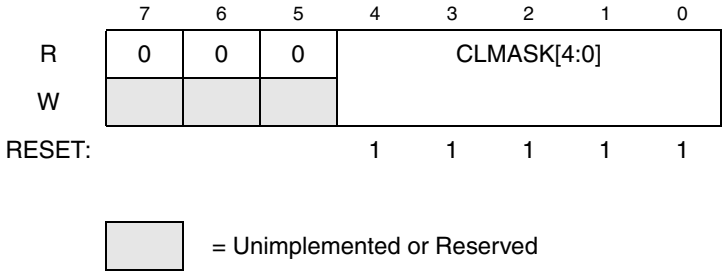


Figure 14-8. Current Level Mask (CLMASK) Register

Table 14-13. Current Level Mask (CLMASK) Field Descriptions

Name	Description
4–0 CLMASK	Current Level Mask This 5-bit field is treated as a signed integer within the range [-1, 0, 1,..., 15]. The value defines the level mask, where only interrupt levels greater than the current value are processed by the controller. if 0x1f, then level 00 - 15 requests are processed if 0x00, then level 01 - 15 requests are processed if 0x01, then level 02 - 15 requests are processed if 0x02, then level 03 - 15 requests are processed if 0x03, then level 04 - 15 requests are processed if 0x04, then level 05 - 15 requests are processed if 0x05, then level 06 - 15 requests are processed if 0x06, then level 07 - 15 requests are processed if 0x07, then level 08 - 15 requests are processed if 0x08, then level 09 - 15 requests are processed if 0x09, then level 10 - 15 requests are processed if 0x0a, then level 11 - 15 requests are processed if 0x0b, then level 12 - 15 requests are processed if 0x0c, then level 13 - 15 requests are processed if 0x0d, then level 14 - 15 requests are processed if 0x0e, then level 15 requests are processed if 0x0f - 0x1e, then all requests are masked

14.8.1.8 Saved Level Mask (SLMASK) Register

The Saved Level Mask Register is provided so the INTC can automatically manage masking of interrupt requests based on the programmed priority level. If enabled by ICONFIG[EMASK] being set, an interrupt acknowledge read cycle returns a vector number identifying the physical request source, and the CLMASK register is loaded with the level number associated with the request and the current contents of the CLMASK register is loaded into the SLMASK register. Typically, once a level-*n* interrupt request is handled, the service routine restores the saved level mask value into the current level mask register to re-enable the lower priority requests.

See Figure 14-9 and Figure 14-14 for the SLMASK definition.

Register address: INTC_Offset + 0x1f

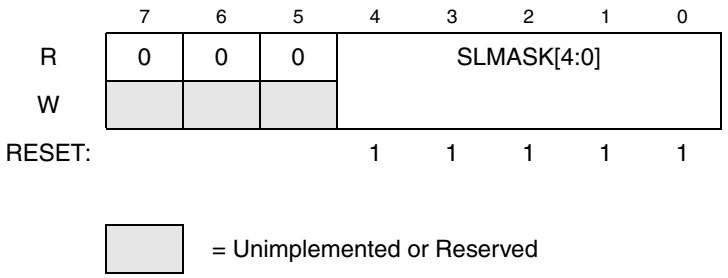


Figure 14-9. Saved Level Mask (SLMASK) Register

Table 14-14. Saved Level Mask (SLMASK) Field Descriptions

Name	Description
4–0 SLMASK	Saved Level Mask This 5-bit field is treated as a signed integer within the range [-1, 0, 1,..., 15]. The value defines the saved level mask. See the CLMASK field definition for more information on the specific values.

14.8.1.9 Interrupt Control Register n (ICRn), n = 0, 1, 2,..., 63

The ICRn registers contain the software-defined interrupt level for each interrupt request. Each ICRn contains a 4-bit interrupt level [0-15]. These registers are cleared by reset and should be programmed with the appropriate levels before interrupts are enabled.

When multiple interrupt requests are programmed to the same level number, they are processed in a descending request number order. As an example, if requests 63, 62, 2, 1 are programmed to a common level, request 63 is processed first, then request 62, then request 2 and finally request 1.

This definition allows software maximum flexibility in grouping interrupt request sources within any given priority level.

See Figure 14-10 and Figure 14-15 for the ICRn definition.

Register address: INTC_Offset + 0x40 + n

	7	6	5	4	3	2	1	0
R	0	0	0	0	Level[3:0]			
W								
RESET:					0	0	0	0

= Unimplemented or Reserved

Figure 14-10. Interrupt Control Register n (ICRn)

Table 14-15. Interrupt Control Register n (ICRn) Field Descriptions

Name	Description
3–0 Level	Interrupt Request Level This 4-bit field maps the given interrupt request to one of 16 levels, where 0x0 is the lowest priority level and 0xf is the highest priority level. If interrupt masking is enabled (ICONFIG[EMASK] = 1), the acknowledgment of a level-n request forces the controller to automatically mask all interrupt requests of level-n and lower.

14.8.1.10 IRQ Interrupt Acknowledge Register (IRQIACK)

The IRQIACK register is a read-only resource containing the vector number of the interrupt request currently being processed. It is typically read early in an IRQ interrupt service routine. There is a fixed association between the vector number returned in the IRQIACK register and the physical interrupt request input signal, namely:

```
vector_number =   64 + ipi_int[x] (for a single INTC implementation)
```

If there is no pending IRQ interrupt request when the read of the IRQIACK is performed, the interrupt controller returns a value of 63 signalling a spurious interrupt. This is also the reset value of the register. Any attempted write to this register generates an error termination.

See Figure 14-11 and Figure 14-16 for the IRQIACK definition.

Register address: INTC_Offset + 0xec

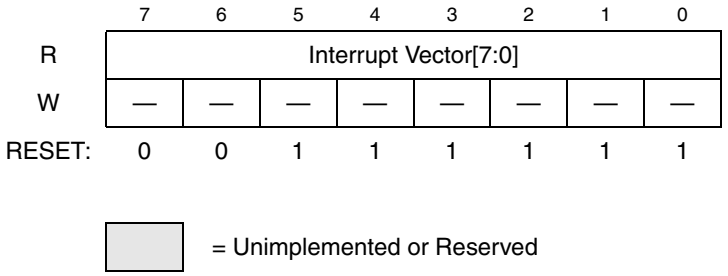


Figure 14-11. IRQ Interrupt Acknowledge Register (IRQIACK)

Table 14-16. IRQ Interrupt Acknowledge Register (IRQIACK) Field Descriptions

Name	Description
7–0 Interrupt Vector	Interrupt Vector Number This 8-bit field provides the vector number for the interrupt request currently being acknowledged. The vector number is derived from the physical interrupt request signal as: vector_number = 64 + ipi_int[x] If there is no pending IRQ request when the IRQIACK is read, a spurious interrupt vector number (63) is returned.

14.8.1.11 FIQ Interrupt Acknowledge Register (FIQIACK)

The FIQIACK register is a read-only resource containing the vector number of the interrupt request currently being processed. It is typically read early in an FIQ interrupt service routine. There is a fixed association between the vector number returned in the FIQIACK register and the physical interrupt request input signal, namely:

```
vector_number =   64 + ipi_int[x] (for a single INTC implementation)
```

If there is no pending FIQ interrupt request when the read of the FIQIACK is performed, the interrupt controller returns a value of 63 signalling a spurious interrupt. This is also the reset value of the register. Any attempted write to this register generates an error termination.

See Figure 14-12 and Figure 14-17 for the FIQIACK definition.

Register address: INTC_Offset + 0xf0

	7	6	5	4	3	2	1	0
R	Interrupt Vector[7:0]							
W	—	—	—	—	—	—	—	—
RESET:	0	0	1	1	1	1	1	1

= Unimplemented or Reserved

Figure 14-12. FIQ Interrupt Acknowledge Register (FIQIACK)

Table 14-17. FIQ Interrupt Acknowledge Register (FIQIACK) Field Descriptions

Name	Description
7–0 Interrupt Vector	Interrupt Vector Number This 8-bit field provides the vector number for the interrupt request currently being acknowledged. The vector number is derived from the physical interrupt request signal as: vector_number = 64 + ipi_int[x] If there is no pending FIQ request when the FIQIACK is read, a spurious interrupt vector number (63) is returned.

14.9 Functional Description

This section provides an overview of the functional operation of the INTC module.

14.9.1 Interrupt Controller Theory of Operation

To support the interrupt architecture of the ARM core programming model, the combined 64 interrupt sources are organized as 16 levels, with an arbitrary number of requests programmed to each level. Consider the priority structure within a single interrupt level (from highest to lowest priority):

```
level i:ipi_int[a]              programmed as level i (highest priority)
ipi_int[b]                      programmed as level i
ipi_int[c]                      programmed as level i
ipi_int[d]                      programmed as level i (lowest priority)
```

where the bit numbers [a,b,c,d] are defined such that a > b > c > d. In this example, 4 programmable interrupt sources are mapped into a single interrupt level.

The operation of the interrupt controller can be broadly partitioned into three activities:

- Recognition
- Prioritization
- Vector Generation during IACK

Recall the INTC is designed to provide a unique vector number for each interrupt request. This allows the operating system kernel to manage a vector table of addresses defining the starting location for each interrupt service routine. Throughout this discussion, it is assumed that the vector table contains 32-bit

addresses and uses the interrupt vector number as an index into this table so that execution in the appropriate service routine can begin as quickly as possible.

Refer to the INTC block diagram, Figure 14-1, for the subsequent discussion.

14.9.1.1 Interrupt Recognition

The interrupt controller continuously examines the request sources (the IPR register) and the interrupt mask register (IMR) to determine if there are active requests. This is the recognition phase. The Interrupt Force Register (INTFRC) also factors into the generation on an active request. An active request (assuming the hardware masking is enabled) is defined by the following boolean equation:

```
active_request[n]       = (IPR[n]    & ~IMR[n]
                        |  INTFRC[n])& (ICRn > CLMASK)
```

14.9.1.2 Interrupt Prioritization and Level Masking

As an active request is detected, it is first translated into the programmed interrupt level. Next, the appropriate level masking is performed, if this feature is enabled. Recall the level of the active request must be greater than the current mask level before it is signaled to the processor. The resulting 16-bit unmasked decoded priority level (intc_active_level[15:0]) is then driven out of the interrupt controller. The decoded priority levels from all the interrupt controllers are logically summed together and the highest priority interrupt level is then encoded into the 2-bit FIQ/IRQ signals that are sent to the processor core during this prioritization phase. The mapping of the interrupt levels into the 2-bit FIQ/IRQ signals (the level conversion) is controlled by the ICONFIG[FIQDEF] field.

14.9.1.3 Vector Generation during IACK

Once the core has sampled for pending interrupts and completed interrupt exception processing, it begins execution of the interrupt service routine (ISR) and typically generates a byte-sized operand read from the controller known as an interrupt acknowledge cycle. The type of interrupt request being acknowledged (FIQ or IRQ) determines the access address. The IACK transfer is a memory-mapped byte read via the AIPS controller of the FIQIACK or IRQIACK register,which, in the case of multiple interrupt controller instantiations, is routed into a separate interrupt arbiter module that determines the interrupt controller that gets the IACK transfer based upon which controller has the appropriate level pending. Interrupt arbitration priority is discussed in **Section** Figure 14.9.1.4. Next, the interrupt controller determines the highest unmasked level for the type of interrupt being acknowledged, and generates an 8-bit interrupt vector for that request to complete the cycle. The 8-bit interrupt vector is formed using the following algorithm:

```
For intc0,                          vector_number =   64 + ipi_int[x]
For optional intc1,                 vector_number = 128 + ipi_int[x]
For optional intc2,                 vector_number = 192 + ipi_int[x]
```

where the bit position [x] within the ipi_int[63:0] source directly determines the vector number. Vector numbers 0 - 63 are reserved for the processor and its internal exceptions. Thus, the following mapping of bit positions to vector numbers applies for the mandatory intc0:

```
if ipi_int[0] is active and acknowledged,          then vector_number =  64
if ipi_int[1] is active and acknowledged,          then vector_number =  65
if ipi_int[2] is active and acknowledged,          then vector_number =  66
...
if ipi_int[63] is active and acknowledged,         then vector_number = 127
```

The net effect is a fixed mapping between the bit position within the source requests to the actual interrupt vector number.

If there is no active unmasked interrupt source at the time of the IACK, a special “spurious interrupt” vector (vector_number = 63) is returned and it is the responsibility of the service routine to handle this error situation. For interrupting devices following the IPI Indigo protocol, this error condition should never be encountered.

Note this protocol implies the interrupting peripheral is **not** accessed during the acknowledge cycle since the interrupt controller completely services the acknowledge. ***This means the interrupt source must be explicitly disabled in the interrupt service routine.*** This design provides unique vector capability for all interrupt requests, regardless of the “complexity” of the peripheral device.

In most applications, it is expected that the hardware masking of interrupt levels by the INTC is enabled. In this mode of operation, the IACK read cycle also causes the current interrupt level mask to be saved in the SLMASK register, and the new level being acknowledged loaded into the CLMASK register. This operation then automatically masks the new level (and all lower levels) while in the service routine. Generally, as the service routine completes execution and the initiating request source has been negated, the saved mask level is restored into the current mask level to re-enable the lower priority levels.

Finally, the vector number returned during the IACK cycle provides the association with the request and the physical interrupt signal.

The CLMASK and SLMASK registers are all loaded (if properly enabled) during the interrupt acknowledge read cycle.

For more information on the specific operations typically performed in an interrupt service routine, see **Section** Figure 14.10.3.

It is anticipated that this interrupt controller can be modified to supply the vector number at the same time as the FIQ/IRQ signals are asserted to the core for future ARM microarchitectures supporting vectored interrupts, e.g., the ARM11 core.

14.9.1.4 Multiple Controller Requirements

For platform configurations with multiple interrupt controllers, there are additional guidelines which must be followed to insure correct operation. See Figure 14-6 for information about the mapping of the multiple controllers within the IPS memory region. For these configurations, there is a separate interrupt arbiter module that interfaces between the IPS bus and the multiple interrupt controllers. The interrupt arbiter, on the interrupt controller side, inputs the level pending and the read data from an IACK transfer and outputs a modified module enable that is based on ’global’ write and IACK transfer information. On the IPS bus side, the interrupt arbiter inputs the module enables, and special IACK module enable, and the IPS address bus and outputs the encoded active interrupt request value and the muxed IACK transfer data from all three

interrupt controllers. The interrupt arbiter is completely combinatorial and does not need to be configured, however the interrupt controller does have some requirements when more than one is present.

First, the Interrupt Configuration Register (ICONFIG) must be loaded with the exact same value in all instantiations of the interrupt controller. This is accomplished by writing to the ICONFIG register of intc0. This single write is then broadcast to all the controllers in the device via the interrupt arbiter.

Second, all reads and writes of the Current Level Mask (CLMASK) and Saved Level Mask (SLMASK) register must reference the mandatory intc0 controller. For multiple controller designs, the master copy of these two control registers is maintained in intc0.

Third, the interrupt controllers have a fixed priority, where intc0 is the highest, and intc2 is the lowest. The net effect is the combined interrupt levels across the multiple controllers are defined as (highest to lowest):

```
intc0:                  level 15
intc1:                  level 15
intc2:                  level 15
intc0:                  level 14
intc1:                  level 14
intc2:                  level 14
intc0:                  level 0
intc1:                  level 0
intc2:                  level 0
```

The resulting logic to steer an IACK read in a three-controller configuration using the intc*x*_active_level[15:0] output vector is defined in Figure 14-18:

Table 14-18. Global IACK Steering Algorithm[1] (3 Controllers)

intc0_active_level [15:0]	intc1_active_level [15:0]	intc2_active_level [15:0]	Level Acknowledged	IACK Steering
1---------------	----------------	----------------	Level 15	Route to intc0
0---------------	1---------------	----------------	Level 15	Route to intc1
0---------------	0---------------	1---------------	Level 15	Route to intc2
01--------------	0---------------	0---------------	Level 14	Route to intc0
00--------------	01--------------	0---------------	Level 14	Route to intc1
00--------------	00--------------	01--------------	Level 14	Route to intc2
...	...	...		
0000000000000001	0000000000000-	000000000000000-	Level 0	Route to intc0
0000000000000000	000000000000001	000000000000000-	Level 0	Route to intc1
0000000000000000	000000000000000	0000000000000001	Level 0	Route to intc2

1. “-” indicates a don’t care logic value.

14.9.2 Performance

There are two key performance parameters for the interrupt controller: latency from the assertion of an interrupt request, and vector generation timing.

As can be seen in the INTC block diagram (Figure 14-1), there are two levels of hardware registers between the assertion of an interrupt request on the ipi_int[63:0] input and the posting of the FIQ/IRQ request to the processor core. Thus, if an interrupt request is first asserted in cycle *i*, then the first assertion of a properly-enabled request to the processor core occurs in cycle *i+2*. The processor's sampling of the interrupt request signals is then dependent on the instruction stream being executed. For more information on the core behavior, see the appropriate ARM reference manual. As an example, see Section 2.9, page 69 of the *ARM7TDMI Technical Reference Manual, Rev. 4*.

As the IACK cycle is performed during the interrupt service routine, it appears as a normal IPS read cycle, and requires three platform cycles in the AMBA-AHB data phase (2 wait-states).

14.10 Initialization/Application Information

14.10.1 Initialization

The interrupt controller's reset state has all requests masked via the IMR. Before any interrupt requests are enabled, the following steps must be taken:

1. The ICONFIG register needs to be set to the desired system configuration.
2. All the ICRn registers need to be programmed with the appropriate interrupt levels.
3. The reset value for the Level Mask registers (CLMASK and SLMASK) are set to -1, the value with no levels masked. Typically, these registers do not need to be modified before interrupts are enabled.
4. The appropriate interrupt vector tables and interrupt service routines must be loaded into memory. Additionally, the memory address pointers for the FIQ and IRQ stacks loaded into the R13_{fiq,irq} registers.
5. Enable the interrupt requests by clearing the appropriate bits in the IMR and the CPSR[F,I] .

14.10.2 Typical Applications

In many real-time system designs, a typical configuration for supporting priority-based preemptive task scheduling requires only a *single* interrupt signal to the processor core. Stated differently, the two levels of interrupt support provided by an ARM core (FIQ/IRQ) are not necessarily required, and a single level is sufficient. For ARM cores with only a single interrupt level is implemented, the FIQ exception mode is typically the one to be used since it provides significantly more hardware resources for the interrupt processing than IRQ.

By setting ICONFIG[FIQDEF] = 0x00, *all* interrupt requests are mapped into the FIQ request signal, and the IRQ core functionality is completely unused.

Another common configuration may chose to define certain "non-maskable" interrupt requests. Typically, these requests are programmed as level 15, and would be logically connected to the FIQ core input. In this configuration, priority levels 0-14 would then be available for use of normal (IRQ) requests.

For applications where the wakeup functionality is used, the interrupt controller includes logic that limits the largest value of the interrupt mask level to "maximum – 1" so the controller can always generate an sleep mode exit. Thus, if **ipg_lp_int_mask** is set to the maximum value, the interrupt controller converts

this value to “maximum – 1" in the wakeup logic. This guarantees that a level 15 interrupt request generates the sleep mode exit.

14.10.3 Interrupt Service Routines

This section focuses on the interaction of the interrupt masking functionality with the service routine. Figure 14-13 presents a timing diagram showing various phases during the execution of an interrupt service routine with the controller level masking functionality enabled. It is important to note the time scale in this diagram is *not* meant to be accurate.

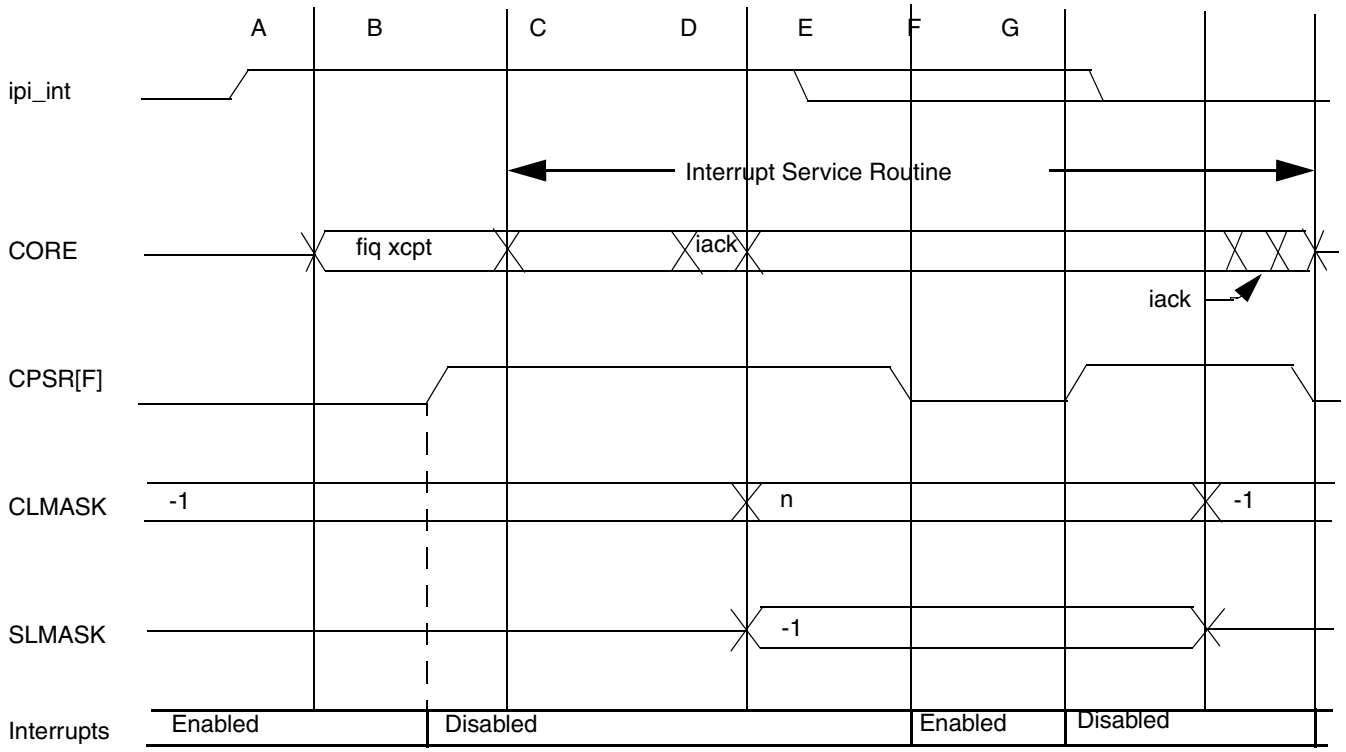


Figure 14-13. Interrupt Service Routine and Masking (Not To Scale)

Consider the events depicted in each “segment” [A-G] of this diagram.

In A, an interrupt request is asserted, which is mapped into the FIQ core signal.

As B begins, the interrupt request is recognized and the core begins FIQ exception processing. During the core’s exception processing, the CPSR sets the F bit, disabling all interrupts. At the end of the core’s exception processing, control passes to the interrupt service routine, shown as the beginning of segment C.

During C, the ISR performs the interrupt acknowledge read cycle to retrieve the vector number associated with the request. As the interrupt acknowledge read is performed, the vector number is returned to the core, and the CLMASK register raised to level n, the interrupt level being acknowledged. The former contents of the CLMASK is loaded into the SLMASK register at this time, at the end of C.

During segment D, the ISR accesses the peripheral to negate the interrupt request source. At the conclusion of segment D, the CPSR[F] flag is cleared to re-enable interrupts.

The bulk of the interrupt service routine executes in segment E, with interrupts enabled.

Near the end of the service routine, the CPSR[F] flag is again set, disabling interrupt requests, and preparing to perform the context switch.

At the end of segment F, the original value in the saved level mask (SLMASK) is restored to the current level mask (CLMASK). Optionally, the service routine can directly load the CLMASK register with any value with pending interrupt requests of certain levels need to be examined.

In segment G, the interrupt service routine completes execution. During this period of time (recall interrupts are disabled in the CPSR), it is possible to access the interrupt controller to see if there are any pending properly-enabled requests. Checking for any pending interrupt requests at this time provides the ability to initiate processing of another interrupt without the need to return from the original ISR and then incur the overhead of another interrupt exception.

At the conclusion of segment G, the processor core returns to the original interrupted task, or a different task that is ready to execute.

Chapter 15
MAC7200 Crossbar Switch (AXBS)

15.1 Introduction

This section provides information on the layout, configuration, and programming of the crossbar switch bus. The crossbar switch bus is a two master by five slave (2Mx5S) configuration where the core complex and the direct memory access controller are the two masters (2M) and the flash controller, FlexBus, RAM controller, external AHB and the peripheral controller are the five slaves (5S).

Figure 15-1 is a block diagram of the crossbar switch bus connections.

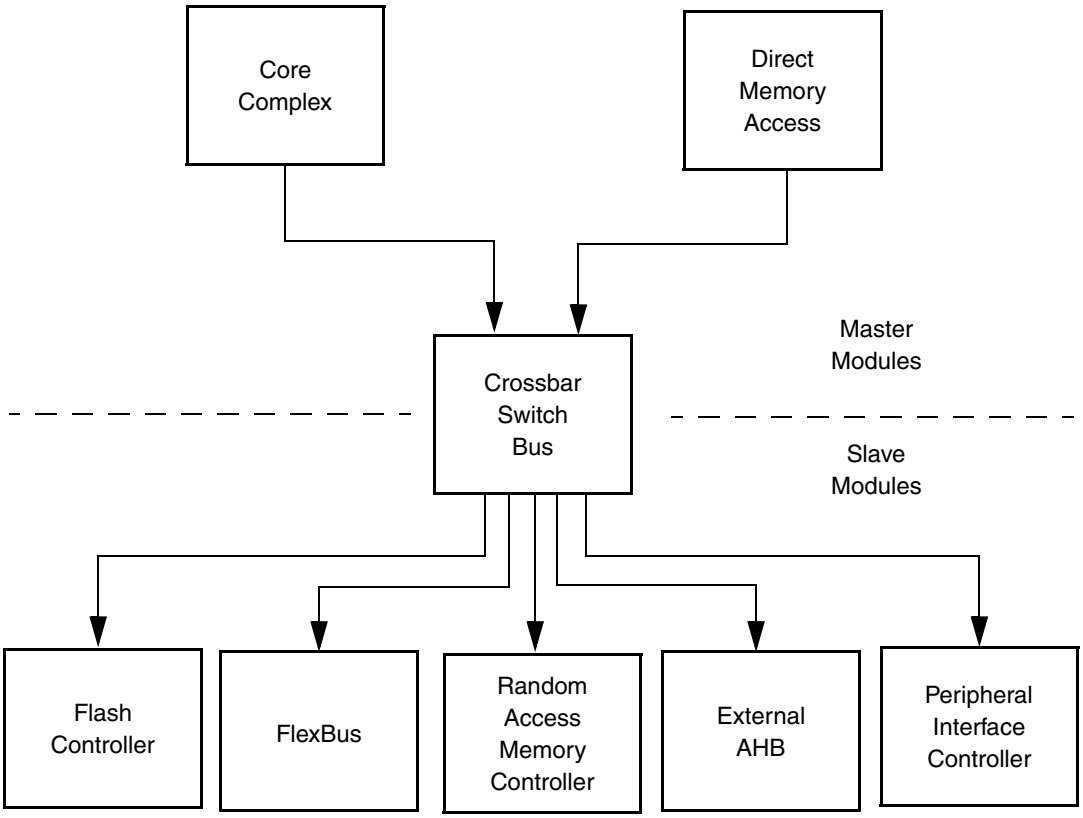


Figure 15-1. Crossbar Switch Bus Block Diagram

15.1.1 Overview

The crossbar switch bus connects the bus masters and bus slaves using a crossbar switch structure. This structure allows bus masters to simultaneously access different bus slaves with no interference while providing arbitration among the bus masters when they access the same slave. A variety of bus arbitrations methods and attributes may be programmed on a bus slave by bus slave basis.

15.1.2 Features

The crossbar switch bus includes these distinctive features:

- Symmetric crossbar bus switch implementation
  - — Allows concurrent accesses from different masters to different slaves
  - — Slave arbitration attributes configured on a slave by slave basis
- The crossbar switch bus is 32 bits wide and supports byte, half-word (2byte), word (4byte), and 16byte burst transfers
- 32-bit address bus width, 32-bit data bus width
- Crossbar switch bus operates at a 1-to-1 clock frequency with the bus masters
- Support for AHB v2.0 AHB-Lite bus protocol
- Up to 8x8 simultaneous Master => Slave connections
- Each Slave Port is individually configurable via a Peripheral Bus interface
- Memory Map: 32-bit peripheral with 256 bytes per slave port, word addressable only
- Each slave port has a 3-bit base address, to select a 512Mb address space within the 4Gb addressable space. All combinations of master/slave are valid. The crossbar switch can support simultaneous master transactions, as long as the slave being accessed is different.

Table 15-1. Crossbar Slave Port Addresses

Slave Port	Address Range
S0	$0000_0000 - $1FFF_FFFF
S1	$2000_0000 - $3FFF_FFFF
S2	$4000_0000 - $5FFF_FFFF
S3	$6000_0000 - $7FFF_FFFF
S4	$8000_0000 - $9FFF_FFFF
S5	$A000_0000 - $BFFF_FFFF
S6	$C000_0000 - $DFFF_FFFF
S7	$E000_0000 - $FFFF_FFFF

15.1.3 AXBS Integration

The AXBS is integrated with the following configuration:

- Alternate Master Priority (AMPR) register is not enabled

- Alternate Slave General Purpose (ASGPR) register is not enabled
- Master Port 0 (ARM7 core) is present with burst enabled
- Master Port 1 (DMA) is present with burst disabled
- Slave Port 0 (Flash) is present
- Slave Port 1 (FlexBus) is present
- Slave Port 3 (SRAM) is present
- Slave Port 5 (BAM) is present
- Slave Port 7 (AIPS) is present
- 32-bit datapath

Table 15-2. MAC72xx AXBS Master and Slave Ports

Master Port	Slave Port
M0: ARM7 Core	S0: Selectable
M1: eDMA	S1: Selectable
	S3: SRAM Controller
	S5: Boot Assist Module (BAM)
	S7: Peripheral Bus Bridge (AIPS)

15.1.4 Modes of Operation

The crossbar switch bus provides two arbitration modes, fixed or round-robin. The arbitration mode may be set on a slave by slave basis. For slaves configured for fixed arbitration mode, a unique arbitration level is assigned to each bus master.

For a given slave with fixed arbitration operation, the highest priority active master accessing that slave is granted the master bus switch path to that slave. A higher priority master will block access to a given slave from a lower priority master as long as the higher priority master continuously requests that slave.

For a given slave with round-robin arbitration, active masters accessing that slave are initially granted the slave based on their master port number. Master priority is then modified in a wrap-around manner to give all masters fair access to the slave.

15.2 External Signal Description

There are no AXBS signals that drive or are driven from MCU pins.

15.3 Memory Map Definition

There are two registers that reside in each slave port of the crossbar switch bus. Read and write transfers both require two IP bus clock cycles. The registers can only be read from and written to in supervisor mode. Additionally, these registers can only be read from or written to by 32-bit accesses. Non 32-bit accesses to legal registers are ignored.

The registers are fully decoded and a bus error response is returned if an unimplemented location is accessed within the crossbar switch bus.

The slave registers also feature a bit, which when written with a 1, will prevent the registers from being written to again. The registers will still be readable, but future write attempts will have no effect on the registers and will be terminated with an error response.

Table 15-3 is the memory map for the crossbar switch bus program-visible registers.

Table 15-3. Module Memory Map

Offset from crossbar_base	Register Name and Use	Access
0x000	Priority Register for Flash Controller Slave port (PR_FC)	word (4byte)
0x010	Control Register for Flash Controller Slave port (CR_FC)	word (4byte)
0x100	Priority Register for FlexBus Slave port (PR_FB)	word (4byte)
0x110	Control Register for FlexBus Slave port (CR_FB)	word (4byte)
0x300	Priority Register for RAM Controller Slave port (PR_RC)	word (4byte)
0x310	Control Register for RAM Controller Slave port (CR_RC)	word (4byte)
0x500	Priority Register for external AHB Slave port (PR_AHB)	word (4byte)
0x510	Control Register for external AHB Slave port (CR_AHB)	word (4byte)
0x700	Priority Register for Peripheral Controller Slave port (PR_PC)	word (4byte)
0x710	Control Register for Peripheral Controller Slave port (CR_PC)	word (4byte)

15.4 Register Descriptions

This section consists of register descriptions for the crossbar switch bus.

15.4.1 Priority Register

The Priority Register (PR) sets the priority of each master port on a per slave port basis and resides in each slave port.

Table 15-4. Priority Register Summary

PR_FC PR_FB PR_RC PR_AHB PR_PC	Priority Register for Slave Port to Flash Controller Priority Register for Slave Port to FlexBus Priority Register for Slave Port to RAM Conttroller Priority Register for Slave Port to external AHB Priority Register for Slave Port to Peripheral Controller											crossbar_base + 0x000 crossbar_base + 0x100 crossbar_base + 0x300 crossbar_base + 0x500 crossbar_base + 0x700				
Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Name	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	0

Table 15-4. Priority Register Summary (Continued)

Name	Reserved											DMA				Core
Type	r	r	r	r	r	r	r	r	r	r	r	rw	r	r	r	rw
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

Table 15-5. Priority Register Descriptions

Field	Description
31–5	Priority Register Reserved. These bits are reserved for future expansion. They read as zeros and should be written with zeros for upward compatibility.
4 DMA_ MSTR	DMA Master Priority. This bit sets the arbitration priority for this port on the associated slave port. This bit is initialized by hardware reset. The reset value is 1. 0 This master has the highest priority when accessing the slave port. 1 This master has the lowest priority when accessing the slave port.
3–1	Priority Register Reserved.These bits are reserved for future expansion. They read as zeros and should be written with zeros for upward compatibility.
0 CORE_ MSTR	Core Master Priority. This bit sets the arbitration priority for this port on the associated slave port. This bit is initialized by hardware reset. The reset value is 0 0 This master has the highest priority when accessing the slave port. 1 This master has the lowest priority when accessing the slave port.

The Priority Register can only be accessed with 32-bit accesses. Once the RO (Read Only) bit has been set in the slave Control Register the Priority Register can only be read, attempts to write to it will have no effect on the PR and result in a bus error response.

Additionally, no two available master ports may be programmed with the same priority level. Attempts to program two or more available masters with the same priority level will result in an error response and the PR will not be updated.

15.4.2 Control Register

The Control Register (CR) controls several features of each slave port.

The Read Only (RO) bit will prevent any registers associated with this slave port from being written to once set. This bit may be written with 0 as many times as the user desires, but once it is written to a 1 only a reset condition will allow it to be written again.

The Halt Low Priority (HLP) bit will set the priority of a request to enter low power mode to the lowest possible priority for initial arbitration of the slave ports. By default it is the highest priority. Please note, setting this bit will not effect the request for low power mode from attaining highest priority once it has control of the slave ports.

The Arbitration Mde (ARB) bits control the arbitration mode. The arbitration mode may be fixed or round-robin.

The Parking Control (PCTL) bits determine how the slave port will park when no master is actively making a request. The available options are to park on the master defined by the PARK bits, park on the last master to use the slave port, or go into a low power park mode which will force all the outputs of the

slave port to inactive states when no master is requesting an access. The low power park feature can result in an overall power savings if the slave port is not saturated; however, it will force an extra clock of latency whenever any master tries to access it when it is not in use because it will not be parked on any master.

The PARK bits determine which master the slave will park on when no master is making an active request and there is no request for low power mode. Please use caution to only select master ports that are actually present in the design.

Table 15-6. Control Register Summary

CR_FC CR_FB CR_RC CR_AHB CR_PC		Control Register for Slave Port to Flash Controller Control Register for Slave Port to FlexBus Control Register for Slave Port to RAM Conttroller Control Register for Slave Port to external AHB Control Register for Slave Port to Peripheral Controller										crossbar_base + 0x010 crossbar_base + 0x110 crossbar_base + 0x310 crossbar_base + 0x510 crossbar_base + 0x710					
Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Name	RO	HLP														
Type	rw[1]	rw	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	0
Name							ARB				PCTL					Park
Type	r	r	r	r	r	r	rw	rw	r	r	rw	rw	r	r	r	rw
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1. Once this bit is written to a 1 only hardware reset will return it to a 0.

Table 15-7. Control Register Descriptions

Name	Description
31 RO	Read Only. This bit is used to force all of a slave port's registers to be read only. Once written to 1 it can only be cleared by hardware reset. This bit is initialized by hardware reset. The reset value is 0. 0 All this slave port's registers can be written. 1 All this slave port's registers are read only and cannot be written (attempted writes have no effect and result in an error response).
30 HLP	Halt Low Priority. This bit is used to set the initial arbitration priority for low power mode requests. This bit is initialized by hardware reset. The reset value is 0 0 The low power mode request has the highest priority for arbitration on this slave port. 1 The low power mode request has the lowest initial priority for arbitration on this slave port.
29–10	Control Register Reserved. These bits are reserved for future expansion. They read as zero and should be written with zero for upward compatibility.
9–8 ARB	Arbitration Mode. These bits are used to select the arbitration policy for the slave port. These bits are initialized by hardware reset. The reset value is 00. 00 Fixed Priority. 01 Round Robin (rotating) Priority. 10 Reserved 11 Reserved

Table 15-7. Control Register Descriptions (Continued)

Name	Description
7–6	Control Register Reserved. These bits are reserved for future expansion. They are read as zero and should be written with zero for upward compatibility.
5–4 PCTL	Parking Control. These bits determine the parking control used by this slave port. These bits are initialized by hardware reset. The reset value is 00. 00 When no master is making a request the arbiter will park the slave port on the master port defined by the PARK bit field. 01 When no master is making a request the arbiter will park the slave port on the last master to be in control of the slave port. 10 When no master is making a request the arbiter will park the slave port on no master and will drive all outputs to a constant safe state. 11 Reserved
3–1	Slave General Purpose Control Register Reserved. This bit is reserved for future expansion. It is read as zero and should be written with zero for upward compatibility.
0 PARK	PARK - This bit is used to determine which master port this slave port parks on when no masters are actively making requests and the PCTL bits are set to 00. This bit is initialized by hardware reset. The reset value is 0. 0 Park on CORE Master Port. 1 Park on DMA Master Port.

The CR can only be accessed with 32-bit accesses. Once the RO (Read Only) bit has been set in the CR the CR can only be read, attempts to write to it will have no effect on the CR and result in an error response.

15.5 Functional Description

This section describes in more detail the functionality of the crossbar switch bus.

15.5.1 Arbitration

The crossbar switch bus supports two arbitration schemes; a simple fixed-priority comparison algorithm, and a simple round-robin fairness algorithm. The arbitration scheme is independently programmable for each slave port.

15.5.1.1 Fixed Priority Operation

When operating in fixed-priority mode, each master is assigned a unique priority level in the PR (Priority Register). If two masters both request access to a slave port the master with the highest priority in the selected priority register will gain control over the slave port.

Any time a master makes a request to a slave port the slave port checks to see if the new requesting master's priority level is higher than that of the master that currently has control over the slave port (unless the slave port is in a parked state). The slave port does an arbitration check at every bus transfer boundary to ensure that the proper master (if any) has control of the slave port.

If the new requesting master's priority level is higher than that of the master that currently has control of the slave port the new requesting master will be granted control over the slave port at the next clock edge. The exception to this rule is if the master that currently has control over the slave port is running a fixed

length burst transfer or a locked transfer. In this case the new requesting master will have to wait until the end of the burst transfer or locked transfer before it will be granted control of the slave port.

If the new requesting master's priority level is lower than that of the master that currently has control of the slave port the new requesting master will be forced to wait until the master that currently has control of the slave port either runs an IDLE cycle or runs a non IDLE cycle to a location other than the current slave port.

15.5.1.2 Round-Robin Priority Operation

When operating in round-robin mode, each master is assigned a relative priority based on the master port number. This priority is based on how far ahead the master port number of the requesting master is to the master port number of the current bus master for this slave. Master port numbers are compared modulo the total number of bus masters, i. e. take the requesting master port number minus the current bus master's port number modulo the total number of bus masters. The master port whose priority is the highest based on this comparison will be granted control over the slave port at the next bus transfer boundary.

For the case of only the two bus masters on this device, this means when operating in round-robin mode, if the core complex is the current bus master for a given slave, than the DMA has the highest priority when requesting that slave. Likewise, if the DMA is the current bus master for a given slave, then the core complex has the highest priority when requesting that slave.

Once granted access to a slave port, a master may perform as many transfers as desired to that port until another master makes a request to the same slave port. The next master in line will be granted access to the slave port at the next transfer boundary.

Parking may still be used in a round-robin mode, but will not affect the round-robin pointer unless the parked master actually performs a transfer. Handoff will occur to the next master in line after one cycle of arbitration. If the slave port is put into low power park mode the round-robin pointer will be reset to point at master port 0, giving it the highest priority.

15.5.2 Priority Assignment

Each master port needs to be assigned a unique 1 bit priority level. If an attempt is made to program multiple master ports with the same priority level within the priority register (PR) the crossbar switch bus will respond with a bus error and the registers will not be updated.

15.6 Initialization/Application Information

No initialization is required by or for the crossbar switch bus. Hardware reset ensures all the register bits used by the crossbar switch bus are properly initialized.

15.7 AXBS Bus Aborts

15.7.1 IPI Register Interface

The AXBS module supports Peripheral Bus bus aborts, and enforces the following memory map:

Table 15-8. AXBS Bus Aborts

Abort	Allowed
	$0000-$0007
$0008-$000f	
	$0010-$0017
$0018-$00ff	
	$0100-$0107
$0108-$010f	
	$0110-$0117
$0118-$01ff	
	$0200-$0207
$0208-$020f	
	$0210-$0217
$0218-$02ff	
	$0300-$0307
$0308-$030f	
	$0310-$0317
$0318-$03ff	
	$0400-$0407
$0408-$040f	
	$0410-$0417
$0418-$04ff	
	$0500-$0507
$0508-$050f	
	$0510-$0517
$0518-$05ff	
	$0600-$0607
$0608-$060f	
	$0610-$0617
$0618-$06ff	
	$0700-$0707
$0708-$070f	
	$0710-$0717
$0718-$07ff	
	$0800-$0804
$0805-$08ff	
	$0900-$0904
$0905-$09ff	
	$0a00-$0a04
$0a05-$0aff	
	$0b00-$0b04
$0b05-$0bff	
	$0c00-$0c04
$0c05-$0cff	
	$0d00-$0d04
$0d05-$0dff	
	$0e00-$0e04
$0e05-$0eff	

Table 15-8. AXBS Bus Aborts (Continued)

Abort	Allowed
	$0f00-$0f04
$0f05-$3fff	

If any part of a read or write falls within an aborted region, the entire transfer is aborted. Since all AXBS registers are 32-bit, you can not have the situation where an access is partially in an aborted region.

Supervisor Access: All accesses to the AXBS registers must be in Supervisor Mode. All User Mode accesses are aborted.

NOTE

If the RO bit in a slave port's Slave General Purpose Control Register is set, then any write access to that slave's registers will be aborted.

15.7.2 Master/Slave Interface

In addition to the IPI interface, the AXBS can abort transfers through the crossbar itself. The following transfers will be aborted:

- Transfers to an unused slave port, as follows:
  - S2: $4000_0000 - $5FFF_FFFF
  - S4: $8000_0000 - $9FFF_FFFF
  - S6: $C000_0000 - $DFFF_FFFF
- Writing to a slave address when the RO bit in the slave's SGPCR register is set (1).

15.8 AXBS Differences from MAC71xx

- Addition of Slave 5 port (BAM). Note that the BAM code is accessible by the application software.
- Replaced tightly coupled Flash bus with Slave 0 port. This means that the DMA may also access the Flash through the same bus as the ARM7 core.

Chapter 16
AHB to IPI Bridge (AIPS)

16.1 Introduction

The AIPS provides the interface between the 32-bit system bus (AHB-Lite 2.v6) and the 32-bit peripheral bus. The AIPS can support up to 32 peripherals, and implements various levels of protection for each peripheral, including a trusted master scheme for both the ARM7 core (MASTER0) and the eDMA (MASTER1).

Not all peripherals are present on all devices. It is the responsibility of the application to ensure that no accesses to reserved address ranges are performed.

16.1.1 Features

The following list summarizes the key features of the AIPS:

- Support for AHB v2.0 AHB-Lite bus protocol
- 32-bit address bus width
- 32-bit data bus width
- 2 clocks for READs, 3 clocks for WRITEs
- 32 on-platform peripheral module enables
- 32 off-platform peripheral module enables (+ one bit-wise OR signal)
- 2 global off-platform address space module enables
- Supervisor/User/Test access support
- Software enabled write buffering (posted write)
- Memory Map: 64MByte (ADDR[25:0]), byte addressable
  - 512KByte: Platform Peripherals #0-#31 (32 modules x 16KBytes each)
    - $0000_0000 - $0007_FFFF
  - 512KByte: MAC72xx Peripherals (32 peripherals x 16KBytes each)
    - $0008_0000 - $00F_FFFF
  - 63MBytes: Global External Address Space
    - $0010_0000 - $03FF_FFFF
- The AIPS is not relocatable under any circumstances

16.1.2 General Operation

The AIPS is the interface between the AHB-Lite 2.v6 interface and on-chip IPS v3.0 peripherals as shown in Figure 16-1.

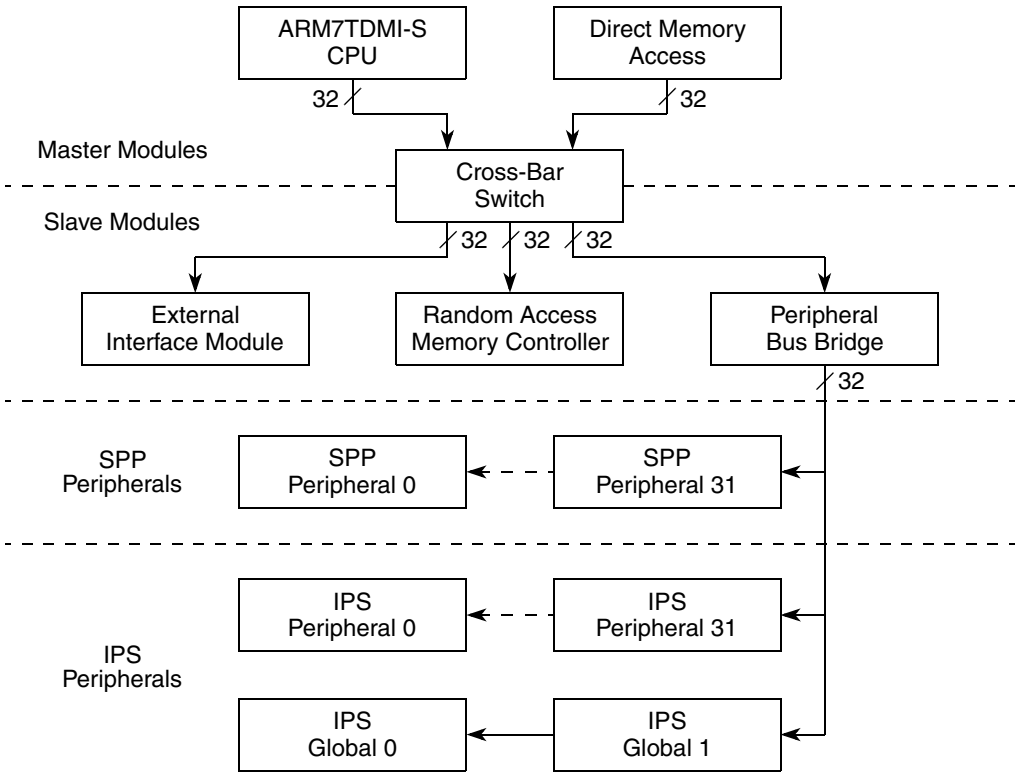


Figure 16-1. AIPS Interface Block Diagram

IPS peripherals are modules that contain readable/writable control and status registers. The AHB master reads and writes these registers through the AIPS. The AIPS generates module enables, the module address, transfer attributes, byte enables and write data as inputs to the IPS peripherals. The AIPS captures read data from the IPS interface and drives it on the AHB.

The AIPS occupies a 64MByte portion of the address space. A 0.5MByte portion of this space is allocated to on-platform peripherals. The remaining 63.5MBytes are available for off-platform devices. The register maps of the IPS peripherals are located on 16Kbyte boundaries. Each IPS peripheral is allocated one 16Kbyte block of the memory map, and is activated by one of the module enables from the AIPS. Up to thirty-two 16Kbyte external IPS peripherals may be implemented, occupying contiguous blocks of 16Kbytes. Two global external IPS module enables are available for the remaining 63Mbytes of address space to allow for customization and expansion of addressed peripheral devices. In addition, a single “non-global” module enable is also asserted whenever any of the thirty-two non-global module enables is asserted.

The AIPS memory map is shown in Figure 16-2.

The connection of a particular module enable to a peripheral, and hence the exact address assignment for an IPS peripheral is system dependent, and is defined in the system specification. Each IPS peripheral selects its internal registers based on the address driven.

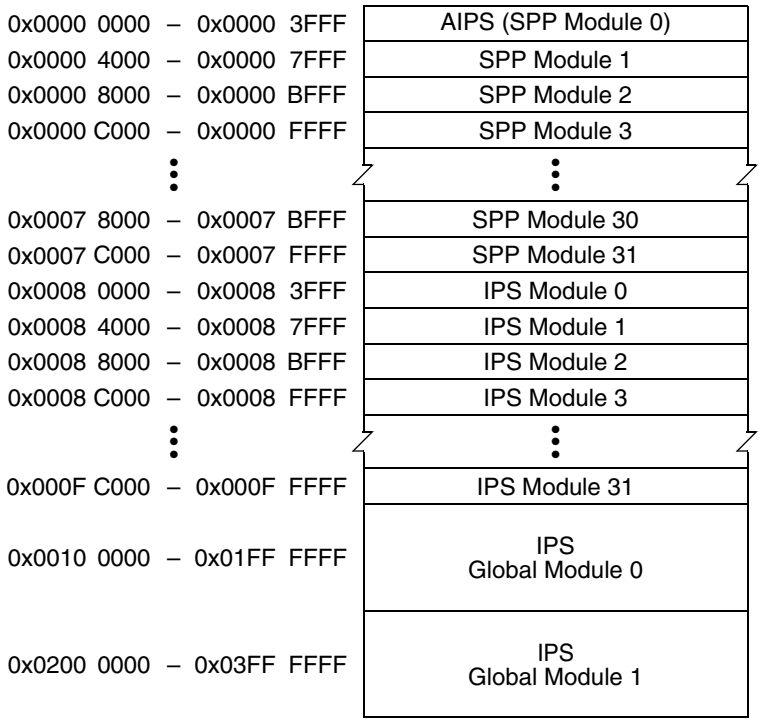


Figure 16-2. AIPS Memory Map

The AIPS is responsible for indicating to IPS peripherals if an access is in supervisor or user mode. The AIPS may block user mode accesses to certain IPS peripherals or it may allow the individual IPS peripherals to determine if user mode accesses are allowed. In addition, peripherals may be designated as write-protected. The AIPS supports the notion of “trusted” masters for security purposes. Masters may be individually designated as trusted for reads, trusted for writes, or trusted for both reads and writes, as well as being forced to look as though all accesses from a master are in user-mode privilege level. Refer to Section 16.4.2, “Control Registers,” for more information.

All peripheral devices require aligned accesses equal to or smaller in size than the peripheral size. An exception to this rule is supported for 32-bit peripherals to allow memory to be placed on the IPS.

16.2 AIPS Protocol

16.2.1 8/16/32--bit accesses

The following table defines the byte ordering scheme on the IPI bus in the MAC72xx.

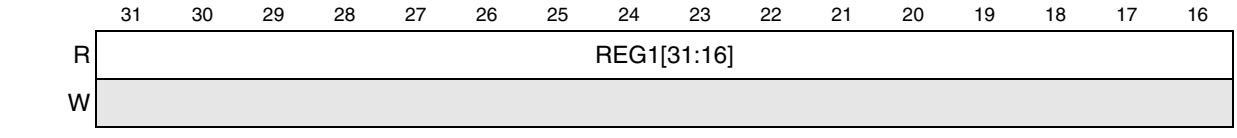
Table 16-1. AIPS 32-bit byte lanes

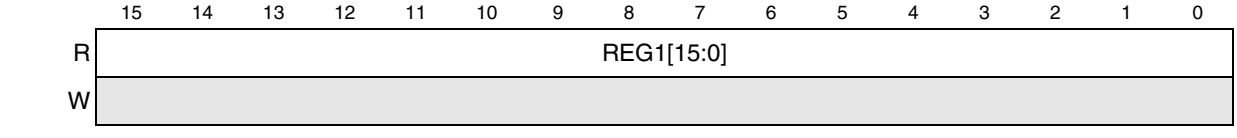
	ips_addr[1:0]	data[31:24]	data[23:16]	data[15:8]	data[7:0]
Byte	00	X			
	01		X		
	10			X	
	11				X
Halfword	00	X	X		
	01	ERR	ERR	ERR	ERR
	10			X	X
	11	ERR	ERR	ERR	ERR
Word	00	X	X	X	X
	01	ERR	ERR	ERR	ERR
	10	ERR	ERR	ERR	ERR
	11	ERR	ERR	ERR	ERR

Essentially, this is a “big endian like” bus, in the sense that high-order bits are stored at the lowest address.

32-bit register

Implement one 32-bit register **REG1** with base address $00.





= Unimplemented or Reserved

Register address: Base + $00

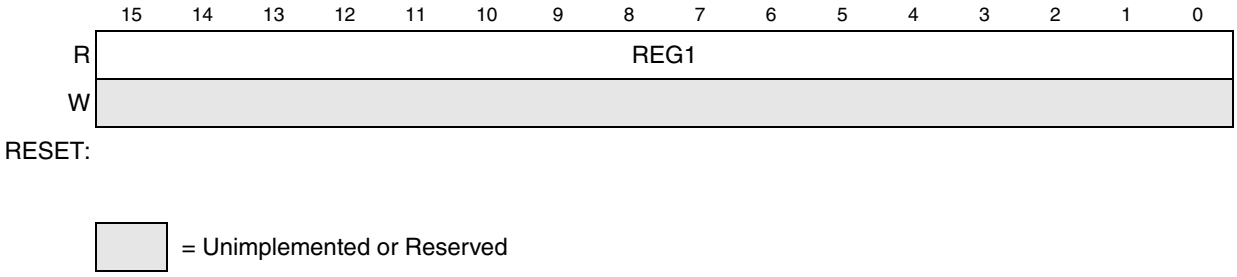
Figure xx: Register #1 Description (REG1)

Performing a READ operation will return the following data:

	Address	Data
Byte	$00	REG1[31:24]
	$01	REG1[23:16]
	$02	REG1[15:8]
	$03	REG1[7:0]
Halfword	$00	REG1[31:16]
	$02	REG1[15:0]
Word	$00	REG1[31:0]

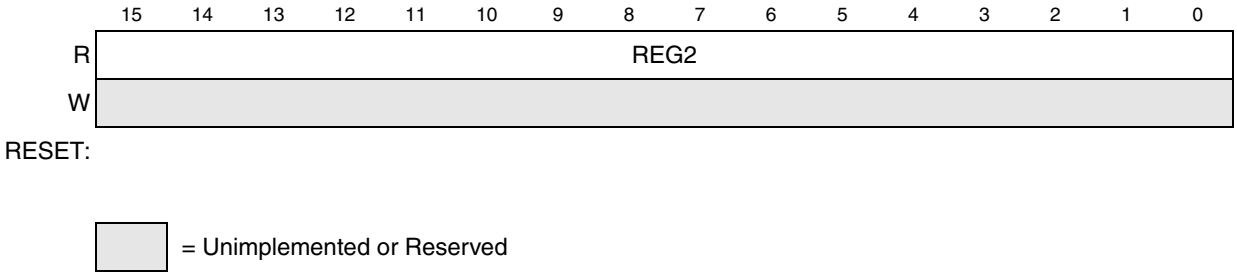
Two 16-bit registers

Implement two 16-bit registers **REG1** and **REG2**, with **REG1** having a base address of $00, and **REG2** having a base address of $02.



Register address: $00

Figure xx: Register #1 Description (REG1)



Register address: $02

Figure xx: Register #2 Description (REG2)

Performing a READ operation will return the following data:

Table 1:

	Address	Data
Byte	$00	REG1[15:8]
	$01	REG1[7:0]
	$02	REG2[15:8]
	$03	REG2[7:0]
Halfword	$00	REG1[15:0]
	$02	REG2[15:0]
Word	$00	{REG1[15:0] , REG2[15:0] }

Four 8-bit registers

Implement four 8-bit registers **REG1**, **REG2**, **REG3** and **REG4**, with addresses of $00, $01, $02 and $03, respectively.

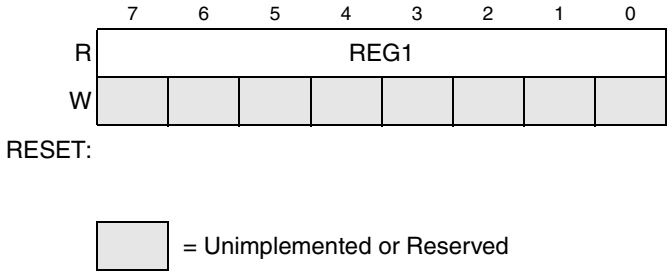


Figure xx: Register #1 Description (REG1)

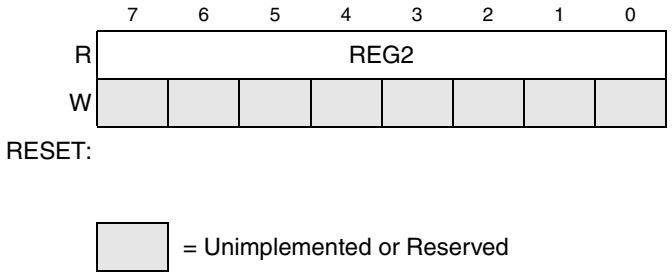
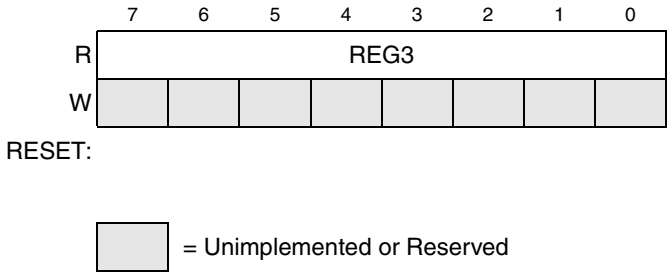
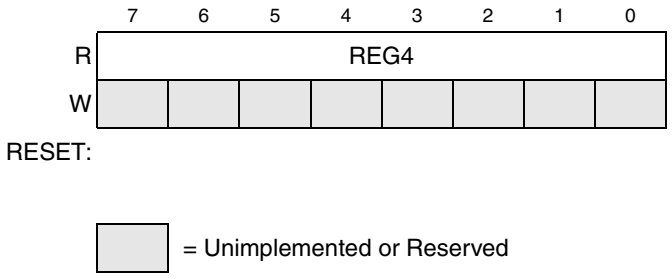


Figure xx: Register #2 Description (REG2)



Register address: $02

Figure xx: Register #3 Description (REG3)



Register address: $03

Figure xx: Register #4 Description (REG4)

Performing a READ operation will return the following data:

	Address	Data
Byte	$00	REG1[7:0]
	$01	REG2[7:0]
	$02	REG3[7:0]
	$03	REG4[7:0]
Halfword	$00	{REG1[7:0] , REG2[7:0]}
	$02	{REG3[7:0] , REG4[7:0]}
Word	$00	{REG1[7:0] , REG2[7:0] , REG3[7:0] , REG4[7:0]}

16.3 External Signal Description

The AIPS has no external pins.

16.4 Memory Map/Register Definition

This section provides information on AIPS registers.

16.4.1 Overview

There are eleven registers that control the AIPS. All registers are 32-bit registers and can only be accessed in supervisor mode by trusted bus masters. Additionally, these registers must only be read from or written to by a 32-bit aligned access. AIPS registers are mapped into the PACR0 address space.

Two system clocks are required for read accesses and three system clocks are required for write accesses to the AIPS registers.

16.4.2 Control Registers

The memory map for the AIPS program-visible registers is shown in Table 16-2. The MPROT fields of the MPR and the PACR and OPACR registers are 4 bits in width.

Table 16-2. AIPS Register Memory Map

AIPS Offset	[31:28]	[27:24]	[23:20]	[19:16]	[15:12]	[11:8]	[7:4]	[3:0]
0x0000	MPROT0	MPROT1	RFU	RFU	RFU	RFU	RFU	RFU
0x0004	RFU	RFU	RFU	RFU	RFU	RFU	RFU	RFU
0x0020	PACR0	PACR1	PACR2	RFU	RFU	RFU	RFU	RFU
0x0024	RFU	RFU	RFU	RFU	RFU	RFU	RFU	RFU
0x0028	PACR16	PACR17	PACR18	RFU	RFU	RFU	RFU	RFU
0x002c	RFU	RFU	RFU	RFU	RFU	RFU	RFU	RFU
0x0040	OPACR0	OPACR1	OPACR2	OPACR3	RFU	OPACR5	OPACR6	RFU
0x0044	RFU	RFU	RFU	OPACR11	RFU	OPACR13	OPACR14	OPACR15
0x0048	RFU	OPACR17	OPACR18	RFU	RFU	RFU	RFU	OPACR23
0x004c	OPACR24	RFU	OPACR26	RFU	OPACR28	RFU	RFU	RFU
0x0050	Reserved							

For the MAC7200, the processor complex is master zero and the enhanced direct memory access module is master one. The mapping between access control registers and peripheral modules is shown in the following table. Note that not all peripherals can be accessed in all MAC7200 operational modes.

Table 16-3. MAC7200 Peripheral to Access Control Register Map

Address Range	Module	Access Control Register
0xFC00_0000 – 0xFC00_3FFF	AIPS – AMBA to IP Bus Bridge	PACR0
0xFC00_4000 – 0xFC00_7FFF	AXBS – AMBA Crossbar Switch	PACR1
0xFC00_8000 – 0xFC00_BFFF	FlexBus – External Interface Module	PACR2
0xFC00_C000 – 0xFC03_FFFF	RFU	
0xFC04_0000 – 0xFC04_3FFF	MCM – Miscellaneous Control Module	PACR16
0xFC04_4000 – 0xFC04_7FFF	eDMA – Enhanced Direct Memory Access Controller	PACR17
0xFC04_8000 – 0xFC04_BFFF	INTC – Interrupt Controller	PACR18
0xFC04_C000 – 0xFC07_FFFF	RFU	
0xFC08_0000 – 0xFC08_3FFF	SSM – System Service Module	OPACR0
0xFC08_4000 – 0xFC08_7FFF	DMA Mux – Direct Memory Access Controller Mux	OPACR1

Table 16-3. MAC7200 Peripheral to Access Control Register Map (Continued)

Address Range	Module	Access Control Register
0xFC08_8000 – 0xFC08_BFFF	CRG – Clock and Reset Generator	OPACR2
0xFC08_C000 – 0xFC08_FFFF	PIT – Programmable Interval Timer	OPACR3
0xFC09_0000 – 0xFC09_3FFF	RFU	
0xFC09_4000 – 0xFC09_7FFF	FlexCAN_A – CAN Controller 0	OPACR5
0xFC09_8000 – 0xFC09_BFFF	FlexCAN_B – CAN Controller 1	OPACR6
0xFC09_C000 – 0xFC0A_BFFF	RFU	
0xFC0A_C000 – 0xFC0A_FFFF	IIC – Inter-IC bus	OPACR11
0xFC0B_0000 – 0xFC0B_3FFF	RFU	
0xFC0B_4000 – 0xFC0B_7FFF	DSPI_A – Serial Peripheral Interface 0	OPACR13
0xFC0B_8000 – 0xFC0B_BFFF	DSPI_B – Serial Peripheral Interface 1	OPACR14
0xFC0B_C000 – 0xFC0B_FFFF	DSPI_C – Serial Peripheral Interface 2	OPACR15
0xFC0C_0000 – 0xFC0C_3FFF	RFU	
0xFC0C_4000 – 0xFC0C_7FFF	ESCI_A – Enhanced Serial Communication Interface 0	OPACR17
0xFC0C_8000 – 0xFC0C_BFFF	ESCI_B – Enhanced Serial Communication Interface 1	OPACR18
0xFC0C_C000 – 0xFC0D_BFFF	RFU	
0xFC0D_C000 – 0xFC0D_FFFF	eMIOS – Enhanced Modular I/O Subsystem	OPACR23
0xFC0E_0000 – 0xFC0E_3FFF	ATD – Analog to Digital Converter	OPACR24
0xFC0E_4000 – 0xFC0E_7FFF	RFU	
0xFC0E_8000 – 0xFC0E_BFFF	PIM – Port Integration Module	OPACR26
0xFC0E_C000 – 0xFC0E_FFFF	RFU	
0xFC0F_0000 – 0xFC0F_3FFF	PFM – Platform Flash Module registers	OPACR28
0xFC0F_4000 – 0xFFFF_FFFF	RFU	

16.4.3 Register Descriptions

16.4.3.1 Master Privilege Registers (MPROT)

Each MPR specifies eight 4-bit fields defining the access privilege level associated with a bus master. The registers provide one field per bus master.

Access: User read/write

	3	2	1	0
R	0	MTR	MTW	MPL
W				
Reset	0	1	1	1

Figure 16-3. Master Protection Registers (MPROT)

Table 16-4. MPROT Field Descriptions

Field	Description
0 MPL	Master Privilege Level. This bit determines how the privilege level of the master is determined. 0 Accesses from this master are forced to user-mode regardless of the master's access attribute. 1 Accesses from this master are not forced to user-mode. The master's access attribute is used directly to determine the peripheral's access attribute
1 MTW	Master Trusted for Writes. This bit determines whether the master is trusted for write accesses 0 This master is not trusted for write accesses. 1 This master is trusted for write accesses.
2 MTR	Master Trusted for Reads. This bit determines whether the master is trusted for read accesses 0 This master is not trusted for read accesses. 1 This master is trusted for read accesses.
3	Reserved, should be cleared.

Accesses to registers or register fields which correspond to master or peripheral locations which are not implemented will return zeros on reads, and will be ignored on writes.

16.4.3.2 Peripheral Access Control Registers (PACR)

Each of the on-platform peripherals have a Peripheral Access Control Register which defines the access levels supported by the given module. Each PACR has the following format:

Access: User read/write

	3	2	1	0
R	0	SP	WP	TP
W				
Reset, PACR0	0	1	0	1
Reset, PACR*n*	0	1	0	0

Figure 16-4. Peripheral Access Control Registers (PACR)

Table 16-5. PACR Field Descriptions

Field	Description
0 TP	Trusted Protect. This bit determines whether the peripheral allows accesses from an untrusted master. 0 Accesses from an untrusted master are allowed. 1 Accesses from an untrusted master are not allowed. If an access is attempted by an untrusted master, the access is terminated with an error response and no peripheral access is initiated on the IPS bus.
1 WP	Write Protect. This bit determines whether the peripheral allows write accesses 0 This peripheral allows write accesses. 1 This peripheral is write protected. If a write access is attempted, the access is terminated with an error response and no peripheral access is initiated on the IPS bus.

Table 16-5. PACR Field Descriptions (Continued)

Field	Description
2 SP	Supervisor Protect. This bit determines whether the peripheral requires supervisor privilege level for access. 0 This peripheral does not require supervisor privilege level for accesses. 1 This peripheral requires supervisor privilege level for accesses. The master privilege level must indicate supervisor via the hprot[1] access attribute, and the MPROTx[MPL] control bit for the master must be set. If not, the access is terminated with an error response and no peripheral access is initiated on the IPS bus.
3	Reserved, should be cleared.

Presence or absence of a particular PACR is based on whether the associated peripheral is present in the platform, as denoted by the synthesis configuration parameters defined in Section 16.5.1, “AIPS Scalability.” When absent, the corresponding PACR is not implemented and will read as 0’s. Writes will be ignored.

16.4.3.3 Off-Platform Peripheral Access Control Registers (OPACRs)

Each of the off-platform peripherals have an Off-platform Peripheral Access Control Register (OPACR) which defines the access levels supported by the given module. Each OPACR has a format identical to the PACR described in Section 16.4.3.2, “Peripheral Access Control Registers (PACR).”

Each OPACR corresponds to the equivalent off-platform peripheral; OPACR0 corresponds to off-platform peripheral0, etc., with OPACR32 corresponding to off-platform global peripheral0, and OPACR33 corresponding toff-platform global peripheral1.

Presence or absence of a particular OPACR is design implementation dependent. When absent, the corresponding OPACR is not implemented and will read as 0’s. Writes will be ignored.

16.5 Functional Description

The AIPS serves as an interface between an AHB 2.v6 system bus and the IPS peripheral bus. It functions as a protocol translator. Accesses which fall within the address space of the AIPS are decoded to provide individual module selects for peripheral devices on the IPS interface.

16.5.1 AIPS Scalability

The AIPS is configurable to support eight masters, from one to thirty-two fixed-size on-platform peripherals, thirty-two fixed-size off-platform peripherals, and two off-platform global peripherals.

16.5.1.1 Peripheral Presence

When a particular peripheral or set of peripherals are not present in a design, those peripheral memory spaces which are not present will not be accessible by software and will be terminated with an AHB ERROR response by the AIPS.

16.5.1.2 Registers

Any attempt to access registers or register bitfields which are not present in the final instantiation will result in a read returning zeroed data and writes being ignored.

16.5.2 Access Protections

The AIPS provides programmable access protections for both masters and peripherals. It allows the privilege level of a master to be overridden, forcing it to user-mode privilege, and allows masters to be designated as trusted or untrusted. Peripherals may require supervisor privilege level for access, may restrict access to a trusted master only, and may be write-protected.

This functionality is described in Section 16.4.3.1, “Master Privilege Registers (MPROT),” Section 16.4.3.2, “Peripheral Access Control Registers (PACR),” and Section 16.4.3.3, “Off-Platform Peripheral Access Control Registers (OPACRs).”

16.5.3 Access Support

Aligned word and halfword accesses, as well as byte accesses are supported for 32-bit peripherals. Peripheral registers must not be misaligned, although no explicit checking is performed by the AIPS.

16.5.4 Read Cycles

Two clock read accesses are possible to on-platform peripherals with the AIPS when the requested access size is 32-bits or smaller.

16.5.5 Write Cycles

Three clock write accesses are possible to on-platform peripherals with the AIPS when the requested access size is 32-bits or smaller.

16.5.6 Aborted Cycles

The AIPS follows a standard procedure when a system bus cycle is aborted and the abort is initiated by the AIPS itself or the targeted IP bus peripheral. The AIPS either blocks initiation or immediately terminates any IP bus activity that is ongoing.

There are several conditions that can cause the AIPS to abort the current operation and report an error. The first is the case in which the targeted IP bus peripheral asserts a bus error. In this case the AIPS immediately terminates access to the targeted IP bus peripheral and follows the abort procedure described above. Whether the current IP bus access is a multi-cycle access or a single cycle access has no bearing on the behavior of the AIPS. The AIPS responds identically in both cases.

The second case that can cause an error response to the AHB is when an access is attempted to an IP bus peripheral whose corresponding PACR or OPACR settings do not allow the access, thus causing a permissions violation. In this case the AIPS does not initiate any IP bus activity, but instead responds by following the abort procedure described above.

The third case that can cause an error response to the AHB is when an access is attempted to a location at which there is no IP bus peripheral. In this case the AIPS does not initiate any IP bus activity but instead responds by following the same abort procedure described above for a permissions violation.

16.6 Initialization/Application Information

The AIPS is configured at reset with all masters trusted for both reads and writes, all masters using their supervisor access attribute, and all peripherals supervisor protected. After reset, a bus master may be used to change the Master Privilege Registers (MPRs), the Peripheral Access Control registers (PACRs), and the Off-platform Peripheral Access Control registers (OPACRs) as needed as described in Section 16.4.2, "Control Registers."

16.7 AIPS Bus Aborts

16.7.1 IPI Register Interface

The AIPS module supports Peripheral Bus bus aborts, and enforces the following memory map:

Table 16-6. AIPS Bus Aborts

Abort	Allowed
	$0000-$0007
$0008-$001f	
	$0020-$002f
$0030-$003f	
	$0040-$0057
$0058-$3fff	

Address aborting in the AIPS is done only on a 32-bit address boundary.

Supervisor Access: All accesses to the AIPS registers must be in Supervisor Mode. All User Mode accesses are aborted.

16.7.2 IPI Bridge Interface

In addition to the IPI interface, the AIPS can abort transfers through the bridge itself. The following transfers will be aborted:

- User mode access to any AIPS slave with the corresponding SP bit set in the PACR/OPACR register.
- Write to any AIPS slave with the corresponding WP bit set in the PACR/OPACR register.
- Any access from an untrusted master to any AIPS slave with the corresponding TP bit set in the PACR/OPACR register.
- Any access to an AIPS slave that is not implemented (i.e.-marked as "Reserved" in the documentation)

(See Section 9.9, "Peripheral Bus Memory Map" for a list of PACR/OPACR registers)

16.8 AIPS Differences from MAC71xx

- Different mix of peripherals, as follows:

Table 16-7. MAC71x1 versus MAC72xx AIPS PACR Assignment

Address	MAC71x1	MAC72xx	PACR register
$FC00 0000	AIPS	AIPS	PACR0
$FC00 4000	AXBS	AXBS	PACR1
$FC00 8000	EIM	FlexBus	PACR2
$FC00 C000 to $FC03 FFFF	Reserved	Reserved	
$FC04 0000	MCM	MCM	PACR16
$FC04 4000	DMA	DMA	PACR17
$FC04 8000	INTC	INTC	PACR18
$FC04 C000 to $FC07 FFFF	Reserved	Reserved	
$FC08 0000	SSM	SSM	OPACR0
$FC08 4000	DMA Mux	DMA Mux	OPACR1
$FC08 8000	CRG	CRG	OPACR2
$FC08 C000	PIT	PIT	OPACR3
$FC09 0000	VREG	Reserved	
$FC09 4000	CAN_A	CAN_A	OPACR5
$FC09 8000	CAN_B	CAN_B	OPACR6
$FC09 C000	CAN_C	Reserved	
$FC0A 0000	CAN_D	Reserved	
$FC0A 4000	Reserved	Reserved	
$FC0A 8000	Reserved	Reserved	
$FC0A C000	IIC_A	IIC_A	OPACR11
$FC0B 0000	Reserved	Reserved	
$FC0B 4000	SPI_A	SPI_A	OPACR13
$FC0B 8000	SPI_B	SPI_B	OPACR14
$FC0B C000	Reserved	SPI_C	OPACR15
$FC0C 0000	Reserved	Reserved	
$FC0C 4000	SCI_A	SCI_A	OPACR17
$FC0C 8000	SCI_B	SCI_B	OPACR18
$FC0C C000	SCI_C	Reserved	
$FC0D 0000	SCI_D	Reserved	
$FC0D 4000	Reserved	Reserved	
$FC0D 8000	Reserved	Reserved	
$FC0D C000	eMIOS	eMIOS	OPACR23
$FC0E 0000	ADC_A	ADC_A	OPACR24
$FC0E 4000	ADC_B	Reserved	

Table 16-7. MAC71x1 versus MAC72xx AIPS PACR Assignment (Continued)

Address	MAC71x1	MAC72xx	PACR register
$FC0E 8000	PIM	PIM	OPACR26
$FC0E C000	Reserved	Reserved	
$FC0F 0000	CFM_REGS	H7FB_REGS	OPACR28
$FC0F 4000 to $FC0F FFFF	Reserved		

Chapter 17
External Bus Interface (FlexBus)

17.1 Introduction

This chapter describes data transfer operations, error conditions, and reset operations. It describes transfers initiated by the MAC7200 and includes detailed timing diagrams showing the interaction of signals in supported bus operations.

NOTE

Unless otherwise noted, in this chapter “clock” refers to the CLKOUT used for the bus.

The external bus interface on the MAC7200 can be used as a bus interface to external peripherals or memories. In general, the external bus is available on larger pin count packages only, which also makes it ideal for debugging purposes. It is a simple, flexible bus that supports a variety of external memory devices with little or no external logic needed, and is based on the external bus of the 68000/Coldfire family of devices. All transfers on the external bus are controlled by the MCU masters (CPU or eDMA) only (i.e., the MAC7200 device may not be used as an external bus slave).

The interface is implemented with a 16-bit data bus capable of supporting byte, half word (2 byte) and word (4 byte) transfers.

Accesses which are greater than the selected port width of the data bus (i.e.,16-bit access with an 8-bit Port Size) are automatically decomposed into a sequence of individual transfers. The address bus has up to 22 address lines (ADDR0[0:21]), allowing access to up to 4Mbytes of external addresses.

Three chip select signals ($\overline{\text{CS}}$[0:2]) can be configured for different address ranges and transfers types, including Auto Acknowledge with a defined number of wait states. If an access to the external bus maps to no chip selects, the FlexBus will return a bus abort. Accesses that map to multiple chip selects will cause multiple chip selects to be asserted. It is the responsibility of the external hardware to correctly handle this situation, especially in the case of a read.

Chip select $\overline{\text{CS0}}$ can be dedicated to boot ROM access and can be programmed to be byte (8 bits), word (16 bits), or longword (32 bits) wide. Control signal timing is compatible with common ROM / flash memories.

The port width of the data bus can be selected as either 8-bits or 16-bits wide. The value of Port F3 on reset is used to determine the external data bus port width, as described in Section 2.2, “MCU Hardware Configuration Summary” and Table 17-2.

Table 17-1. External Bus Auto Acknowledge Configuration

PF2/AA	Description
0	Disabled. Use the $\overline{TA}$ input to acknowledge transfers
1	Enabled. Transfers are acknowledged internally, but may also be acknowledged using the $\overline{TA}$ input

Table 17-2. External Bus Port Size Configuration

PF3/PS	Description
0	Port Size is 8-bits (DATA0 to DATA7)
1	Port Size is 16-bits (DATA0 to DATA15)

17.1.1 Block Diagram

The block diagram of the FlexBus is shown in Figure 17-1.

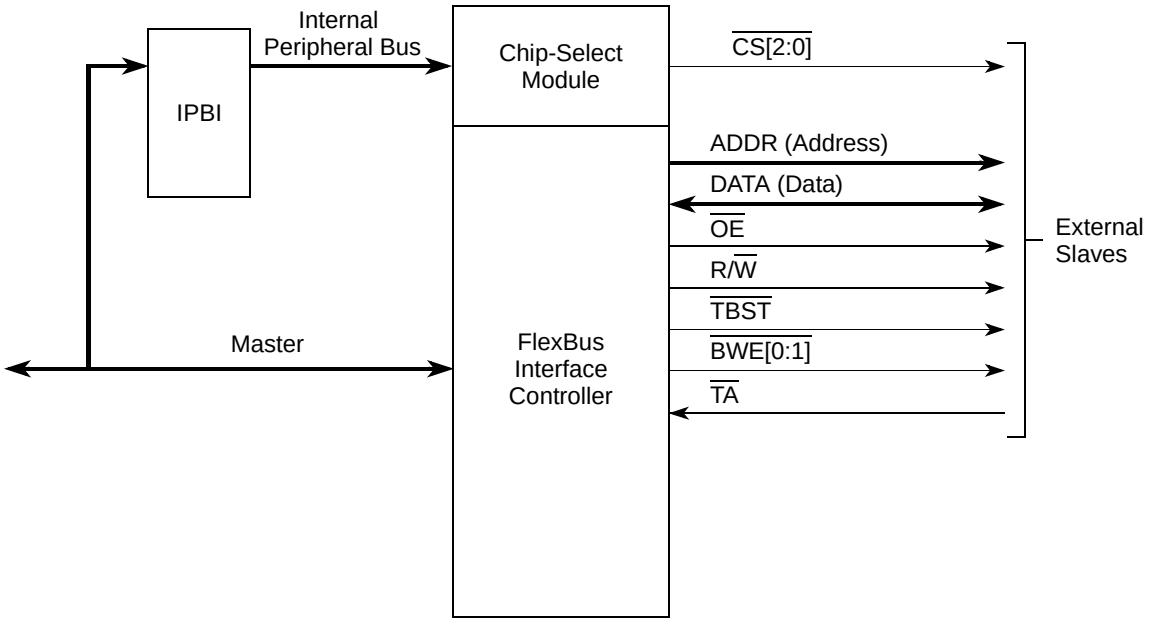


Figure 17-1. FlexBus Controller Conceptual Diagram (Non-Muxed Implementation)

17.1.2 Features

The following list summarizes the key FlexBus features on the MAC7200:

- 22-bit address bus (Maximum of 22 bits pinned out on the MAC72xx)
- 8/16-bit data bus (Maximum of 16-bits pinned out on the MAC72xx)
- 3 dedicated Chip Selects (Maximum of 3 chip selects pinned out on the MAC72xx)
- Slave only (MAC72xx is always Master)

- Implements a programmable version of the MCF5407 External Bus protocol
- Byte, word, longword, and line sized transfers
- Programmable burst and burst-inhibited transfers selectable for each chip select and transfer direction
- Programmable address setup time with respect to the assertion of chip select
- Programmable address hold time with respect to the negation of chip select and transfer direction

17.1.3 FlexBus Implementation

The FlexBus is an enhanced version of the EIM bus implemented on the MAC71xx family of devices. It has the following special enhancements over the FlexBus controllers found on other devices:

- Issues a bus abort for all transfers that do not hit any chip selects

17.1.4 FlexBus Memory Map Relocation

As discussed in Section 9.16, “Memory Map Relocation”, there is the possibility to relocate the base address of the FlexBus at reset, including the option of removing it completely from the memory map (for security reasons).

17.2 External Signals

This section describes the external signals that are involved in data transfer operations. Table 17-3 summarizes the MAC7200 FlexBus signals.

Table 17-3. FlexBus Signal Summary

Signal Name	Direction	Description	Reset State
$\overline{\text{CS[2:0]}}$[†]	O	General purpose chip-selects	Hi-Z
ADDR[21:0]	O	Address bus	Hi-Z
DATA[15:0]	I/O	Data bus	Hi-Z
$\overline{\text{BWE[1:0]}}$[1]	O	Byte Selects	Hi-Z
$\overline{\text{OE}}$[1]	O	Output Enable	Hi-Z
R/$\overline{\text{W}}$	O	Read/Write. 1 = Read, 0 = Write	Hi-Z
$\overline{\text{TBST}}$	O	Burst Transfer indicator	Hi-Z
$\overline{\text{TA}}$	I	Transfer Acknowledge	—

1. On maskset 0M34D, output is always enabled

If full address decoding is done externally for anything other than an 8-bit port size, then ADDR[21:0] and SIZE[1:0] is required, otherwise only ADDR[21:1] and $\overline{\text{BS0}}$/$\overline{\text{BS1}}$ are required. The basis for this required scheme is:

- On a 8-bit port size, only pins DATA[7:0] are available

- For an even address 8-bit access with a 16-bit port, the data should appear on **DATA[15:8]** in order to be consistent

17.2.1 Chip-Select ($\overline{\text{CS[2:0]}}$)

The chip-select signal indicates which device is being selected. A particular chip-select asserts when the transfer address is within the device's address space as defined in the base and mask address registers, see Section 17.3.2, "Chip-Select Registers."

17.2.2 Address Bus (ADDR[21:0])

The ADDR[21:0] bus carries address. The address is always driven on the first clock of a bus cycle (address phase).

17.2.3 Data Bus (DATA[15:0])

The DATA[15:0] bus carries data. The number of byte lanes used to carry the data during the data phase is determined by the port size associated with the matching chip select.

17.2.4 Read/Write (R/$\overline{\text{W}}$)

The MAC7200 drives the $\overline{\text{R/W}}$ signal to indicate the direction of the current bus operation. It is driven high during read bus cycles and driven low during write bus cycles.

17.2.5 Transfer Burst ($\overline{\text{TBST}}$)

Transfer Burst indicates that a burst transfer is in progress as driven by the MAC7200. A burst transfer can be 2 to 16 beats depending on the port size.

NOTE

When burst ($\overline{\text{TBST}}$ = 0) and the address is misaligned within the 16-byte boundary, the external device must be able to wrap around the address.

17.2.6 Byte Write Enable/Byte Select ($\overline{\text{BWE[1:0]}}$)

The byte strobe $\overline{\text{BWE[1:0]}}$) outputs indicate that data is to be latched or driven onto a byte of the data when driven low. $\overline{\text{BWE}}$*n* signals are asserted only to the memory bytes used during a read or write access.

17.2.7 Output Enable ($\overline{\text{OE}}$)

The output enable signal ($\overline{\text{OE}}$) is sent to the interfacing memory and/or peripheral to enable a read transfer. $\overline{\text{OE}}$ is asserted only when a chip select matches the current address decode.

17.2.8 Transfer Acknowledge ($\overline{TA}$)

This signal indicates that the external data transfer is complete. During a read cycle, when the processor recognizes $\overline{TA}$, it latches the data and then terminates the bus cycle. During a write cycle, when the processor recognizes $\overline{TA}$, the bus cycle is terminated.

If auto-acknowledge is disabled, the external device drives $\overline{TA}$ to terminate the bus transfer; if auto-acknowledge is enabled, the $\overline{TA}$ is generated internally after a specified wait states or the external device may assert external $\overline{TA}$ before the wait-state countdown which in turn terminates the cycle early. The MAC7200 negates $\overline{CS}n$ a cycle after the last $\overline{TA}$ asserts. During read cycles, the peripheral must continue to drive data until $\overline{TA}$ is recognized. For write cycles, the processor continues to drive data one clock after $\overline{CS}n$ is negated.

The number of wait states is determined either by internally programmed auto acknowledgement or by the external $\overline{TA}$ input. If the external $\overline{TA}$ is used, the peripheral has total control on the number of wait states.

17.3 Chip-Select Operation

Each chip-select has a dedicated set of the following registers for configuration and control.

- Chip-select address registers (CSARn) control the base address space of the chip-select. Section 17.3.2.1, “Chip-Select Address Registers (CSAR0–CSAR2).
- Chip-select mask registers (CSMRn) provide 16-bit address masking and access control. Section 17.3.2.2, “Chip-Select Mask Registers (CSMR0–CSMR2).
- Chip-select control registers (CSCRn) provide port size and burst capability indication, wait-state generation, address setup and hold times, and automatic acknowledge generation features. Section 17.3.2.3, “Chip-Select Control Registers (CSCR0–CSCR2).

$\overline{CS0}$ is a global chip-select after reset and provides re-locatable boot ROM capability.

17.3.1 General Chip-Select Operation

When a bus cycle is initiated, the MAC7200 first compares its address with the base address and mask configurations programmed for chip-selects 0–2 (configured in CSCR0–CSCR2). If the driven address matches a programmed chip-select, the appropriate chip-select is asserted fulfilling the requirements as programmed in the respective configuration register.

17.3.1.1 8-bit and 16-bit Port Sizing

Static bus sizing is programmable through the port size bits, CSCR[PS]. See Section 17.3.2.3, “Chip-Select Control Registers (CSCR0–CSCR2).” Note that the MAC7200 always drives a 22-bit address on the address bus in the first cycle regardless of the external device’s address size. The external device must connect its address lines to the appropriate ADDR bits starting from ADDR0 and upward. It must also connect its data lines to the DATA bus starting from the DATA15 and downward. No bit ordering is required when connecting address and data lines to the bus. For example, a 16-bit address/16-bit data device would connect its addr[15:0] to ADDR[15:0] and data[15:0] to DATA[15:0]. See Figure 17-5 for graphical connection.

17.3.1.2 Global Chip-Select Operation

$\overline{CS0}$, the global (boot) chip-select, allows address decoding for boot ROM before system initialization. Its operation differs from other external chip-select outputs after system reset.

After system reset, $\overline{CS0}$ is asserted for every external access. No other chip-select can be used until the valid bit, CSMR0[V], is set, at which point $\overline{CS0}$ functions as configured. After this, $\overline{CS}$[2:1] can be used as well. At reset, the port size, and automatic acknowledge functions of the global chip-select are determined by the logic levels on the PF[3:2] signals (see Table 17-1 and Table 17-2).

17.3.2 Chip-Select Registers

The following tables describe in detail the registers and bit meanings for configuring chip-select operation. The chip-select controller register map is accessed relative to the memory base address register (MBAR). Table 17-4 shows the chip-select register memory map. Reading unused or reserved locations terminates normally and returns zeros.

Table 17-4. Chip-Select Registers

Offset	[31:24]	[23:16]	[15:8]	[7:0]	ResetValue	Access [1]
0x00	Chip-select address register—bank 0 (CSAR0)		Reserved [2]		0x0000_0000	R/W
0x04	Chip-select mask register—bank 0 (CSMR0)				0x0000_0000	R/W
0x08	Chip-select control register—bank 0 (CSCR0)				BSTW = 0 BSTR = 0 PS = AD[1:0] AA = AD[2] WS = 111111 WRAH = 11 RDAH = 11 ASET = 11 SWSEN = 0 SWS = 000000	R/W
0x0C	Chip-select address register—bank 1 (CSAR1)		Reserved [2]		0x0000_0000	R/W
0x10	Chip-select mask register—bank 1 (CSMR1)				0x0000_0000	R/W
0x14	Chip-select control register—bank 1 (CSCR1)				0x0000_0000	R/W
0x18	Chip-select address register—bank 2 (CSAR2)		Reserved [2]		0x0000_0000	R/W
0x1C	Chip-select mask register—bank 2 (CSMR2)				0x0000_0000	R/W
0x20	Chip-select control register—bank 2 (CSCR2)				0x0000_0000	R/W

1. The access column indicates whether the corresponding register allows both read/write functionality (R/W), read-only functionality (R), or write-only functionality (W). A read access to a write-only register returns zeros. A write access to a read-only register has no effect.
2. Addresses not assigned to a register and undefined register bits are reserved for expansion. Write accesses to these reserved address spaces and reserved register bits have no effect.

17.3.2.1 Chip-Select Address Registers (CSAR0–CSAR2)

CSAR*n*, Figure 17-2, specify the chip-select base addresses.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	BA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Reg Addr	MBAR + 0x00 (CSAR0); 0x0C (CSAR1); 0x18 (CSAR2)															

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Reg Addr	MBAR + 0x00 (CSAR0); 0x0C (CSAR1); 0x18 (CSAR2)															

Figure 17-2. Chip-Select Address Registers (CSAR*n*)

Table 17-5. CSAR*n* Field Descriptions

Bits	Name	Description
31–16	BA	Base address. Defines the base address for memory dedicated to chip-select $\overline{\text{CS}}n$. BA is compared to bits 31–16 on the internal address bus to determine if chip-select memory is being accessed.
15–0	—	Reserved, should be cleared

17.3.2.2 Chip-Select Mask Registers (CSMR0–CSMR2)

CSMR*n*, Figure 17-2, are used to specify the address mask and allowable access types for the respective chip-selects.

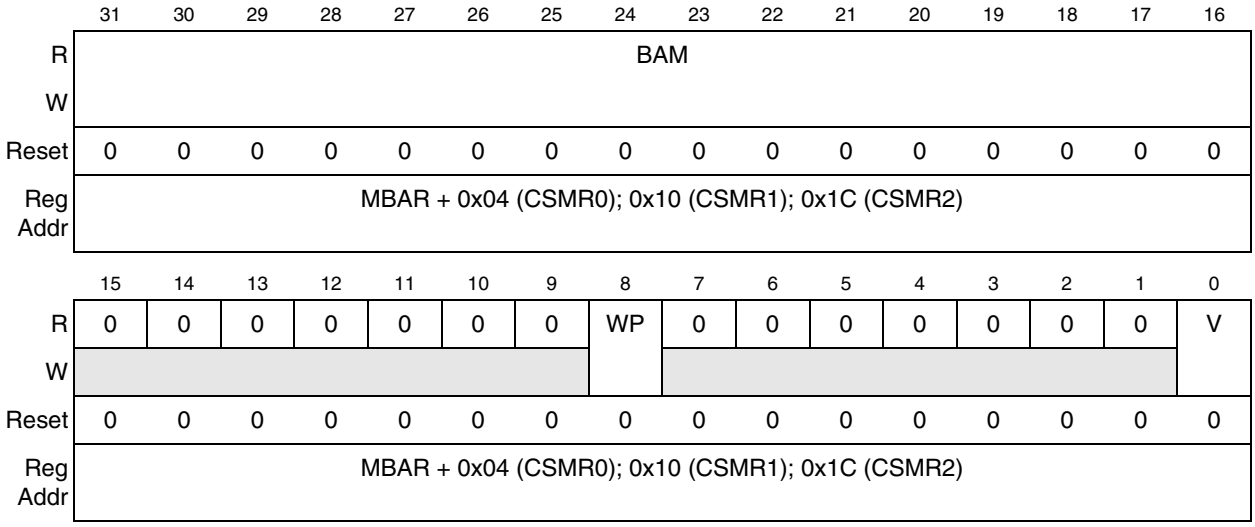


Figure 17-3. Chip-Select Mask Registers (CSMR*n*)

Table 17-6 describes CSMR fields.

Table 17-6. CSMR*n* Field Descriptions

Bits	Name	Description
31–16	BAM	Base address mask. Defines the chip-select block size by masking address bits. Setting a BAM bit causes the corresponding CSAR bit to be a “don’t care” in the decode. 0 Corresponding address bit is used in chip-select decode. 1 Corresponding address bit is a don’t care in chip-select decode. The block size for $\overline{CS}n$ is 2^n; n = (number of bits set in respective CSMR[BAM]) + 16. For example, if CSAR0 = 0x0000 and CSMR0[BAM] = 0x0008, $\overline{CS0}$ would address two discontinuous 64-Kbyte memory blocks: one from 0x0000–0xFFFF and one from 0x8_0000–0x8_FFFF. In another example, $\overline{CS0}$ is setup to access 32 Mbytes of address space starting at location 0x0 and $\overline{CS1}$ is setup to access 16 Mbytes at the next byte after $\overline{CS0}$. Then CSAR0 = 0x0000, CSMR0[BAM] = 0x01FF, CSAR1 = 0x0200, and CSMR1[BAM] = 0x00FF.However, the size of the window may also physically be constrained by the size of the data port, which constrains the number of address lines available in some modes.
15–9	—	Reserved, should be cleared
8	WP	Write protect. Controls write accesses to the address range in the corresponding CSAR. Attempting to write to the range of addresses for which CSAR*n*[WP] = 1 results in the appropriate chip-select not being selected and . No exception occurs. 0 Both read and write accesses are allowed 1 Only read accesses are allowed
7–1	—	Reserved, should be cleared
0	V	Valid bit. Indicates whether the corresponding CSAR, CSMR, and CSCR contents are valid. Programmed chip-selects do not assert until V bit is set (except for $\overline{CS0}$, which acts as the global chip-select). Reset clears each CSMR*n*[V]. At reset, no chip-select other than $\overline{CS0}$ can be used until the CSMR0[V] is set. At which point $\overline{CS}[2:0]$ functions as configured. 0 chip-select invalid 1 chip-select valid

17.3.2.3 Chip-Select Control Registers (CSCR0–CSCR2)

Each CSCR*n*, Figure 17-3, controls the auto acknowledge, external master support, address setup and hold times, port size, burst capability, and activation of each chip-select. Note that to support the global chip-select, $\overline{CS0}$, the CSCR0 reset values differ from the other CSCRs. $\overline{CS0}$ allows address decoding for boot ROM before system initialization.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	SWS						0	0	SWSEN	—	ASET		RDAH		WRAH	
W																
Reset: CSCR0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1
Reset: CSCRs	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Reg Addr	MBAR + 0x08 (CSCR0); 0x14 (CSCR1); 0x20 (CSCR2)															

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	WS						0	AA	PS		0	BSTR	BSTW	0	0	0
W																
Reset: CSCR0	1	1	1	1	1	1	0	AD2	AD[1:0]		0	0	0	0	0	0
Reset: CSCRs	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Reg Addr	MBAR + 0x08 (CSCR0); 0x14 (CSCR1); 0x20 (CSCR2)															

Figure 17-4. Chip-Select Control Registers (CSCR*n*)

Table 17-7 describes CSCR*n* fields.

Table 17-7. CSCR*n* Field Descriptions

Bits	Name	Description
31–26	SWS	Secondary wait states. The number of wait states inserted before an internal transfer acknowledge is generated for burst transfer except for the first termination, which is controlled by the wait state count. The secondary wait state is only used if the secondary wait state enable is set, otherwise the wait state value is used for all burst transfers.
25–24	—	Reserved, should be cleared
23	SWSEN	Secondary wait state enable. 0 The wait state value is used to insert wait states before an internal transfer acknowledge is generated for all transfers. 1 The secondary wait state value is used to insert wait states before an internal transfer acknowledge is generated for burst transfer secondary terminations.
22	—	Reserved, should be cleared

Table 17-7. CSCR*n* Field Descriptions (Continued)

Bits	Name	Description
21–20	ASET	Address setup. This field controls the asserting of chip-select with respect to assertion of a valid address and attributes. 00 Assert chip-select on rising clock edge after address is asserted. (Default $\overline{CS}n$) 01 Assert chip-select on second rising clock edge after address is asserted. 10 Assert chip-select on third rising clock edge after address is asserted. 11 Assert chip-select on fourth rising clock edge after address is asserted.(Reset $\overline{CS0}$)
19–18	RDAH	Read Address Hold or (Deselect). This field controls the address and attribute hold time after the termination during a read cycle that hits in the chip-select address space. 00 Hold address and attributes one cycle after $\overline{CS}n$ negates on reads. (Default $\overline{CS}n$) 01 Hold address and attributes two cycles after $\overline{CS}n$ negates on reads. 10 Hold address and attributes three cycles after $\overline{CS}n$ negates on reads. 11 Hold address and attributes four cycles after $\overline{CS}n$ negates on reads. (Reset $\overline{CS0}$)
17–16	WRAH	Write Address Hold or (Deselect). This field controls the address, data and attribute hold time after the termination of a write cycle that hits in the chip-select address space.The hold time only applies at the end of a transfer. Therefore, a burst transfer which is burst inhibited only has a hold time added after the last bus cycle.] 00 Hold address and attributes one cycle after $\overline{CS}n$ negates on writes. (Default $\overline{CS}n$) 01 Hold address and attributes two cycles after $\overline{CS}n$ negates on writes. 10 Hold address and attributes three cycles after $\overline{CS}n$ negates on writes. 11 Hold address and attributes four cycles after $\overline{CS}n$ negates on writes. (Reset $\overline{CS0}$)
15–10	WS	Wait states. The number of wait states inserted after $\overline{CS}n$ asserts and before an internal transfer acknowledge is generated (WS = 0 inserts zero wait states, WS = 0x3F inserts 63 wait states). If AA = 0, $\overline{TA}$ must be asserted by the external system regardless of the number of wait states generated. In that case, the external transfer acknowledge ends the cycle. An external $\overline{TA}$ supersedes the generation of an internal $\overline{TA}$.
9	—	Reserved, should be cleared.
8	AA	Auto-acknowledge enable. Determines the assertion of the internal transfer acknowledge for accesses specified by the chip-select address. 0 No internal $\overline{TA}$ is asserted. Cycle is terminated externally. 1 Internal $\overline{TA}$ is asserted as specified by WS. Note that if AA = 1 for a corresponding $\overline{CS}n$ and the external system asserts an external $\overline{TA}$ before the wait-state countdown asserts the internal $\overline{TA}$, the cycle is terminated. Burst cycles increment the address bus between each internal termination.
7–6	PS	Port size. Specifies the width of the data port associated with each chip-select. It determines where data is driven during write cycles and where data is sampled during read cycles. 00 32-bit port size. Valid data sampled and driven on D[31:0]. Not valid on MAC72xx. 01 8-bit port size. Valid data sampled and driven on D[7:0] 1x 16-bit port size. Valid data sampled and driven on D[15:0]
5	—	Reserved, should be cleared.
4	BSTR	Burst read enable. Specifies whether burst reads are used for memory associated with each $\overline{CS}n$. 0 Data exceeding the specified port size is broken into individual, port-sized non-burst reads. For example, a longword read from an 8-bit port is broken into four 8-bit reads. 1 Enables data burst reads when the transfer size is larger than the specified port size, including longword reads from 8- and 16-bit ports, word reads from 8-bit ports, and line reads from 8- and 16-bit ports.

Table 17-7. CSCR*n* Field Descriptions (Continued)

Bits	Name	Description
3	BSTW	Burst write enable. Specifies whether burst writes are used for memory associated with each $\overline{\text{CS}}n$. 0 Break data larger than the specified port size into individual port-sized, non-burst writes. For example, a longword write to an 8-bit port takes four byte writes. 1 Enables burst write of data when the transfer size is larger than the specified port size, including longword writes to 8 and 16-bit ports, word writes to 8-bit ports and line writes to 8- and 16--bit ports.
2–0	—	Reserved, should be cleared.

17.4 Functional Description

17.4.1 Data Transfer Operation

Data transfers between the MAC7200 and other devices involve the following signals:

- Address / data bus (ADDR[21:0] / DATA[15:0])
- Control signal $\overline{\text{TA}}$
- $\overline{\text{CSn}}$
- $\overline{\text{OE}}$
- $\overline{\text{BWE[1:0]}}$
- Attribute signals (R/$\overline{\text{W}}$, $\overline{\text{TBST}}$)

The address and write data (ADDR[21:0] and DATA[15:0]), R/$\overline{\text{W}}$, $\overline{\text{CS}}n$, and all attribute signals change on the rising edge of the clock. Read data is registered in the MAC7200 on the rising edge of the clock.

The MAC7200 FlexBus supports byte, word, longword, and 16-byte cache line operand transfers and allows accesses to 8- and 16-bit data ports. Transfer parameters such as address setup and hold, port size, the number of wait states for the external device being accessed, automatic internal transfer termination enable or disable, and burst enable or disable are programmed in the chip-select control registers (CSCRs), Section 17.3.2.3, “Chip-Select Control Registers (CSCR0–CSCR2).

17.4.2 Data Byte Alignment and Physical Connections

The MAC7200 aligns data transfers in FlexBus byte lanes, the number of lanes depending on the width of the data port. Figure 17-5 shows the byte lanes that external memory should be connected to and the sequential transfers if a longword is transferred for two port sizes. For example, an 8-bit memory should be connected to the single lane DATA[7:0]. A longword transfer through this 8-bit port takes four transfers on DATA[7:0], starting with the MSB and going to the LSB. A longword transfer through a 16-bit port requires two transfers on each of the two byte lanes of the FlexBus.

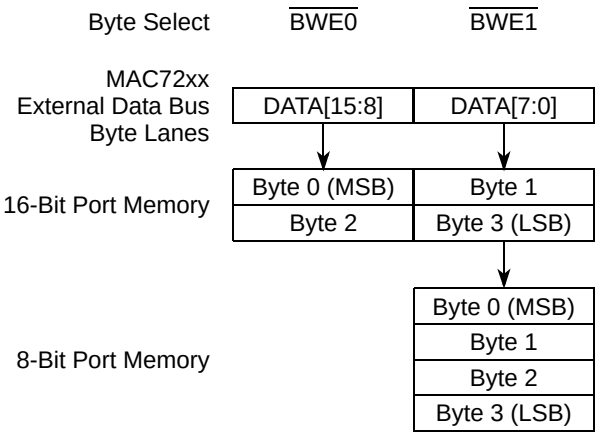


Figure 17-5. Connections for External Memory Port Sizes

17.4.3 Bus Cycle Execution

Basic bus operations occur in four clocks, as follows:

1. At the first clock edge, the address, and attributes are driven.
2. $\overline{\text{CS}}n$ is asserted at the second rising clock edge to indicate which device has been selected and by that time the address and attributes are valid and stable. is negated at this edge.

   For a write transfer, data is driven on the bus at this clock edge and continues to be driven until one clock cycle after $\overline{\text{CS}}n$ negates. For a read transfer, data is also returned at this cycle.

   External slave asserts $\overline{\text{TA}}$ at this clock edge.
3. Read data and $\overline{\text{TA}}$ are sampled on the third clock edge. $\overline{\text{TA}}$ can be negated after this edge and read data can then be tristated.
4. $\overline{\text{CS}}n$ is negated at the fourth rising clock edge. This last clock of the bus cycle uses what would be an idle clock between cycles to provide hold time for address, attributes, and write data.

17.4.3.1 Data Transfer Cycle States

The data transfer operation in the MAC7200 is controlled by an on-chip state machine. The state transition diagram for basic read and write cycles is shown in Figure 17-6.

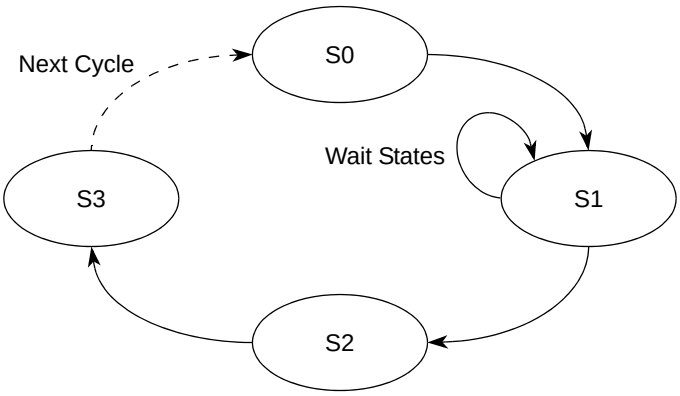


Figure 17-6. Data Transfer State Transition Diagram

Table 17-8 describes the states as they appear in subsequent timing diagrams.

Table 17-8. Bus Cycle States

State	Cycle	Description
S0	All	The read or write cycle is initiated. On the rising clock edge, the MAC7200 places a valid address on ADDR[21:0]and drives R/W high for a read and low for a write, if these signals are not already in the appropriate state.
S1	All	On the rising edge of CLK, $\overline{CS}n$ is asserted. Data is driven on DATA[15:Y] for writes, and DATA[15:Y] is three-stated for reads. Address continues to be driven on ADDR[X:0] pins that are unused for data. If $\overline{TA}$ is recognized asserted, then the cycle moves on to S2. If $\overline{TA}$ is not asserted either internally or externally, then the S1 state continues to repeat.
	Read	Data is made available by the external device before the rising edge of CLK with $\overline{TA}$ asserted. The MAC7200 will latch data on this rising clock edge.
S2	All	For internal termination the chip select will negate the internal $\overline{TA}$ signal. For external termination, the external device should negate $\overline{TA}$. Chip select is negated after the rising edge of CLK at the end of S2.
	Read	The external device can stop driving data after the rising edge of CLK at the beginning of S2. However, data can be driven until the end of S3 or any additional address hold cycles.
S3	All	Address, data, and R/$\overline{W}$ go invalid off the rising edge of CLK at the end of S3, terminating the read or write cycle.

17.5 FlexBus Bus Aborts

17.5.1 IPI Register Interface

The FlexBus module does not support Peripheral Bus aborts, and has the following memory map:

Table 17-9. FlexBus Bus Aborts

Abort	Allowed
	$0000-$3fff
None	

Supervisor Access: Unused

17.5.2 FlexBus Interface

In addition to the IPI interface, the FlexBus can also abort external bus transfers. The following transfers will be aborted:

- Transfers to an external bus address which does not map into any chip select address range

17.6 FlexBus Differences from MAC71xx

- The FlexBus interface has numerous small differences from the EIM. Please refer to the respective Block Guides for differences in programming interface and protocol.
  - Changed offset of registers from $0080 to $0000
  - Changed max number of wait states from 15 to 63
  - Added Secondary Wait State functionality (SWS, SWSEN bits in the CSCR)
  - Added Extended Transfer Start functionality (EXTS bit in the CSCR)
  - Added Address Setup and Hold functionality (ASET, RDAH, WDAH bits in the CSCR)
  - Added Multiplexed address Mode (MUX bit in the CSCR)

Table 17-10. MAC71x1 to MAC72xx External Bus mapping

MAC71x1	MAC72xx
A[21:0]	ADDR[21:0]
D[15:0]	DATA[15:0]
$\overline{CS0}$	$\overline{CS0}$
$\overline{CS1}$	$\overline{CS1}$
$\overline{CS2}$	$\overline{CS2}$
$\overline{OE}$	$\overline{OE}$
$\overline{BS0}$	$\overline{BWE0}$
$\overline{BS1}$	$\overline{BWE1}$
R$\overline{W}$	R$\overline{W}$
CLKOUT	CLKOUT
$\overline{TA}$	$\overline{TA}$
$\overline{AS}$	-
-	$\overline{TBST}$

- The mix of external bus signals available at pins is different from the MAC71x1 to the MAC72xx.
- Fixed MUCts01460: Thumb mode accesses have incorrect address and/or byte selects

17.7 FlexBus Application Usage

17.7.1 Enabling the FlexBus

It is not necessary to enable the FlexBus before it can be used. Please see Section 17.7.2, “Global Chip Select Mode below for more information on the behavior of the FlexBus out of reset.

17.7.2 Global Chip Select Mode

After reset, the FlexBus is put into a Global Chip Select Mode, where all external bus accesses map to $\overline{CS0}$. This allows the maximum flexibility to boot from external sources before the FlexBus is configured. The configuration of $\overline{CS0}$ in Global Chip Select Mode is as follows:

Table 17-11. Global Chip Select Mode Configuration

Register	Field Values at Reset
CSAR0	0x0000_0000
CSMR0	0x0000_0000
CSCR0	SWS[31:26] = 0x00
	SWSEN[23] = 0
	ASET[21:20] = 11
	RDAH[19:18] = 11
	WRAH[17:16] = 11
	WS[15:10] = 0x3f (63 wait states)
	MUX[9] = 0
	AA[8] = from pin PF2
	PS1[7] = from pin PF3
	PS0[6] = 1
	BEM[5] = 0
	BSTR[4] = 0
	BSTW[3] = 0

To exit Global Chip select mode, set the Valid bit in the CSMR0 register. The only possibility to enter Global Chip Select mode again after that is a Reset. Until Global Chip Select Mode is exited, all chip selects, other than $\overline{CS0}$, are unavailable. Outside of Global Chip Select mode, the $\overline{CS0}$ registers behave identically to all other chip select registers.

When modifying the configuration of a chip select, it is generally recommended to first clear the Valid bit for that chip select in order to avoid possible race conditions between external bus accesses and

configuring of the external bus. If software can positively assure that no external bus accesses will occur during the configuration of the chip select(s), then this step is not necessary.

To summarize, a short example is presented. To boot from an external device, and enable three chip selects on the external bus, you would perform the following steps:

1. Boot the device (either Power On Reset or assert the $\overline{\text{RESET}}$ pin)
2. The external bus (if available) will be in Global Chip Select Mode, with all external bus accesses mapped to $\overline{\text{CS0}}$. $\overline{\text{CS1}}$ and $\overline{\text{CS2}}$ will be unavailable.
3. If the device is in Expanded Mode, it will boot from the external bus, causing $\overline{\text{CS0}}$ to be asserted, and a READ access to appear on the external bus. The Port Size and Auto Acknowledge bits (CSCR[7] and CSCR[8], respectively) are determined by the value of the PF3 and PF2 pins (respectively) at Reset.
4. The boot code (from the external device) can now re-configure all of the chip-selects, including $\overline{\text{CS0}}$. $\overline{\text{CS0}}$ should be re-configured in most cases, because the Global Chip Select mode provides the most flexible (and hence slowest) access to external devices.
5. To configure $\overline{\text{CS1}}$, program the CSAR1, CSCR1 and CSMR1 register, ensuring that the Valid bit (CSMR1[0]) is set. Even though the Valid bit is set, $\overline{\text{CS1}}$ will not become active until Global Chip Select Mode is exited.
6. Configuring $\overline{\text{CS2}}$ is similar to configuring $\overline{\text{CS1}}$.
7. To configure $\overline{\text{CS0}}$ and exit Global Chip Select Mode, program the CSAR0, CSCR1 and CSMR0 registers. Once the Valid bit (CSMR0[0]) bit is set, the FlexBus will exit Global Chip Select Mode, and all three chip selects will be active with their programmed configuration. Therefore, the CSMR0 register should be programmed last.

17.7.3 FlexBus speed

Note that, due to the increase in the number of wait states in Global Chip Select Mode between the MAC71xx and MAC7200 family of devices, the MAC72xx device will run about 4x slower when booting from the external bus. For this reason, the FlexBus registers should be programmed as early in the boot process as possible in order to set the correct number of wait states.

17.7.4 How to use the external bus in Expanded Secured/Unsecured Mode

There is no PIM setup required to use the external bus in Expanded Mode. In this mode, the following pins are automatically changed to Peripheral Mode and the FlexBus clock is automatically enabled:

- PA0-PA15
- PC0-PC15
- PD0-PD2
- PD5-PD15

17.7.5 How to Use the External Bus in Single Chip Unsecured Mode

It is possible to use the external bus in Single Chip Unsecured Mode, but the PIM must be setup first. To do so, set the MODE bits (i.e.-Set Peripheral Mode) in the CONFIG register for each of the following pads:

- PA0-PA15
- PC0-PC15
- PD0-PD2
- PD5-PD15

and enable the clock to the FlexBus module by setting the **EIMCLKEN** bit in the **PIMCONFIG** register in the PIM. The following code example illustrates this:

```
#include <stdarg.h>
#include "addr_map_defines.vrh"
#include "pim_registers.h"

void main()
{
  typedef volatile unsigned char * REG8;
  typedef volatile unsigned short * REG16;
  volatile unsigned char  *pim_pointer;
  volatile unsigned short *flexbus_pointer;
  int i;

// Turn on the FlexBus clock
  pim_pointer = (REG8) (PIM_REGISTER_MAP_OFFSET+0x3C2);
  *pim_pointer = 0x002;

// Set PIM base address for Port A CONFIG registers
  pim_pointer = (REG8) (PIM_REGISTER_MAP_OFFSET+0x01);
  for (i=0; i<=15; i++) {
    *pim_pointer = 0x80;
    pim_pointer=pim_pointer+2;
  }

// Set PIM base address for Port C CONFIG registers
  pim_pointer = (REG8) (PIM_REGISTER_MAP_OFFSET+0x81);
  for (i=0; i<=15; i++) {
    *pim_pointer = 0x80;
    pim_pointer=pim_pointer+2;
  }

// Set PIM base address for Port D CONFIG registers
  pim_pointer = (REG8) (PIM_REGISTER_MAP_OFFSET+0xC1);
  for (i=0; i<=2; i++) {
    *pim_pointer = 0x80;
    pim_pointer=pim_pointer+2;
  }
  pim_pointer=pim_pointer+2;
  pim_pointer=pim_pointer+2;
  for (i=5; i<=15; i++) {
    *pim_pointer = 0x80;
    pim_pointer=pim_pointer+2;
  }
```

```
// Now let's do a 16-bit write and read to/from the external bus
  flexbus_pointer =  (REG16) (0x20000000);
  *flexbus_pointer = 0x8000;
  i=*flexbus_pointer;
```

17.7.6 Enabling and Disabling CLKOUT

Even though the CLKOUT (PD2) signal is intended primarily as the FlexBus clock, it may be desirable in some cases to enable this clock even when the FlexBus is not being used. If the CLKOUT is used primarily as part of the FlexBus interface, please see below.

To disable CLKOUT: Change pad PD2 to an input by writing 0x00 to the **CONFIG** register for PD2

To change the drive strength on CLKOUT: Set/Clear the **SLEW and SLEWEN** bits in the **CONFIG** register for PD2.

NOTE

It is not possible to use pad PD2 as a General Purpose Output.

Chapter 18
FLASH (H7Fb) and FLASH Controller (PFLASH)

This chapter provides an overview of the flash array and the flash controller as they are implemented on the MAC72xx. Please refer to Chapter 19, “Hip7a Low-Cost Embedded Flash (H7Fb)” for more detailed technical information on the flash array, and Chapter 20, “SPP Flash Controller (PFLASH_H7Fb) Module” for more detailed information on the flash controller.

18.1 Introduction

The Flash Module provides the non-volatile memory and its control interface for the MAC7200 family of devices. Its implementation offers a range of memory sizes depending on the device and intended function of the memory. For more information on the non-volatile memory options within the MAC7200 family please refer to Figure 1-1 in Chapter 1, “Introduction”.

The Flash provides two types of memory, the Flash main array and the Shadow Block. The Flash main array (also sometimes referred to as the Program Flash) is connected directly to the core via the crossbar switch (See Chapter 16, “AHB to IPI Bridge (AIPS)”). The Shadow Block can be used to store the primary boot loader. This memory is accessed across the same interface as the Flash main array and is constructed using the same basic array and memory cell as the Flash main array.

On the MAC72x1 devices 512 K of Flash main array and a 32K Shadow Block are available, on the MAC72x2 devices, up to 320K of Flash main array and a 32K Shadow Block are available.

The Shadow Block includes several locations which hold information on the state of the Flash main array and Shadow Block protection, access and security, as well as a security key to unlock the device, if enabled. The security state of the entire MCU is determined solely by the security settings stored in the Shadow Block. The device can only be unlocked by writing this security key or by using a JTAG Lockout Recovery mechanism to erase and very the contents of the non-volatile memory. Please refer to Section 18.7.9, “Flash Security” for more details.

Both the Flash main array and Shadow Block memories are provided with protection sectors to help eliminate errors caused by inadvertent program or erase operations being executed on the wrong area of the memory. The Flash main array has 6 protected sectors, with 2 sectors of 128K and 4 sectors of 16K. The Shadow Block has a single sector of 32K.

- Flash Main Array (New): 15x4K + 1x196K + 15x4K + 1x196K
- Shadow Block (New): 8x4K

In addition to the erase/program protection provided, both the Flash main array and Shadow Block may be protected against unintended accesses, based on the access type (read vs. write, supervisor vs. user mode and instruction fetch vs. data access). The Flash main array has 32 access protection sectors, 30 with 4K and 2 with the remaining 200K. The Shadow Block has 8 access protection sectors, each with 4K, and may not be protected against writes. This will restrict accesses to the defined sector to be only possible

when the ARM7 core is in Supervisor mode and/or when a load/store operation is being performed. Note that all eDMA transfers are done in Supervisor/Data mode.

For information on the location of the Flash main array and Shadow Block in the memory map, refer to Chapter 9, “Device Memory Map”.

Consult Chapter 19, “Hip7a Low-Cost Embedded Flash (H7Fb)” and Chapter 20, “SPP Flash Controller (PFLASH_H7Fb) Module” for information on the operation and control of the on-chip Flash implemented on the MAC7200 family of devices.

18.2 Flash Features

18.2.1 General Features

- READ and WRITE paths through AHB
- Configurable line buffers for performance tuning based on application

18.2.2 Main Array Features

- 512K / 320K Flash
- Typically stores Application Code and Data
- Memory Map: See Section 9.12, “Flash Main Array Memory Map”
- Shares a single AHB interface with the Shadow Block

18.2.3 Shadow Block Features

- 32K Flash
- Stores the Primary Boot Loader (PBL) and/or Parameter Data
- Memory Map: See Section 9.13, “Shadow Block Memory Map”
- Shares a single AHB interface with the main array

18.2.4 Flash Modes

- Flash User Mode
  - Read Operations
  - Write (Program) Operations
  - Erase Operation
  - Entering/Exiting Stop Mode

18.3 Flash Implementation

18.3.1 MAC72x1 Flash

The Flash configuration for the MAC72x1 is as follows:

- SFS = 0 (No special Flash selection)
- SIZE = 1 (512k)
- LAS = 4 (4*16k, 4*16k, 2*64k)
- MAS = 0 (2*128k)
- 32k Shadow Block

The Flash blocks and partitions are remapped to form the following order:

- 2*64k (Paritition 2: LAS-Blocks)
- 4*16k (Partition 0:)
- 4*16k (Partition 1:)
- 2*128k (Partition 3: MAS-Block 1, MAS-Block 0)
- 32k Shadow Block

Because of this remapping, the addresses of the lower half of each section will appear after the higher half (see Figure 18-1).

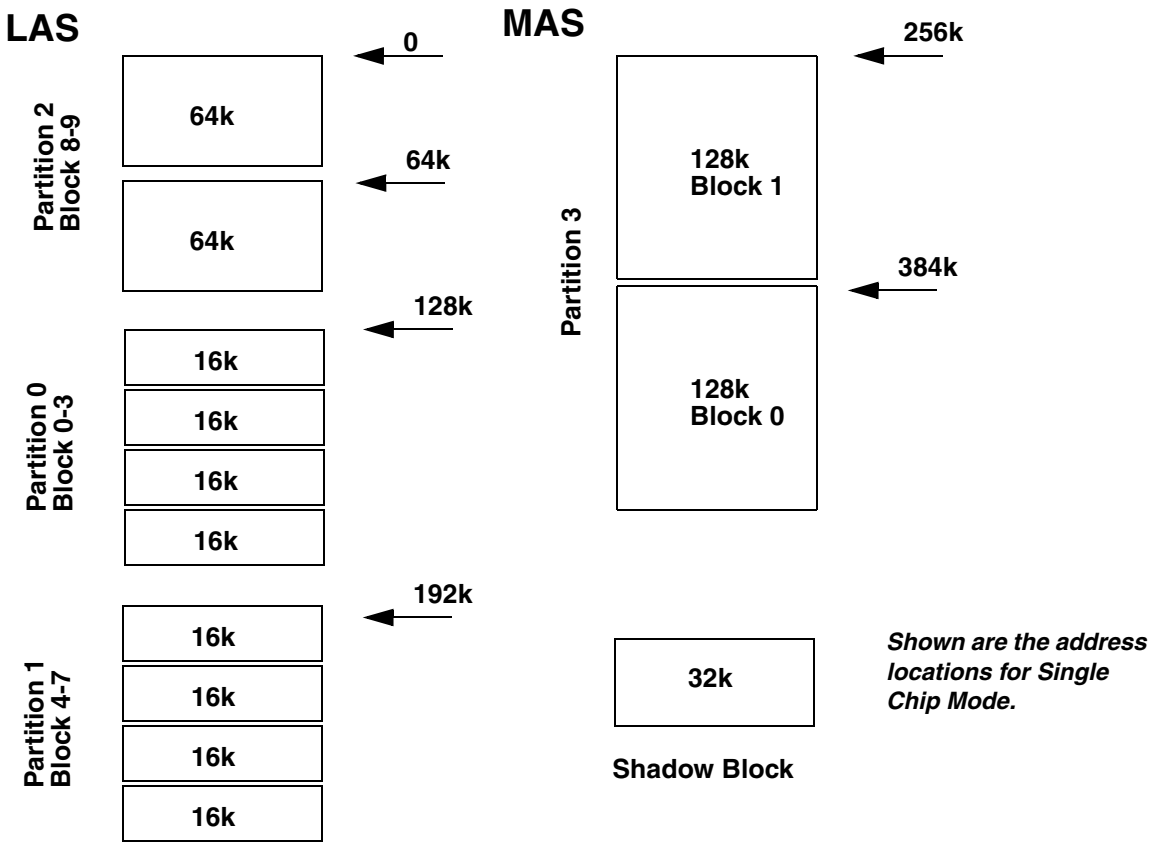


Figure 18-1. MAC72x1 Flash Blocks and Partitions

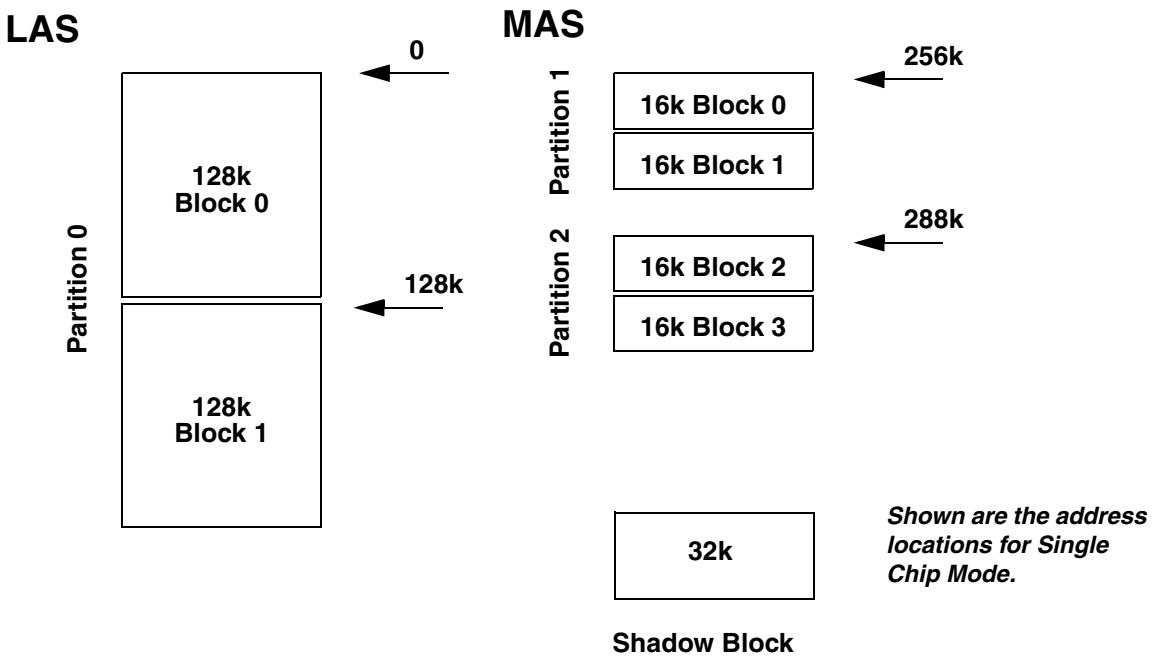


Figure 18-2. MAC72x2 Flash Blocks and Partitions

This remapping needs to be taken into account, when accessing the H7fb control registers. For example the MLOCK[3:0] field in the Low/Mid Address Space Block Locking Register, is mapped to the following address ranges (using SC mode):

- MLOCK[0] = address range 384k to 512k-1
- MLOCK[1] = address range 256k to 384k-1

The LLOCK[15:0] mapping for SC mode is:

- LLOCK[0] = address range 128k to 144k-1
- LLOCK[1] = address range 144k to 160k-1
- ...
- LLOCK[4] = address range 192k to 208k-1
- ...
- LLOCK[8]= address range 0 to 64k-1
- LLOCK[9]= address range 64kto 128k-1

18.3.2 MAC72x2 Flash

The Flash configuration for the MAC72x2 is as follows:

- SFS = 1 (Special Flash selected: 320k)
- SIZE = 1 (Normal size 512k, with SFS: 320k)

- LAS = 0
- MAS = 1

18.4 Flash External Pins

There are no Flash or Flash Controller signals that drive or are driven from MCU pins.

18.5 PFLASH Bus Aborts

18.5.1 IPI Register Interface

The H7Fb module does not support Peripheral Bus bus aborts, but this capability is implemented outside of the H7Fb block, to give the following:

Table 18-1. PFLASH Bus Aborts

Abort	Allowed
	$0000-$002f
$0030-$3fff	

Supervisor Access: Unused.

18.5.2 Flash Array Interface

In addition to Peripheral Bus bus aborts, the flash can also abort array interface (AHB for the MAC72xx) accesses, for a variety of reasons:

- Access generated an ECC error
- Access generated a Read-while-Write violation (e.g.-trying to read the same partition that is being programmed or erased)
- Aborting a program/erase high voltage operation

18.6 PFLASH Differences MAC72x2 from MAC71xx

- All accesses are through a single bus interface, residing on AXBS Slave Port 0
- Removed Peripheral Bus read/write interface
- Added ECC (1-bit correction, 2-bit detection)
- Completely different programming interface for the Flash
- Removed Configuration Field. Configuration data is now stored at the upper addresses of the Shadow Block.
- Program/Erase partitioning changed
  - Flash Main Array (Old): 15x4K + 2x196K + 15x4K
  - Flash Main Array (New - MAC72x2): 2x128K + 4x16K
  - Shadow Block (Old): 2x2K + 1x4K + 2x8K + 1x4K + 2x2K

  - Shadow Block (New - MAC72x2): 1x32K
- Access Protection partitioning changed
  - Write protection for DACC/SACC registers changed
  - Added WACC (write protect) for Flash main array only(Same partitioning as SACC/DACC)
  - Flash Main Array (Old): 15x4K + 2x196K + 15x4K
  - Flash Main Array (New): 15x4K + 1x196K + 15x4K + 1x196K
  - Shadow Block (Old): 2x2K + 1x4K + 2x8K + 1x4K + 2x2K
  - Shadow Block (New - MAC72x2): 8x4K
- Read-while-Write (RWW) partitioning changed
  - Flash Main Array (Old): 1x512K
  - Flash Main Array (New - MAC72x2): 128K + 128K + 2x16K + 2x16K
  - Shadow Block (Old): 1x32K
  - Shadow Block (New - MAC72x2): 1x32K
- Number of wait states increased
- Shadow Block base address changed to FLASH_BASE + $00F0 0000
- Security enforcement changed
  - Removed Backdoor Access functionality
  - Remove the KEYEN[1:0] bits from the Security Word
  - Renamed CFMSEC register to System Censor Word
- Memory map relocation (based on security/chip mode) is unchanged
- Lockout Recovery changed
  - Lockout Recovery now erases the Shadow Block as well
  - Must set "start lockout recovery" bit in SC4 with RESET asserted
  - No need to clear the "start lockout recovery" bit in SC4 - self clearing
  - No need to re-program security word - done automatically at the end
  - Must reset device at end of lockout recovery
  - No software lockout recovery (i.e.-from Expanded Mode)

18.7 PFLASH Application Usage

18.7.1 Flash Terminology

Table 18-2. Flash Terminology

Term	Definition
Access Protection	The ability to block accesses of a defined type (i.e.-read vs. write, user vs. supervisor, etc.) on a section of Flash memory. This differs from Erase/Program Protection.
Erase/Program Protection	The ability to erase or program a section of Flash memory. This differs from Access Protection.

Table 18-2. Flash Terminology

Term	Definition
Read-while-Write (RWW)	The ability to read code/data from one partition while erasing or programming another partition.
Block	The smallest unit to which Erase/Program Protection may be applied.
Partition	The smallest unit to which Read-while-Write may be applied.
Programmed	A bit is programmed when it is logical ‘0’
Erased	A bit is erased when it is logical ‘1’
Suspend	Suspends the current erase of a block so a read could be done within its partition.

18.7.2 Enabling the PFLASH

It is not necessary to enable the Flash controller before it can be used. However, it is generally a good idea to configure the correct settings in the Flash controller as soon as possible after boot for performance reasons.

18.7.3 Flash Array Memory Map

Please refer to Section 9.12, “Flash Main Array Memory Map” and Section 9.13, “Shadow Block Memory Map” for more details on the Flash Main Array and Shadow Block memory map.

18.7.4 Flash Registers

The Flash related registers are located in two places:

- Flash Block - Programming/Erase and BIU registers located within the Peripheral Bus space of the Flash block itself. The BIU registers are used by the Flash Controller for access protection and performance tuning.
- MCM - Flash Write ACCess (WACC) register located within the Peripheral Bus space of the MCM block.

18.7.4.1 Flash Block User Registers

These registers are located within the Peripheral Bus space of the Flash (at $FC0F 0000), and include program/erase related registers and BIU registers.

Table 18-3. Flash Block IPI User Registers

Register	Address Offset	Function	Reset Value
MCR	$0000	Module Configuration	$1101 3600
LML	$0004	LAS/MAS/Shadow Erase/Program Protection	From Shadow $7de8
HBL	$0008	HAS Erase/Program Protection	From Shadow $7df0
SLL	$000c	LAS/MAS Erase/Program Protection	From Shadow $7df8

Table 18-3. Flash Block IPI User Registers

Register	Address Offset	Function	Reset Value
LMS	$0010	LAS/MAS Erase/Program Block Select	$0000 0000
HBS	$0014	HAS Erase/Program Block Select	$0000 0000
ADR	$0018	Aborted Address	$0000 0000
BIU0	$001c	PFCR1 (Pre-fetch control, # wait states, etc.)	$0001 4a15
BIU1	$0020	PFAPR (Master access protection)	$0000 0007
BIU2	$0024	PFCR2 (Line Buffer Config + Shadow SACC/DACC)	From Shadow $7e00
BIU3	$0028	PFSACC (Main Array)	From Shadow $7e08
BIU4	$002c	PFDACC (Main Array)	From Shadow $7e10

18.7.4.1.1 PFCR1 - PFLASH Configuration Register 1

The **PFCR1** register contains line buffer and pipeline controls.

Bits	Function	Reset
[31:18]	**MxPFE** - Master x Pre-fetch Enable (not used)	$0000
[17]	**M1PFE** - DMA Master Pre-fetch Enable	0
[16]	**M0PFE** - Core Master Pre-fetch Enable	1
[15:13]	**APC** - Address Pipelining Control	010
[12:11]	**WWSC** - Write Wait State Control	01
[10:8]	**RWSC** - Read Wait State Control	010
[7]	Reserved for future use	0
[6]	**DPFEN** - Data Prefetch Enable	0
[5]	Reserved for future use	0
[4]	**IPFEN** - Instruction Prefetch Enable	1
[3]	Reserved for future use	0
[2:1]	**PFLIM** - Prefetch Limit	10
[0]	**BFEN** - Line Read Buffer Enable	1

The **PFCR1** register may be locked (i.e.-writes are not allowed) by clearing **PFCR2[23]**. In the case of any problems with the line buffers in the Flash controller (i.e.-functional bugs, etc.), the **BFEN** bit may be cleared, thus disabling the buffers and associated logic.

The following table describes the allowed values for each of the fields in the **PFCR1** register:

Table 18-4. Flash PFCR1 Register Settings

Field	PFCR1 Bits	Allowed Values
Reserved	31:18	-
M1PFE	17	{0,1}
M0PFE	16	{0,1}
APC	15:13	{RWSC-1,...,7}
WWSC	12:11	{1,3}
RWSC	10:8	{0,...,7}
Reserved	7	-
DPFEN	6	{0,1}
Reserved	5	-
IPFEN	4	{0,1}
Reserved	3	-
PFLIM	2:1	{0,1,2,3}
BFEN	0	{0,1}

NOTE

RWSC = 0 is allowed, however you must set APC = 0 and you must set the MCR[PRD] bit = 1. This PRD bit is a H7Fb configuration bit that is located in the H7Fb MCR register bit [7].

Example — APC and RWSC Valid Configurations

The following table shows the valid configurations of APC and RWSC given a 25MHz frequency and the rules above. All other configurations are invalid and will cause errors

APC	RWSC
{0,1,2,3,4,5,6,7}	0
{1,2,3,4,5,6,7}	1
{2,3,4,5,6,7}	2
{3,4,5,6,7}	3
{4,5,6,7}	4
{5,6,7}	5
{6,7}	6
7	7

Table 18-5 shows the settings for the APC and RWSC fields, given a few frequencies. These values were calculated using the following formulas:

$$APC = Int\left(\frac{\text{Total Addr Delay}}{\text{Period}}\right) \qquad \textit{Eqn. 18-1}$$

$$RWSC = Int\left(\frac{\text{Total Data Delay}}{\text{Period}}\right) \qquad \textit{Eqn. 18-2}$$

with the following parameters :

- H7fb access delay tacc = 29 ns
- H7fb pipelined read delay trr = 18 ns
- pflash_h7fb delay = 2 ns
- axbs hrdata delay = 1.5 ns
- bus master setup = 2.5 ns
- Total Data Delay = 35 ns
- Total Addr Delay = 24 ns

Table 18-5. Flash Controller APC and RWSC Settings

Frequency (MHz)	Period (ns)	APC	RWSC	PRD	Notes
20	50	0	0	1	See Note [a]
25	40	0	0	1	See Note [a]
30	33.33	0	0	1	See Note [a]
40	25	0	1	1	See Note [b]
50	20	1	1	1	See Note [a]
60	16.66	1	1	0	See Note [a]
70	14.286	1	2	0	See Note [b]

a. **APC** must be greater than or equal to **RWSC** at this frequency. If **APC** < **RWSC**, a pipelined read will return data before the previous read data has been acknowledged. Therefore, the previous read will return data from the next pipelined read

b. **APC** must be greater than or equal to **RWSC**-1 at this frequency. If **APC** < **RWSC**-1, a pipelined read will return data before the previous read data has been acknowledged. Therefore, the previous read will return data from the next pipelined read

18.7.4.1.2 PFAPR - PFLASH Access Protection Register

The **PFAPR** register contains access protection controls. The ARM7 core is designated M0, and the DMA is designated M1.

Bits	Function	Reset
[31:4]	**MxAP** - Master x Access protection (not used)	$0000000
[3:2]	**M1AP** - DMA Access protection	00
[1:0]	**M0AP** - Core Access protection	01

The reset value corresponds to the ARM7 core having read/write access, and the DMA not.

The **PFAPR[31:4]** register bits may not be locked. **PFAPR[3:2]** may be locked by clearing **PFCR2[17]**, and **PFAPR[1:0]** may be locked by clearing **PFCR2[16]**

18.7.4.1.3 PFCR2 - PFLASH Configuration Register 2

The **PFCR2** register contains register locking, Shadow SACC/DACC and line buffer configuration.

Bits	Function	Reset[a]
[31:30]	LBCFG - Line Buffer Configuration	11
[29:24]	Reserved for future use	11111
[23]	CR1UNL - Enable Writes to the PFCR1 register	1
[22]	SACCUNL - Enable Writes to the PFSACC register	1
[21]	DACCUNL - Enable Writes to the PFDACC register	1
[20:18]	Reserved for future use	111
[17:16]	MPAUNL - Enable Writes to the M0AP/M1AP bits in the PFAPR [17] - M1AP [16] - M0AP	 1 1
[15:8]	SHSACC - Shadow Supervisor/User Access 0 = Supervisor only allowed 1 = Supervisor or User allowed	$ff
[7:0]	SHDACC - Shadow Data/Instruction Access 0 = Data only allowed 1 = Data or Instruction allowed	$ff

a. The reset value is determined by the state of the Flash. The reset values shown are for a fully erased Shadow Block.

The **PFCR2** register is not lockable.

18.7.4.1.4 PFSACC - PFLASH Supervisor/user ACCess

The **PFSACC** register contains the SACC for the main array.

Bits	Function	Reset[a]
[31:0]	**SACC** - Main Array Supervisor/User Access 0 = Supervisor only allowed 1 = Supervisor or User allowed	$ffff ffff

a. The reset value is determined by the state of the Flash. The reset values shown are for a fully erased Shadow Block.

The **PFSACC** register may be locked (i.e., writes are not allowed) by clearing **PFCR2[22]**.

18.7.4.1.5 PFDACC - PFLASH Data/instruction ACCess

The **PFDACC** register contains the SACC for the main array.

Bits	Function	Reset[a]
[31:0]	**DACC** - Main Array Data/Instruction Access 0 = Data only allowed 1 = Data or Instruction allowed	$ffff ffff

a. The reset value is determined by the state of the Flash. The reset values shown are for a fully erased Shadow Block.

The **PFDACC** register may be locked (i.e., writes are not allowed) by clearing **PFCR2[21]**.

18.7.4.2 Flash MCM Registers

These registers are located within the Peripheral Bus space of the MCM (at $FC04 0000), and include the Write ACCess protection for the main array.

Table 18-6. MCM Block Flash Registers

Register	Function	Reset Value
MUDCR	PFWACC (Write ACCess Protection)	$0000 0000

18.7.4.2.1 PFWACC - PFLASH Write ACCess

The **PFWACC** register contains the WACC for the main array. There is no corresponding functionality for the Shadow Block.

Bits	Function	Reset
[31:0]	**WACC** - Main Array Write Access 0 = Writes allowed 1 = Writes not allowed	$0000 0000

The **PFWACC** register may not be locked.

18.7.5 Flash Access Protection

Access Protection refers to the ability to block accesses of a defined type (i.e., read vs. write, user vs. supervisor, instruction vs. data) on a section of Flash memory.

The Shadow Block has two types of access protection (mapping is shown in Table 18-8):

Table 18-7. Flash Shadow Block Access Protection Types

Field	Definition
SHSACC	0 Supervisor mode only 1 Supervisor and User mode allowed
SHDACC	0 Data fetch only 1 Instruction and Data fetch allowed

Table 18-8. Flash Shadow Block Access Protection Address Ranges

Address Offset	Size	SHSACC	SHDACC
$0000 0000 $0000 0FFF	4K	SHSACC[0]	SHDACC[0]
$0000 1000 $0000 1FFF	4K	SHSACC[1]	SHDACC[1]
$0000 2000 $0000 2FFF	4K	SHSACC[2]	SHDACC[2]
$0000 3000 $0000 3FFF	4K	SHSACC[3]	SHDACC[3]
$0000 4000 $0000 4FFF	4K	SHSACC[4]	SHDACC[4]
$0000 5000 $0000 5FFF	4K	SHSACC[5]	SHDACC[5]
$0000 6000 $0000 6FFF	4K	SHSACC[6]	SHDACC[6]
$0000 7000 $0000 7FFF	4K	SHSACC[7]	SHDACC[7]

The main array has three types of access protection (Mapping is shown in Table 18-10):

Table 18-9. Flash Shadow Block Access Protection Types

Field	Definition
SACC	0 Supervisor mode only 1 Supervisor and User mode allowed
DACC	0 Data fetch only 1 Instruction and Data fetch allowed
WACC	0 Reads only 1 Reads and Writes allowed

Table 18-10. Flash Main Array Access Protection

	Address Offset	Size	SACC	DACC	WACC
Low Address Space (LAS)	$0000 0000 $0000 0FFF	4K	SACC[0]	DACC[0]	WACC[0]
	$0000 1000 $0000 1FFF	4K	SACC[1]	DACC[1]	WACC[1]
	$0000 2000 $0000 2FFF	4K	SACC[2]	DACC[2]	WACC[2]
	$0000 3000 $0000 3FFF	4K	SACC[3]	DACC[3]	WACC[3]
	$0000 4000 $0000 4FFF	4K	SACC[4]	DACC[4]	WACC[4]
	$0000 5000 $0000 5FFF	4K	SACC[5]	DACC[5]	WACC[5]
	$0000 6000 $0000 6FFF	4K	SACC[6]	DACC[6]	WACC[6]
	$0000 7000 $0000 7FFF	4K	SACC[7]	DACC[7]	WACC[7]
	$0000 8000 $0000 8FFF	4K	SACC[8]	DACC[8]	WACC[8]
	$0000 9000 $0000 9FFF	4K	SACC[9]	DACC[9]	WACC[9]
	$0000 A000 $0000 AFFF	4K	SACC[10]	DACC[10]	WACC[10]
	$0000 B000 $0000 BFFF	4K	SACC[11]	DACC[11]	WACC[11]
	$0000 C000 $0000 CFFF	4K	SACC[12]	DACC[12]	WACC[12]
	$0000 D000 $0000 DFFF	4K	SACC[13]	DACC[13]	WACC[13]
	$0000 E000 $0000 EFFF	4K	SACC[14]	DACC[14]	WACC[14]
	$0000 F000 $0003 FFFF	196K	SACC[15]	DACC[15]	WACC[15]

Table 18-10. Flash Main Array Access Protection (Continued)

	Address Offset	Size	SACC	DACC	WACC
Mid Address Space (MAS)	$0004 0000 $0004 0FFF	4K	SACC[16]	DACC[16]	WACC[16]
	$0004 1000 $0004 1FFF	4K	SACC[17]	DACC[17]	WACC[17]
	$0004 2000 $0004 2FFF	4K	SACC[18]	DACC[18]	WACC[18]
	$0004 3000 $0004 3FFF	4K	SACC[19]	DACC[19]	WACC[19]
	$0004 4000 $0004 4FFF	4K	SACC[20]	DACC[20]	WACC[20]
	$0004 5000 $0004 5FFF	4K	SACC[21]	DACC[21]	WACC[21]
	$0004 6000 $0004 6FFF	4K	SACC[22]	DACC[22]	WACC[22]
	$0004 7000 $0004 7FFF	4K	SACC[23]	DACC[23]	WACC[23]
	$0004 8000 $0004 8FFF	4K	SACC[24]	DACC[24]	WACC[24]
	$0004 9000 $0004 9FFF	4K	SACC[25]	DACC[25]	WACC[25]
	$0004 A000 $0004 AFFF	4K	SACC[26]	DACC[26]	WACC[26]
	$0004 B000 $0004 BFFF	4K	SACC[27]	DACC[27]	WACC[27]
	$0004 C000 $0004 CFFF	4K	SACC[28]	DACC[28]	WACC[28]
	$0004 D000 $0004 DFFF	4K	SACC[29]	DACC[29]	WACC[29]
	$0004 E000 $0004 EFFF	4K	SACC[30]	DACC[30]	WACC[30]
	$0004 F000 $0007 FFFF	196K	SACC[31]	DACC[31]	WACC[31]

Due to the access protection, there is a possible deadlock situation that the user can find themselves in as follows: If the DACC is programmed to only allow data fetches on Partition 0 ($0000 0000 to $0000 0FFF), and the device is reset, the user will generally be unable to boot the device from the Flash main array.

1. Boot from the Shadow Block, and enter debug mode. For this to work, the SDACC[0] must be cleared (i.e.-Instruction fetches allowed).
2. Boot from the external bus (if available).
3. Perform a JTAG Lockout Recovery sequence, which will erase the DACC.

18.7.6 Flash Program/Erase Protection and Selection

Program and Erase protection refers to the ability to protect flash blocks against program and erase operations, or to program/erase only particular blocks. Table 18-11 details the block mapping for the MAC72x2 Flash.

Table 18-11. Flash Program/Erase Blocks - MAC72x2

Address Offset	Block Number	Size	LML/HBL[a]	LMS/HBS
$0000 0000 $0001 FFFF	LAS #0	128K (Main Array)	LLOCK[0]	LSEL[0]
$0002 0000 $0003 FFFF	LAS #1	128K (Main Array)	LLOCK[1]	LSEL[1]
$0004 0000 $0004 3FFF	MAS #0	16K (Main Array)	MLOCK[0]	MSEL[0]
$0004 4000 $0004 7FFF	MAS #1	16K (Main Array)	MLOCK[1]	MSEL[1]
$0004 8000 $0004 BFFF	MAS #2	16K (Main Array)	MLOCK[2]	MSEL[2]
$0004 C000 $0004 FFFF	MAS #3	16K (Main Array)	MLOCK[3]	MSEL[3]
$00F0 0000 $00F0 7FFF	n/a	32K (Shadow Block)	SLOCK	PEAS[b]

a. Similar mapping also applies to **SLL** register.

b. The **PEAS** field is located in the **MCR** register

Table 18-12 details the block mapping for the MAC72x1 Flash.

Table 18-12. Flash Program/Erase Blocks - MAC72x1

Address Offset	Block Number	Size	LML/HBL[a]	LMS/HBS
$0000 0000 $0000 FFFF	LAS #8 Partition 2	64K (Main Array)	LLOCK[8]	LSEL[8]
$0001 0000 $0001 FFFF	LAS #9 Partition 2	64K (Main Array)	LLOCK9]	LSEL[9]
$0002 0000 $0002 3FFF	LAS #0 Partition 0	16K (Main Array)	LLOCK[0]	LSEL[0]
$0002 4000 $0002 7FFF	LAS #1 Partition 0	16K (Main Array)	LLOCK[1]	LSEL[1]
$0002 8000 $0002 BFFF	LAS #2 Partition 0	16K (Main Array)	LLOCK[2]	LSEL[2]
$0002 C000 $0002 FFFF	LAS #3 Partition 0	16K (Main Array)	LLOCK[3]	LSEL[3]
$0003 0000 $0003 3FFF	LAS #4 Partition 1	16K (Main Array)	LLOCK[4]	LSEL[4]
$0003 4000 $0003 7FFF	LAS #5 Partition 1	16K (Main Array)	LLOCK[5]	LSEL[5]
$0003 8000 $0003 BFFF	LAS #6 Partition 1	16K (Main Array)	LLOCK[6]	LSEL[6]
$0003 C000 $0003 FFFF	LAS #7 Partition 1	16K (Main Array)	LLOCK[7]	LSEL[7]
$0004 0000 $0005 FFFF	MAS #1 Partition 3	128K (Main Array)	MLOCK[1]	MSEL[1]
$0006 0000 $0007 FFFF	MAS #0 Partition 3	128K (Main Array)	MLOCK[0]	MSEL[0]
$00F0 0000 $00F0 7FFF	n/a	32K (Shadow Block)	SLOCK	PEAS[b]

a. Similar mapping also applies to **SLL** register.

b. The **PEAS** field is located in the **MCR** register

18.7.7 Flash Programming Word Size

Due to the use of ECC on a 64-bit word in the Flash array with the 32-bit ARM core, locations in the flash (both main array and Shadow Block) must be programmed in a special manner.

As an example, to program the 64-bit Double Word at address $0000 0000, you must perform the following sequence of accesses in order:

1. 32-bit WRITE to address $0000 0000
2. 32-bit WRITE to address $0000 0004

Even if the data being stored is only 32-bits, you must execute the second 32-bit WRITE in order to perform the actual write into the Flash array.

18.7.8 Read-while-Write (RWW)

Read-while-Write (RWW) refers to the ability to execute code from one partition while programming or erasing another partition. For the definition of the partitions, please refer to Section 9.12, “Flash Main Array Memory Map”. While doing a program/erase on one block in a partition, you cannot read from another block within the partition, unless you do a suspend, and you cannot read from the block that is being programmed/erased; however, you can do normal reads from blocks in other partitions.

18.7.9 Flash Security

The goal of total flash security is simple:

- Prevent any data from a secured flash from being read out of the MAC72xx in any way, shape or form.
- Allow complete access to the Flash when it is unsecured

On the MAC72xx, the security is determined by the state of the System Censor word (Shadow Block address offset $7de0).

On the MAC72xx, the System Censor word in a similar manner to the **CFMSEC** register in the MAC71xx CFM Flash module, as follows:

Table 18-13. System Censor Word Definition

shad_out[31:0]	Function
[31:30]	**KEYEN**: 10 = Backdoor access enabled (Currently not used)
[29:2]	Reserved for future use
[1:0]	**SEC**: 10 = unsecured, all other values = secured

18.7.9.1 Securing the device

In order to secure the device, you must perform the following steps:

1. Program bits [1:0] of the System Censor word (in the Shadow Block at address offset $7de0) to 00 or 11.

2. Reset the device. The new security state does not take effect until after the device has been reset.

18.7.9.2 Unsecuring the device

In order to secure the device, you must perform the following steps:

1. Program bits [1:0] of the System Censor word (in the Shadow Block at address offset $7de0) to 10.
2. Reset the device. The new security state does not take effect until after the device has been reset.
3. If it is not possible to program the System Censor word due to the security of the system, then you must use the Lockout Recovery procedure to unsecure the device.

CAUTION

This procedure will erase the entire contents of both the Flash Main Array and Shadow Block. Please refer to Section 20.3, “BAM Protocol” for details on performing the Lockout Recovery procedure.

18.7.10 Flash Timing

Please refer to Appendix A, “Electrical Characteristics” for information on program and erase times.

Chapter 19 SRAM and SRAM Controller

19.1 SRAM

19.1.1 SRAM Features

- Single port RAM with chip enable/select and 1 byte write enable per byte, used for system RAM
- Memory Map: 20KByte block on a 32KByte boundary (MAC72x2) or 32kByte block(MAC72x1), byte addressable
- Start Address determined by 3-bit base address of slave port in the crossbar switch. Within the 512Mb space for each crossbar switch slave port, the SRAM is replicated *n* times, to fill the entire space. This allows the user to access the SRAM anywhere in the 512Mb space on a 32Kb boundary.
- Built in ECC (2-bit detection)

19.1.2 SRAM Protocol

There is no protocol associated with the SRAM controller other than the standard AHB-Lite v2.0.

19.1.3 SRAM External Pins

There are no SRAM signals that drive or are driven from MCU pins.

19.1.4 SRAM Bus Aborts

There are two situations in which access to the SRAM can be aborted:

- On the MAC72x2, the access results in an ECC Error. Please refer to Section 19.2.3.1, “Error Correcting Code (ECC) for more details on handling this type of abort.
- The access is between SRAM_BASE_ADDRESS + $5000 and SRAM_BASE_ADDRESS + $7FFF. i.e.-between the 20K and 32K address range of the SRAM. Note that above 32K, the address range is mirrored (Please see Section 19.1.6.2, “SRAM Address Mirroring).

19.1.5 SRAM Differences from MAC71xx

- On the MAC72x2, changed memory size from 32K to 20K. Memory map wrap-around is 32K, not 20K.
- Added ECC capability. See Chapter 13, “Miscellaneous Control Module (MCM) for more details on the ECC reporting on the MAC72x2.

19.1.6 SRAM Application Usage

19.1.6.1 SRAM Initialization

In order to avoid ECC errors on an uninitialized array, the SRAM array must be initialized using only 32-bit writes. This is because all RAM is in an indeterminate state after power up and all 8 and 16-bit writes are actually performed as 32-bit READ + MODIFY + WRITE in order to correctly calculate the ECC syndrome (which is based on a 32-bit value). Other than this, no other special handling is required.

19.1.6.2 SRAM Address Mirroring

Within the 512MB address space that the SRAM occupies on the crossbar switch, the actual addressable range of the SRAM is mirrored in the memory map on a 32K boundary, as follows:

Table 19-1. SRAM Address Mirroring

Address	Maps to....
$4000 0000	$4000 0000
$4000 8000	$4000 0000
$4001 0000	$4000 0000
:	:
:	:
$5FFF 0000	$4000 0000
$5FFF 8000	$4000 0000

This means that accessing any mirrored SRAM address is *virtually* identical to accessing the SRAM at its base address of $4000 0000. However, it should be noted that there is a posted write capability built into the SRAM controller. As a consequence, writing/reading to/from address $0000 0000 (for example) is not exactly equivalent (from a hardware point of view) to writing/reading to/from address $0000 8000 (for example). The intended use of this mirroring is to allow the user to select a base address of their choosing, but not to allow changing of this base address during the execution of an application. The default base address for the SRAM is $4000 0000 and is not selectable during reset, but is software selectable after reset.

19.2 Platform RAM Array Controller (PRAM_CTL)

19.2.1 Introduction

The PRAM_CTL module is an asynchronous reset, platform RAM array controller, with integrated error detection and correction. Shown below in Figure 19-1 is a simplified block diagram of how the PRAM_CTL would be implemented a typical platform:

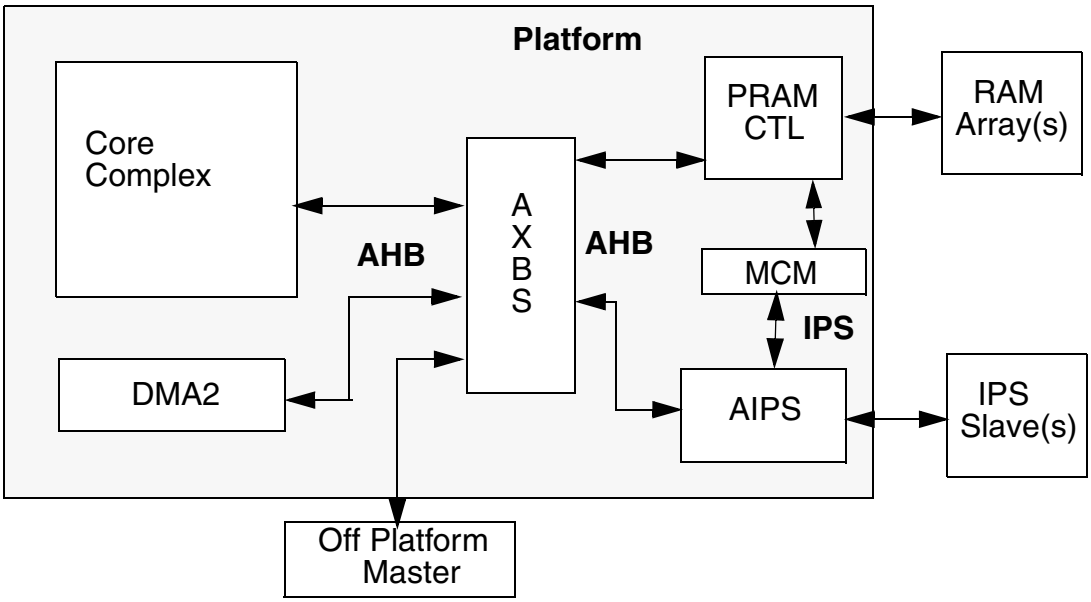
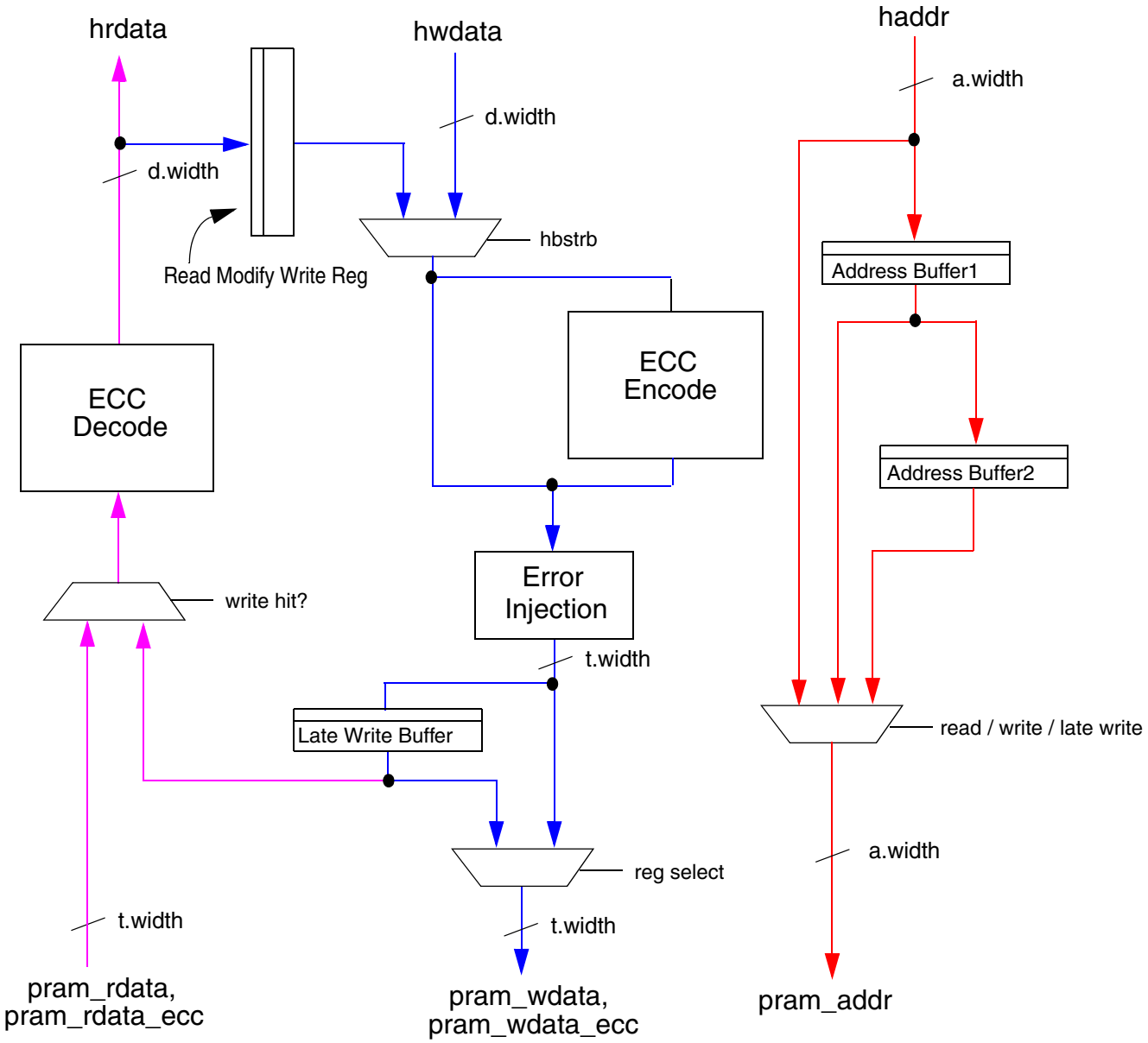


Figure 19-1. Simplified Platform Block Diagram

In Figure 19-1, the platform bus is the AMBA AHB, the AXBS is the AHB crossbar switch (*arbiter*), and the MCM is the Miscellaneous Control Module where error detection information is stored. Refer to the appropriate chapter for a complete description. Contained in this document are descriptions of: interfaces to the AHB, MCM and RAM array; the ECC (*Error Correcting Code*) algorithm and functionality; block timing constraints; timing diagrams illustrating functionality and perfomance; and overall block functionality descriptions. Shown in Figure 19-2 is a block diagram illustrating the 32 bit AHB to 17/18 bit PRAM address path and the 64/32 bit AHB to 78/39 bit PRAM data path through the module.



Note:

1 Red : Address Path
2 Blue : Write Data Path
3 Magenta : Read Data Path

Figure 19-2. Block Diagram: PRAM_CTL

Figure 19-2 above shows the address and data paths throughout the PRAM_CTL. *d.width* above represents the parameterizable data width of the module which can be set to values of 64 or 32. *t.width* (total width) and *a.width* (address width) are derived from the data width parameter. *t.width* is a combination of the data width and 7 ECC bits for each 32 bits of data. *a.width* has a value of 17 bits for a 64 bit data path and 18 bits for a 32 bit data path.

19.2.2 PRAM_CTL Interface Description

19.2.2.1 Overview

Table 19-3 lists all block inputs and outputs. Each signal has a corresponding bit width (**Width**) and timing constraint (**Timing**). Input timing constraints are defined as the percentage of the clock cycle, or delay in nanoseconds, before the input is valid. Output timing constraints are defined as the percentage of the clock cycle, or delay in nanoseconds, after the output is valid. These delays are specified in the form:

$$\text{Input Constraint} = \text{Clk\_Period} \times \text{Relative\_Input\_Delay} \qquad \textit{Eqn. 19-1}$$

$$\text{Output Constraint} = \text{Clk\_Period} \times (1 - \text{Relative\_Output\_Delay}) \qquad \textit{Eqn. 19-2}$$

where **Clk_Period** refers to the period in nanoseconds, **Relative_Input_Delay** refers to the amount of time before the input is valid at the interface, and **Relative_Output_Delay** refers to the amount of time before the output is valid at the interface. Note that timing associated with *pram** signals are not related to the system clock and therefore static.

Table 19-2 lists the width variables used in Table 19-3 and their corresponding values depending on the data width implementation.

Table 19-2. Signal Width Variables

Name	32 bit	64 bit
d.width	32	64
a.width	18	17
ecc.width	7	14
cs.width	1	2

Table 19-3. Signal Properties

Name	Port	Function	Width	Timing
hreset_b	input	asynchronous platform reset, active-low	1	See Note [1]
hsel_pram	input	select for pram slave	1	Clk_Period × 0.60
htrans	input	AHB type of transaction	2	Clk_Period × 0.60
haddr	input	AHB address bus	32	Clk_Period × 0.60
hwdata	input	AHB write data	d.width	Clk_Period × 0.45
hsize	input	AHB size of transfer	2	Clk_Period × 0.60
hbstrb	input	AHB byte strobes	d.width / 8	Clk_Period × 0.60
hwdata	input	AHB write data	d.width	Clk_Period × 0.60
hburst	input	AHB burst transaction type	3	Clk_Period × 0.60
hmaster	input	AHB master address	4	Clk_Period × 0.60
hunalign	input	AHB unalign transfer flag	1	Clk_Period × 0.60

Table 19-3. Signal Properties (Continued)

Name	Port	Function	Width	Timing
pram_rdata	input	read data from array to controller	d.width	
pram_rdata_ecc	input	ecc bits for read data from array	ecc.width	
pram_size	input	static size of RAM array	4	0ns (static)
pram_max_addr	input	maximum addressable memory space	18	0ns (static)
mcm_con_1bi	input	force continuous single bit correctable errors	1	Clk_Period × 0.2
mcm_one_1bi	input	force one single bit correctable error	1	Clk_Period × 0.2
mcm_con_nci	input	force continous multiple bit non-correctable errors	1	Clk_Period × 0.2
mcm_one_nci	input	force one multiple bit non-correctable error	1	Clk_Period × 0.2
mcm_errbit	input	bit position to inject error	7	Clk_Period × 0.2
hclk	input	platform system clock	1	N/A
hresp	output	AHB transfer response	3	Clk_Period × (1 - 0.3)
hready	output	AHB acknowledge	1	Clk_Period × (1 - 0.3)
hrdata	output	AHB read data bus	d.width	Clk_Period × (1 - 0.3)
pram_addr	output	array address	a.width	
pram_wdata	output	write data to array	d.width	
pram_wdata_ecc	output	write data ecc bits to array	ecc.width	
pram_cs_b	output	chip select active low	cs.width	
pram_we	output	write enable, high equals write	1	
ecc_single_error	output	single error found flag	cs.width	Clk_Period × (1 - 0.6)
ecc_multi_error	output	multiple errors found flag	cs.width	Clk_Period × (1 - 0.6)
ecc_syndrome	output	bit position of error	cs.width	Clk_Period × (1 - 0.6)
ecc_parity	output	overall parity of data decoded	ecc.width - cs.width	Clk_Period × (1 - 0.6)
ecc_haddr	output	address of error	32	Clk_Period × (1 - 0.6)
ecc_hmaster	output	master initiating errored transfer	4	Clk_Period × (1 - 0.6)
ecc_hsize	output	size of errored transfer	2	Clk_Period × (1 - 0.6)
ecc_hprot	output	protection of errored transfer	8	Clk_Period × (1 - 0.6)
ecc_hwrite	output	direction of errored transfer	1	Clk_Period × (1 - 0.6)

1. See Appendix A, “Electrical Characteristics” for signal timing constraints.

19.2.2.2 Detailed Signal Descriptions

Signals beginning with "h" correspond to the AMBA AHB bus. Their description can be found in the AMBA AHB-Lite 2.v6 specification. The signal descriptions below are provided to describe the

differences in functionality between the 2.v6 spec and the PRAM implementation. Signals which are exclusive to this block and not sufficiently described in Table 19-3 are further defined as well.

See Section 19.2.3.2, “Max Address,” for a more detailed description.

Table 19-4. Signal Property Details

Name	Description
htrans	A state of BUSY is not supported and furthermore is regarded as IDLE 01 IDLE - no transfer required 01 BUSY - **not supported** 01 NSEQ - first transfer of burst or single transfer 11 SEQL - burst transfer
pram_size	Supported power of 2 RAM array sizes are: 0111 32KB 1000 64KB 1001 128KB 1010 256KB 1011 512KB 1100 1024KB
pram_max_addr	This input defines the maximum address of the attached memory array. This functionality allows non-power-of-2 memories to be supported. See Section 19.2.3.2, “Max Address,” for a more detailed description.
mcm_con_1bi	When asserted, one bit of write data is inverted continously on each active pram write cycle. This inserts a single correctable error is inserted at these locations.
mcm_one_1bi	When asserted, one bit of write data is inverted, one time only, on the next active pram write cycle. This inserts a single correctable error at this location.
mcm_con_nci	When asserted, one bit of write data is inverted continously on each active pram write cycle. In addition, the overall parity bit associated with the active bank (pram_wdata_ecc[0]) is inverted. This inserts a multiple bit non correctable error at these locations.
mcm_one_nci	When asserted, one bit of write data is inverted continously on each active pram write cycle. In addition, the overall parity bit associated with the active bank (pram_wdata_ecc[0]) is inverted. This inserts a multiple bit non correctable error at this location.

Table 19-4. Signal Property Details (Continued)

Name	Description
mcm_errbit	This vector defines the bit position which is logically complemented to create the specific data inversion on a write operation. It is defined as: 00 pram_wdata[0] of the odd bank is inverted 01 pram_wdata[1] of the odd bank is inverted ... 31 pram_wdata[31] of the odd bank is inverted 32 pram_wdata[0] of the even bank is inverted 33 pram_wdata[1] of the even bank is inverted ... 63 pram_wdata[31] of the even bank is inverted 64 pram_wdata_ecc[0] of the odd bank is inverted 65 pram_wdata_ecc[1] of the odd bank is inverted 66 pram_wdata_ecc[2] of the odd bank is inverted 67 pram_wdata_ecc[3] of the odd bank is inverted 68 pram_wdata_ecc[4] of the odd bank is inverted 69 pram_wdata_ecc[5] of the odd bank is inverted 70 pram_wdata_ecc[6] of the odd bank is inverted 71 pram_wdata_ecc[0] of the even bank is inverted 72 pram_wdata_ecc[1] of the even bank is inverted 73 pram_wdata_ecc[2] of the even bank is inverted 74 pram_wdata_ecc[3] of the even bank is inverted 75 pram_wdata_ecc[4] of the even bank is inverted 76 pram_wdata_ecc[5] of the even bank is inverted 77 pram_wdata_ecc[6] of the even bank is inverted All mcm_errbit unused values are reserved and unused. For the 32bit operation, values of mcm_errbit from 32-64 and 71-77 are also reserved and unused.
hresp	The two msb of this field are tied to 0 allowing for the following supported encodings: 000 OKAY 001 ERROR which is asserted for the required two cycle response to signal an ecc_multiple_error assertion or violation of the pram_max_addr field. nnn All other encodings are reserved and never generated by the PRAM controller.
pram_cs_b	This is the active-low chip select. For the 64 bit implementation, there are 2 chip selects, each of which controls a 39 bit wide memory. The chip selects for this implementation are described in Table 19-5. For each chip select: 0 an active, qualified memory array cycle 1 an idle memory array cycle **Table 19-5. *pram_cs_b* 64 Bit Behavior** Acess Type | haddr[2] | pram_cs_b[1] | pram_cs_b[0] | Description IDLE | — | 1 | 1 | No access ≤ 32 bit | 0 | 0 | 1 | Even bank accessed ≥ 32 bit | 1 | 1 | 0 | Odd bank accessed > 32 bit | — | 0 | 0 | Both banks accessed

Table 19-4. Signal Property Details (Continued)

Name	Description
pram_we	This is the read or write flag. 0 read operation 1 write operation
ecc_single_error	Flag to indicate a single error correction has occurred. This signal is only valid when *hready* is asserted.
ecc_multi_error	Flag to indicate a multiple-bit, non-correctable error has occurred. This signal starts the ERROR response on the AHB. ecc_syndrome, ecc_address, and ecc_hmaster information is valid when this flag and *hready* are asserted. A multiple-bit error is defined as a non-zero syndrome and odd ecc_parity, or even ecc_parity and an out of range syndrome.
ecc_syndrome	There is a 6bit syndrome field generated for each 32 bits of data width. The syndrome can be used to determine the bit position of the errored bit within each of the 39 bits of data read from the array. See the table of ECC bits vs. data bits to determine which syndrome relates to which data bit.
ecc_parity	This is the overall parity of the data decoded, based on an odd parity scheme.
ecc_<attribute>	This refers to the final four entries in Table 19-2. These signals reflect the state of the AHB signals with corresponding attribute name in the address phase of the AHB transfer. For example, *ecc_hprot* reflects the AHB HPROT state when this transfer was requested.

The memory map for the PRAM memory space is determined by the system in which the controller is implemented. There are no program-visible registers within the controller. All ECC error reporting and forcing capabilities are implemented via program-visible registers located in the Miscellaneos Control Module (MCM).

19.2.3 Functional Description

19.2.3.1 Error Correcting Code (ECC)

19.2.3.1.1 Overview

The PRAM_CTL memory controller uses a 7-bit error correcting code (ECC) to correct single-bit errors and detect multiple-bit errors within the 32-bit data field. For the 64-bit implementation, the 32-bit encode and decode is effectively duplicated to work on 64 bits for a total of 14 bits of ECC. Since the ECC bits are written to and read from memory with the data, it is essential that each 32-bit memory location be written to a known value before being read. If an uninitialized memory address is read, it is likely the read will result in a multiple-bit ECC error and an errored transaction on the AHB. Only 32-bit and 64-bit write transfers can be used to initialize the memory as 16-bit or 8-bit writes will generate a read/modify/write scenario (see Section 19.2.3.5.2, “Less than 32-bit Writes”). As shown highlighted in blue in Figure 19-2, the ECC is composed of two modules, the encoder and the decoder. The decode module is shown in greater detail in Figure 19-3 below.

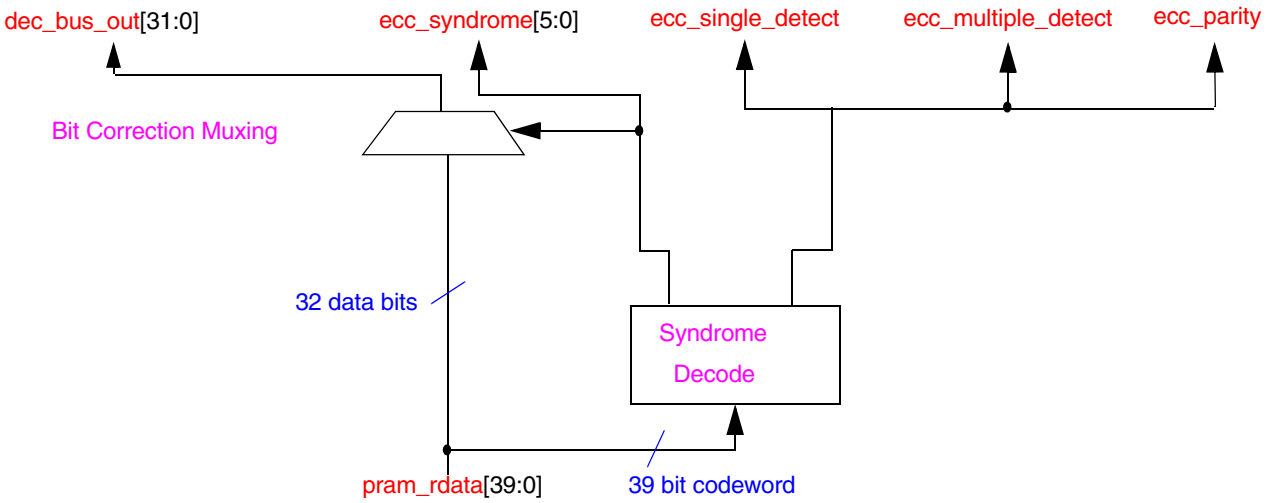


Figure 19-3. Block Diagram: 39-bit ECC Decode

The encoder logic is included in the write data path to the RAM and generates the 7-bit ECC code which is written to the RAM with the data. Together, the data bus and ECC bits are called the *code word*. Six of the ECC bits are calculated using a modified Hamming code. The seventh and final ECC bit is calculated so that the parity of the entire code word is odd. Again, this process is done twice in parallel for the 64-bit implementation.

The decoder logic is included in the read data path from the RAM back to the AHB bus. The decoder uses the same algorithm as the encoder and generates 6 unique ECC bits, or the *syndrome*[5:0], from the 39-bit code word (7 ECC and 32 data bits). This syndrome is used to determine which bit within the 39-bit data bus caused the single-bit correctable error. See Chapter 13, “Miscellaneous Control Module (MCM)” for more detail on which syndromes map to which bits. A seventh ECC bit representing the overall parity of the entire code word is also generated.

Table 19-6 shows the relationship of the overall parity bit, the syndrome and error detection and correction. Odd parity is shown as a value of 1. In this table, the term "valid codeword" refers to the codewords defined in Table 19-11. All other codewords are "invalid."

Table 19-6. Parity vs. Syndrome

Overall Parity	Syndrome[5:0]	Result
0	0	single-bit correction
0	valid codeword	single-bit correction
0	invalid codeword	multiple-bit error
1	non-zero	multiple-bit error
1	0	no error

It is important to note that the first entry in this table, overall even parity and a syndrome of zero, could be generated by the failure of the either the 7th ECC bit or the entire 39 bits of data. For this reason, an odd parity scheme was chosen for the 7th ECC bit, and a check is performed to generate a mulitple bit error when the entire 39-bit codeword is zero.

Single-bit corrections cause the output *ecc_single_error* to be asserted. Multiple-bit errors cause the *ecc_multi_error* output to be asserted as well as an AHB ERROR response to be generated (*see AMBA AHB spec 2.0, pg 3-23*). AHB address attributes are also saved and outputted for each case. These two flags are mutually exclusive.

The assertion of these two flags may cause an interrupt to be generated, the EER bit to be set in the Miscellaneous Control Module (*MCM*), and the errored address, syndrome, parity and master address to be saved in the MCM (See Chapter 13, "Miscellaneous Control Module (MCM) for a programming model). Conclusive research has not been done to investigate the effect of more than 2 errors, although the third and fourth entry of Table 19-6 have been put in place to catch some of these.

Finally, note that for the 64-bit implementation, if the read access size is less than 64 bits, only ECC errors within the 39-bit code word that is applicable to the read can cause a bus error. However, all ECC errors will still be reported.

19.2.3.2 Max Address

The input *pram_max_addr*[17:0] is static and reflects the maximum 32-bit data address (eg. *pram_max_addr*[17:0] is the equivalent notation of *haddr*[19:2]) available for use in the attached memory. This funtionality allows for non power of 2 memory arrays to be used. AHB requests to addresses above this value will be terminated with an ERROR response. This input should be used with the input *pram_size*[3:0] in the following way.

To indicate the memory array size, *pram_max_addr*[17:0] should be set to the maximum addressable row. *pram_size*[3:0] should be set to a value above what the value of *pram_max_addr*[17:0] indicates. For example, to indicate a 48KB memory (12,288 32-bit addresses):

- *pram_size*[3:0] = 4'b1000 or 64KB.
- *pram_max_addr*[17:0] = 18'h 02fff (greates 32-bit address without crossing 48KB boundary)
- Then, if *haddr*[19:0] >= 20'h 0c000, an ERROR response will be generated.

19.2.3.3 Read / Write Introduction

As stated earlier in the ECC Overview, it is essential that each memory address be written to a known value before it is read. This includes reads generated from the read / modify / write operation which occurs when a write transfer of less than 32 bits is requested (see Section 19.2.3.5.2, "Less than 32-bit Writes").

The basic protocol of the AMBA AHB dictates that each address phase is acknowledged with *hready* and is followed by the corresponding data phase. PRAM read transfers work in the same manner but lack a transfer acknowledge. It is assumed that all PRAM phases execute in a single cycle. PRAM write data phases, however, are issued and complete at the same time as the address phase. This behavior is demonstrated in Figure 19-7 in cycles 4 and 5.

In the following timing diagrams, pram_cs_b is always represented in the 64-bit implemenation as a 2-bit field. Since the 32-bit implementation is a subset of the 64-bit implementation, one should be able to infer the behavior of the chip select signal and the controller if there were only one chip select available and the AHB bus was only 32 bits wide. Most of the diagrams depicting 64-bit transfers are shown for specific cases for the 64-bit implementation.

19.2.3.4 Reads

Figure 19-2 shows the read data path through the PRAM_CTL. All read transfers, of any size, complete with a zero wait states response on the AHB. Figure 19-4 shows a few examples of the timing of read transfers on the controller's interfaces. Whether the type of the transfer is sequential or nonsequential, there is no difference in the timing of the controller. Furthermore, reads of less than 32 bits in size are treated exactly the same as a 32-bit read. For the 64-bit implementation, in order to save power, only the applicable chip select is asserted for transfers of 32 bits or less (see Table 19-5).

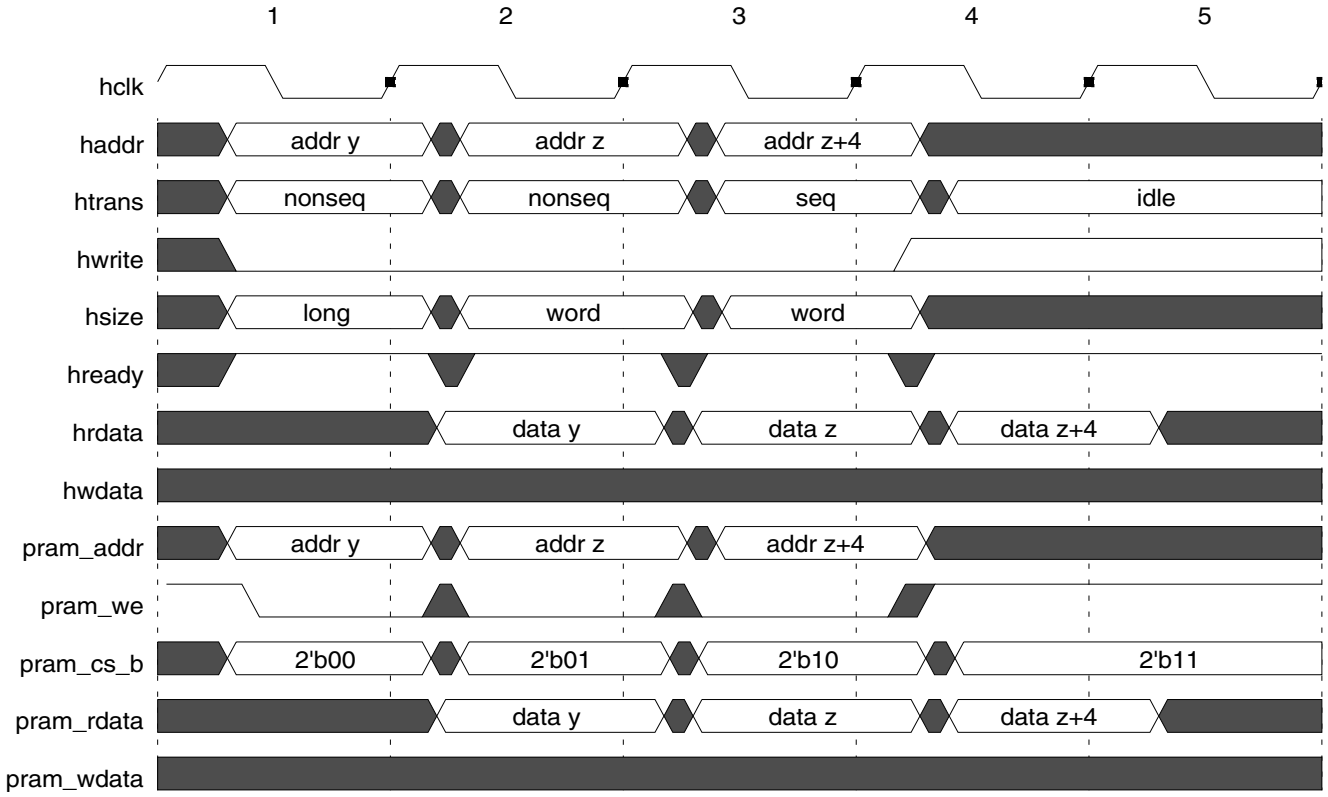


Figure 19-4. 64-bit Read Followed by Two 32-bit Reads

19.2.3.4.1 Unaligned Reads

Any unaligned read will execute as if the size of the transfer, as defined by the *hsize*, is equal to the corresponding unaligned container size. For example, a 32-bit unaligned read will have a container of 64 bits and therefore the controller will execute the read as if it were a 64-bit read. See **Table 4-5** for more information on unaligned accesses.

19.2.3.5 Writes

19.2.3.5.1 32-bit / 64-bit Writes

The write data path is shown in Figure 19-2, highlighted in blue. Aligned 32-bit and 64-bit writes execute in a single AHB data phase cycle allowing for zero wait states on back to back writes of these sizes. Depending on the size and address of the write, one or both array chip selects are asserted (see Table 19-5).

If, during the data phase of a write, a read is requested on the AHB, the write is registered in the late write buffer allowing the read to take place without a wait state. The exception occurs when the read address matches the write address (see Section 19.2.3.6, “Late Write Hits”). The valid buffered or late write data is stored on the next available array address phase.

This behavior is shown in Figure 19-6. Cycle 1 shows the AHB write address phase to address ’y’. In cycle 2, a read request to address ’z’ takes priority over the write and is executed to the array while the write data is stored to the late write buffer. The read in cycle 3 again takes precedence and the late write buffer is held. In cycle 4, the late write is stored. The controller executes in this way for both 32 and 64-bit sizes.

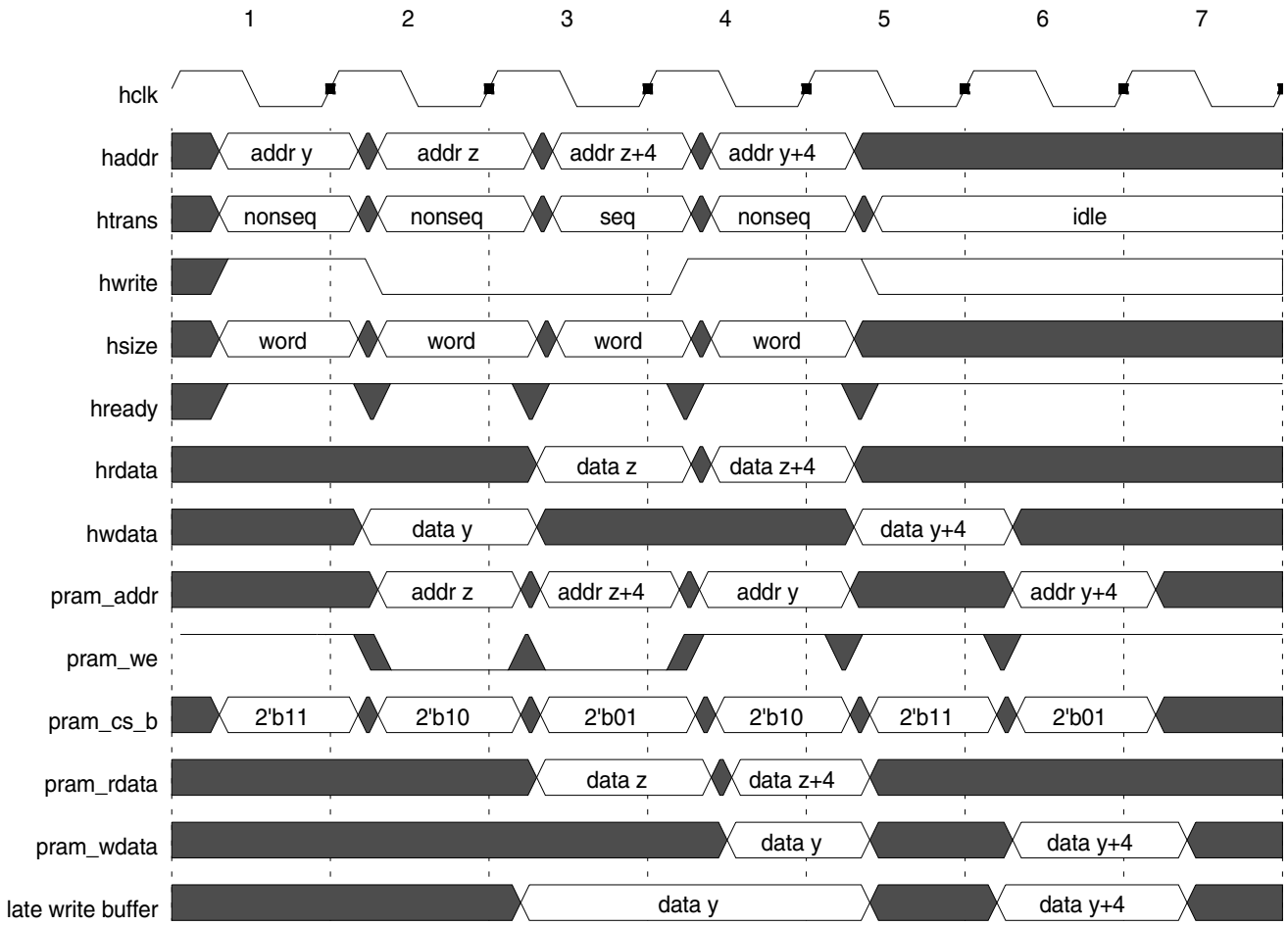


Figure 19-5. 32/64-bit writes with reads

Back to back 32/64-bit writes execute slightly differently in that the write in the AHB data phase is not stored to the late write buffer. Rather, the write data is stored directly to the array in the same cycle in which it is valid on the AHB. Figure 19-7 shows the write to ’addr y’ in cycle 2 executing to the array in the same cycle as the data is valid on the AHB. In addition, the write to ’addr y+32’ in cycle 3 is registered in the late write buffer since it is not followed by a 32 or 64-bit write transfer.

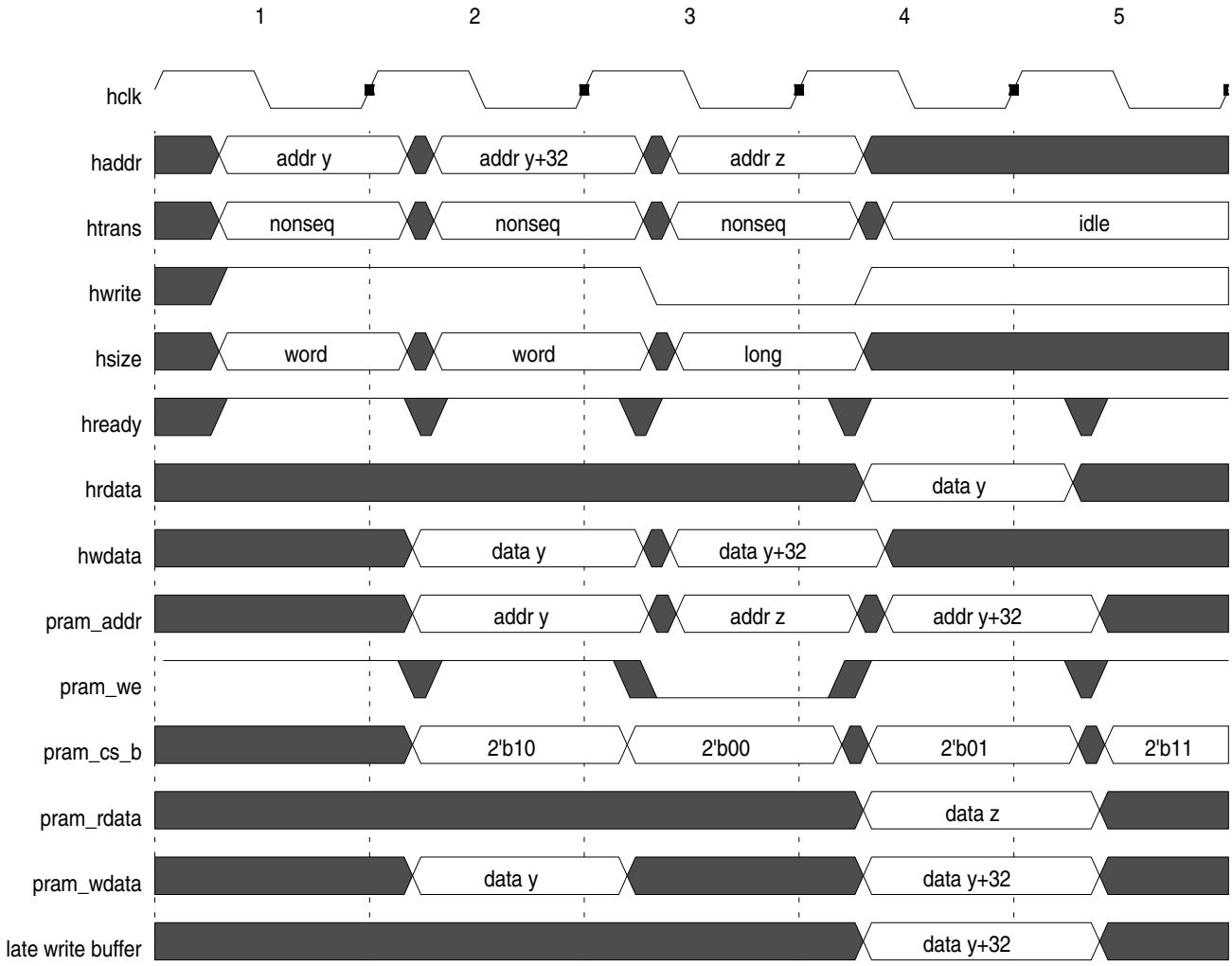


Figure 19-6. Back to Back 32/64-bit Writes

19.2.3.5.2 Less than 32-bit Writes

Less than 32-bit writes require a three step process referred to as a read/modify/write operation. Each 32-bit location must be initialized with a valid codeword created by the ECC encode block shown in Figure 19-2. This codeword must be kept consistent with the entire 32-bit word. Therefore, for each less than 32-bit write, the array data must be read (*read*), the AHB write data merged with the array data and the codeword regenerated (*modify*), before the data can be stored (*write*).

As shown in Figure 19-2, the array read data is registered after it is decoded and before it is merged with the AHB write data. Therefore, every less than 32-bit write requires that one wait state be inserted before the data phase can be completed. This is shown in Figure 19-8. In cycle 3, the AHB data phase completes and the write data is registered in the late write buffer. It is stored to memory on the next available array access. The next available array access is during cycle 4 while the read for ’addr z’ is being decoded.

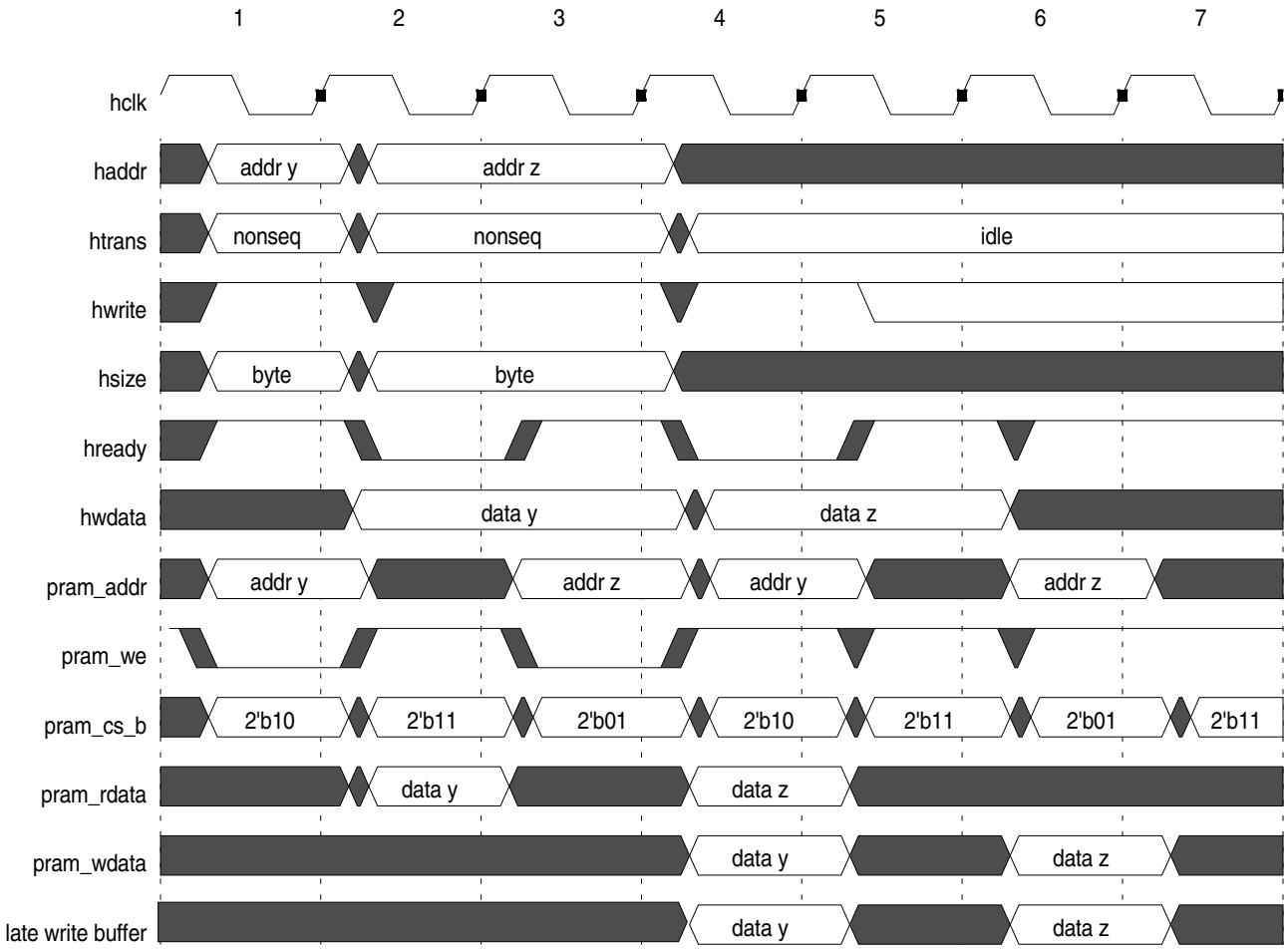


Figure 19-7. Less than 32-bit writes

19.2.3.5.3 Unaligned Writes

The PRAM_CTL controller is compliant with the ARMv6 AMBA Extensions specification with regard to section "3.1 Byte Strobes." In essence, this document specifies that the valid byte strobes (HBSTRB[7:0]) are determined by the container which is determined by HADDR and HSIZE. The size of the transfer will be sufficient to cover all bytes being written and will cover more bytes in the case of a mis-aligned transfer. The address of the transfer will be rounded down to the nearest boundary of the size of the transaction. Table 19-7 below is a replica of the two tables shown in the ARMv6 document. The 32-bit implementation can be inferred from this table as well.

It should be noted that unaligned writes always generate read/modify/write operations in the pram controller in order to preserve the validity of the ECC codeword.

Table 19-7. Unaligned Writes

Transfer description	HADDR	HSIZE[2:0]	HBSTRB[7:0]	HUNALIGN
8-bit access to 0x1000	0x1000	0x0	00000001	0
8-bit access to 0x1003	0x1003	0x0	00001000	0

Table 19-7. Unaligned Writes (Continued)

Transfer description	HADDR	HSIZE[2:0]	HBSTRB[7:0]	HUNALIGN
8-bit access to 0x1007	0x1007	0x0	10000000	0
16-bit access to 0x1000	0x1000	0x1	00000011	0
16-bit access to 0x1005	0x1004	0x2	01100000	1
16-bit access to 0x1007	0x1007 0x1008	0x0 0x0	10000000 00000001	0 0
32-bit access to 0x1000	0x1000	0x2	00001111	0
32-bit access to 0x1002	0x1000	0x3	00111100	1
32-bit access to 0x1003	0x1000	0x3	01111000	1
32-bit access to 0x1007	0x1004 0x1008	0x0 0x2	10000000 00000111	1 1
64-bit access to 0x1000	0x1000	0x3	11111111	0
64-bit access to 0x1003	0x1000 0x1008	0x3 0x2	11111000 00000111	1 1
Update byte 0x0	0x0	0x0	00000001	0
Update byte 0x1	0x1	0x0	00000010	0
Update bytes 0x1, 0x2 and 0x3	0x0	0x2	00001110	1
Update bytes 0x1, 0x2, 0x3 and 0x4	0x0	0x2	00001111	0
Update bytes 0x0, 0x1 and 0x3	0x0	0x2	00001011	1
Update bytes 0x3, 0x4 and 0x5	0x0	0x3	00111000	1
Update bytes 0x0, 0x1, 0x5 and 0x7	0x0	0x3	10100011	1

19.2.3.6 Late Write Hits

A late write hit occurs when the late write buffer is valid and the address of the late write buffer matches *haddr*. The following table shows the possible late write hit scenarios and the result of each.

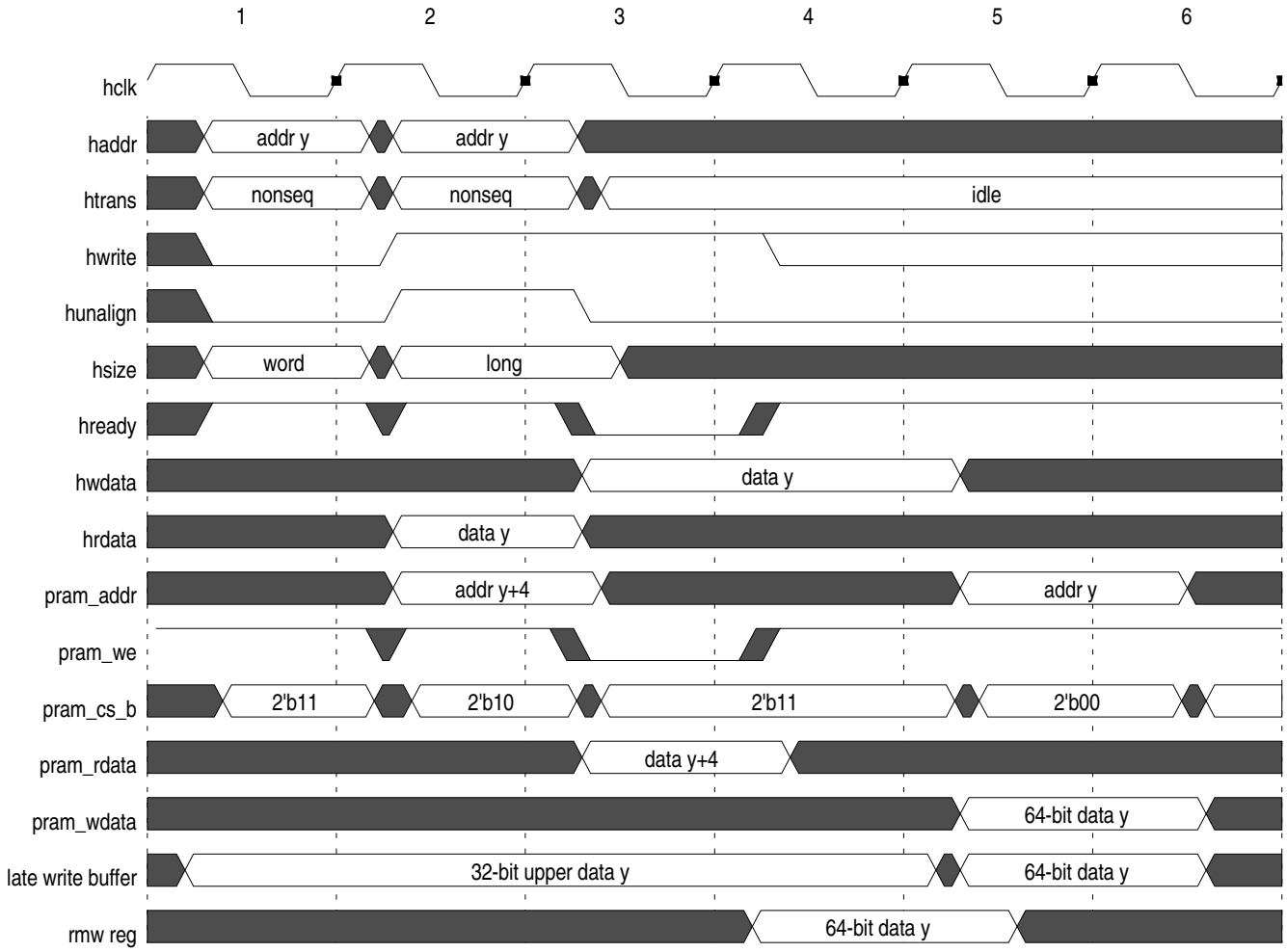
Table 19-8. Late Write Hit Cases

	Access Type	haddr	lwb[1] addr	lwb[1] size	Action Description	Wait State?
1	64-bit read	A^2	A^2	64	hrdata = lwb data	No
2	64-bit read	A^2	$A0^3$ or $A1^4$	32	32bit read of missing word	No
3	64-bit write	A^2	A^{*5}	—	store during address phase	No
4	≤ 32-bit read	$A0^3$	A^2	64	hrdata = lwb data 0	No
5	≤ 32-bit read	$A0^3$	$A0^3$	32	hrdata = lwb data 0	No
6	≤ 32-bit read	$A0^3$	$A1^4$	32	not a hit, actually a miss, lwb held, normal array read	No
7	32-bit write	$A0^3$	—	—	store during address phase	No
8	< 32-bit write	Ax^5	Ax^5	<size>[6]	array read if necessary lwb/pram_rdata merged	No

1. Late Write Buffer
2. 64-bit address A
3. address bit A[2]=0
4. address bit A[2]=1
5. address bit A[2]=either 1 or 0
6. any valid value can be substituted

In all late write hit cases, no additional wait states are inserted on the AHB. For each case in which a read is requested, if the data all of the data is not in the late write buffer, the missing data is requested and the read data mux (shown in Figure 19-2 before the decode block) will select the between the late write buffer and *pram_rdata*.

The timing diagram in Figure 19-8 shows two cases. The first read request demonstrates the case when all read data required is present in the late write buffer. The second case, a read modify write demonstrates the case when only part of the data is in the late write buffer.



Note: rmw reg : read/modify/write register shown in Figure 19-2.

Figure 19-8. Late Write Hits

19.2.3.7 ECC Events on Reads

There are two types of ECC events that can occur on a read from the array, a single-bit or a multiple-bit error. AHB unaligned writes or less than 32-bit writes will generate a read from the array. Therefore, these transactions are subject to ECC decode events.

19.2.3.7.1 Single Bit Errors

Single-bit errors do not affect the AHB transfer in progress. The data bit in error is logically complemented or corrected, and the transaction proceeds normally. In addition, the *ecc_single_error* flag is asserted during the transfer acknowledge (asserted *hready*) and the syndrome and related attributes are valid at this time.

19.2.3.7.2 Mulitiple Bit Errors

A decoded multiple-bit ECC error will result in the termination of the corresponding AHB transaction with the AHB two cycle ERROR response (*hresp)*. This behavior is shown in Figure 19-9. The first transaction, a byte write to 'addr y', results in an error decoded in cycle 2. The data for an errored transaction is not stored back to the array. The array is left unchanged. The response shown in Figure 19-9 would be the same had the request been a read or unaligned write rather than a less than 32-bit write.

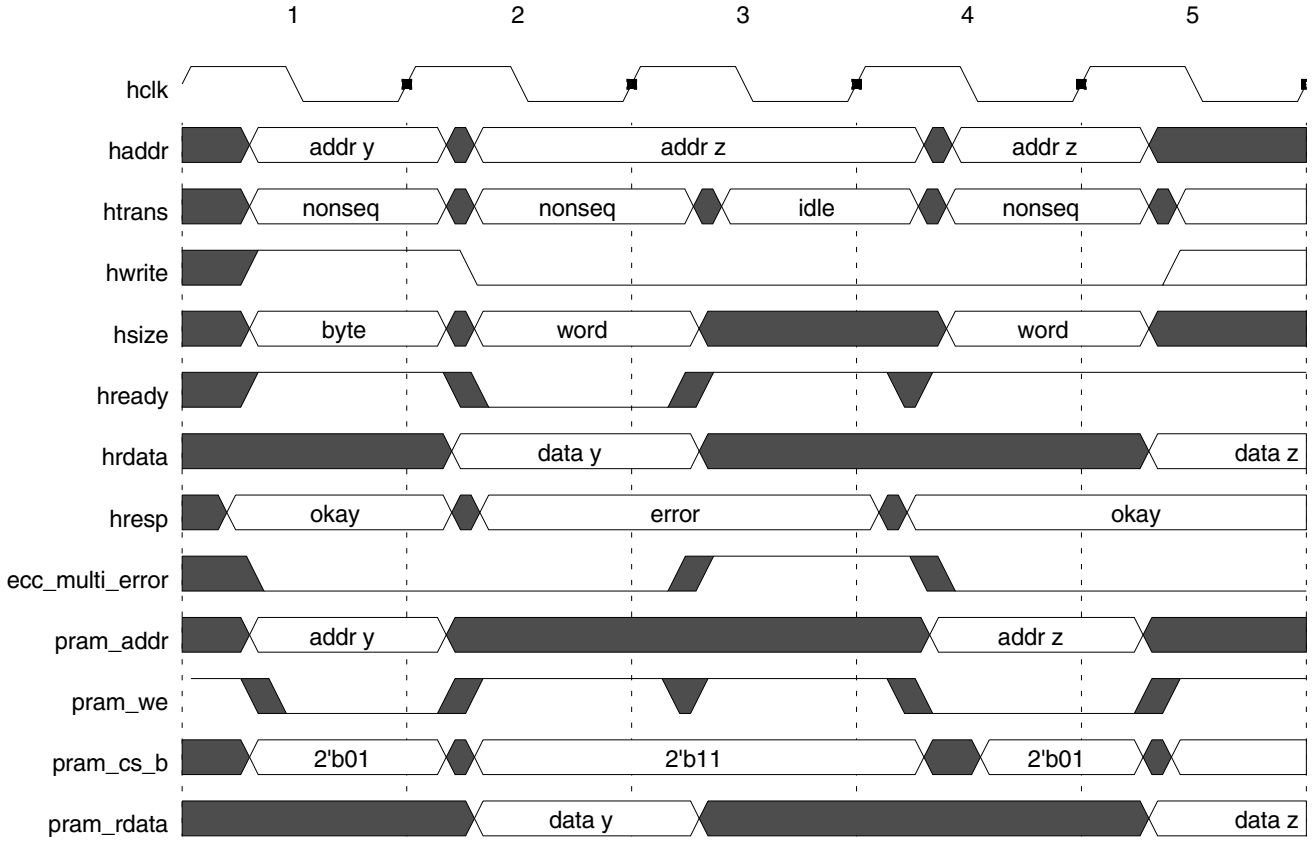


Figure 19-9. Multiple-Bit Error on AHB Read Request

19.2.4 Initialization/Application Information

It is essential that each memory address be written to a known value before it is read. This includes reads generated from the read/modify/write operation which occurs when a write transfer of less than 32 bits or unaligned write is requested (see Section 19.2.3.5.2, “Less than 32-bit Writes”). Without writing an address to a known value first, a read from this address will most likely generate an ECC multiple error.

19.3 Hamming Algorithm

19.3.1 Basic Algorithm

The ECC algorithm is a modified form of the basic Hamming code. First is a description of how the basic Hamming Code is calculated.

19.3.1.1 The Hamming Rule

The number of parity bits required to detect a single-bit error is given by the Hamming rule, and is a function of the number of bits of information transmitted. The Hamming rule is expressed by the following inequality:

$$d + p + 1 \leq 2^p \qquad \textit{Eqn. 19-3}$$

Where **d** is the number of data bits, and **p** is the number of parity (*ECC*) bits. Therefore, to code 32 bits of data and detect *single* errors, 6 ECC bits are needed. By adding an extra ECC bit to check the overall parity of the entire code word, two-bit errors can be detected.

19.3.1.2 Creating A Hamming Codeword

The code word can be created as follows. (Assume the code word bit positions to be laid out as follows: msb < position 1, 2, 3, ... d+p-1, d+p > lsb, this is contradictory notation from the actual rtl notation where the msb is labelled with the highest number as in syndrome[6:0])

1. Mark all bit positions in the code word that are powers of two as ECC bits. (*position 1, 2, 4, 8, 16, etc.*)
2. Place the data to be encoded in remaining bit positions skipping positions reserved for ECC bits. (*data_bit[x] goes into position 3, data_bit[x-1] is placed into position 5, data_bit[x-2] into position 6, etc.*)
3. Each ECC bit calculates the parity for some of the bits in the code word. The position of the ECC bit determines the sequence of bits that it alternately checks and skips in its parity calculation beginning with its own position.
4. These Hamming ECC bits are based on even parity, therefore if the total number of 1’s in the positions it checks is odd, the parity bit is set to a 1 to make the entire parity check even. If the total number of 1’s in the positions checked is even, the parity bit is set to 0.

Example

- Data Byte = `10011010`

- p = 4 bits of ECC
- Code Word = `_ _ 1 _ 0 0 1 _ 1 0 1 0`

NOTE

Remember code word positions 1, 2, 4, and 8 are reserved, so the msb of the data byte is placed in position 3, the next in position 5, etc.

- Position 1 checks every other bit position 1, 3, 5, 7, 9, 11: (? represents the bit position being set)
  `? _ 1 _ 0 0 1 _ 1 0 1 0` : Even parity so position 1 set to a 0.
- Position 2 checks two bits, skips two bits 2, 3, 6, 7, 10, 11 (Notice it checks 2, skips 2, etc)
  `0` **`? 1 _ 0 0 1 _ 1 0 1`** `0` : Odd parity so position 2 set to a 1.
- Position 4 checks 4 bits starting with position 4, then skips 4, etc. Each parity bit continues in the same way checking and skipping the same number as its position.
  `0 1 1` **`? 0 0 1 _ 1 0 1 0`** : Odd parity so position 4 set to a 1.
  `0 1 1 1 0 0 1` **`? 1 0 1 0`** : Even parity so position 8 set to a 0.

The resulting 12-bit code word is `011100101010`. To create the syndrome, the code word is decoded. Each of the 4 parity combinations are checked again using the same method as above, but now on the completed 12bit code word. If the code word does not change, the syndrome will equal all zeros. If, for example, if the bit in position 3 of the code word is inverted, the generated syndrome will be {position 8,4,2,1} = 4'b0011 ~ 3. Bit 1 of the syndrome is decoded as the parity of positions 1, 3, 5, 7, 9, 11. If bit position 3 has flipped, then the number of 1's has changed from 4 to 3. Therefore this syndrome bit is assigned a 1 because the parity of the positions it calculates is odd. The same occurs for syndrome bit 2. The entire syndrome decodes to 3. This number gives the exact bit position in the code word to be corrected.

Adding one more ECC bit to check the parity of the entire code word allows for double bit detection. In the PRAM_CTL this bit is based on odd parity. Therefore, for the example above, an odd parity based ECC bit would be assigned the value 1'b1 and added to the end of the code word to make the entire code word odd (*there are 6 1's otherwise in the code word*). When code word bit 3 flips, the parity of the entire codeword becomes even, signalling a single-bit error when the syndrome is non-zero. If the syndrome is non-zero, and the parity check of the entire code word is a one (*odd*), then at least two bits must have changed.

19.3.1.3 Hamming Parity Code Table

This simple Hamming code is nearly the same algorithm used in the PRAM_CTL, but on a larger scale of 32 bits at a time. Table 19-9 shows the would be relationship between each data bit and the ECC bits for the basic Hamming Code. For each column, an 'X' represents the inclusion of the corresponding data bit in the XOR equation for the corresponding ECC bit. For instance, data bit 31 goes into the equations for calculating ECC bits 0 and 1. Therefore, if the resulting syndrome is 6'b000011 ~ 3, then data bit 31 would be corrected.

Table 19-9. Parity Codes: ECC Bits vs. Data Bits

	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
5																											X	X	X	X	X	X
4												X	X	X	X	X	X	X	X	X	X	X	X	X	X	X						
3					X	X	X	X	X	X	X								X	X	X	X	X	X	X	X						
2		X	X	X				X	X	X	X				X	X	X	X					X	X	X	X				X	X	X
1	X		X	X		X	X			X	X		X	X			X	X			X	X			X	X		X	X			X
0	X	X		X	X		X		X	X		X		X		X		X		X		X		X		X	X		X		X	

19.3.1.4 PRAM_CTL Implementation

Although correct, the above table is not the most efficient way to map the 64 possible codes to the 32 data bits for harware purposes. In Table 19-9, the first codeword used is 6’b000011. Each consecutive codeword increments by 1, skipping each codeword which is a factor of 2. Notice above that each data bit must factor into at least 2 parity bit equations. Another way to say all of this is at least 2 parity bits must be used to cover a data bit.

Also notice that not all of the codewords are used. The final code word used, 6’b100110, is on data bit 0. This leaves all of the codewords from 100111 to 111111 unused. By intelligently changing out codewords with a large number of Xs for unused codewords with fewer 1’s in them, we can reduce the loading on the data bits as well as more evenly distribute the number of data bits needed to calculate each parity bit. In Table 19-10, you can see the number of data bits (or Xs in each row on the table) that are used to calculate each parity bit. There is an average of 15 with a critical path of 18 data bits to XOR.

Table 19-10. Hamming Parity Delay

ParityBit	# Data Bits
5	6
4	15
3	15
2	18
1	18
0	18

By creatively substituting some of the unused codewords we can balance the delay for each parity bit and decrease the loading on the data bits. Table 19-11 shows a modified Hamming code where data bits 6, 7, 8, 10, 14, 21 and 22 have had their codewords changed for unused codewords. With this implementation the average number of data bits to be XOR’ed for each parity bit reduces to 13.5 and the critical path reduces to 14 data bits.

Table 19-11. Modified Parity Codes: ECC Bits vs. Data Bits

	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
5											X							X				X		X	X	X	X	X	X	X	X	X
4												X	X	X	X	X	X		X	X	X	X	X	X	X	X						
3					X	X	X	X	X	X	X							X	X	X	X		X		X							
2		X	X	X				X	X	X					X	X	X	X					X							X	X	X
1	X		X	X		X	X			X			X	X			X				X	X						X	X			X
0	X	X		X	X		X		X			X		X		X				X				X			X		X		X	

Chapter 20
Boot Assist Module (BAM)

20.1 BAM Introduction

The Boot Assist Module (BAM) provides the JTAG Lockout Recovery functionality that was available in hardware on the MAC71xx family of devices. It is essentially an AHB slave containing hard-code instructions to perform the following sequence of events:

1. Unlock all Flash registers (PFCR2 register)
2. Enable the Flash for read/write (PFAPR register)
3. Configure the DACC/SACC registers
4. Enable LAS and MAS (LML register)
5. Enable HAS (HBL register)
6. Enable Shadow (SLL)
7. Enable all blocks for program/erase
8. Erase the main array
   - Set the ERS bit
   - Perform an interlock write
   - Set the EHV bit
9. Wait until erase is done (DONE=1)
10. Check that the erase was successful (PEG=1)
11. Clear the EHV bit
12. Clear the ERS bit
13. Erase the shadow array
    - Set the ERS bit
    - Perform an interlock write
    - Set the EHV bit
14. Wait until erase is done (DONE=1)
15. Check that the erase was successful (PEG=1)
16. Clear the EHV bit
17. Clear the ERS bit
18. Program the System Censor word in the Shadow Block
    - Set the PGM bit
    - Perform a program write (Write $80000002 to address SHADOW_BASE+$7de0)

— Set the EHV bit

19. Wait until program is done (DONE=1)
20. Check that the program was successful (PEG=1)
21. Clear the EHV bit
22. Clear the PGM bit
23. Perform a data fetch on the first instruction in the BAM to indicate that the sequence is complete.

20.2 BAM Features

- Provides JTAG Lockout Recovery sequence

20.3 BAM Protocol

The BAM uses the AHB-Lite v2.0 specification for its main interface. Figure 20-1 illustrates the handshaking between the SC4 debug register and the BAM module.

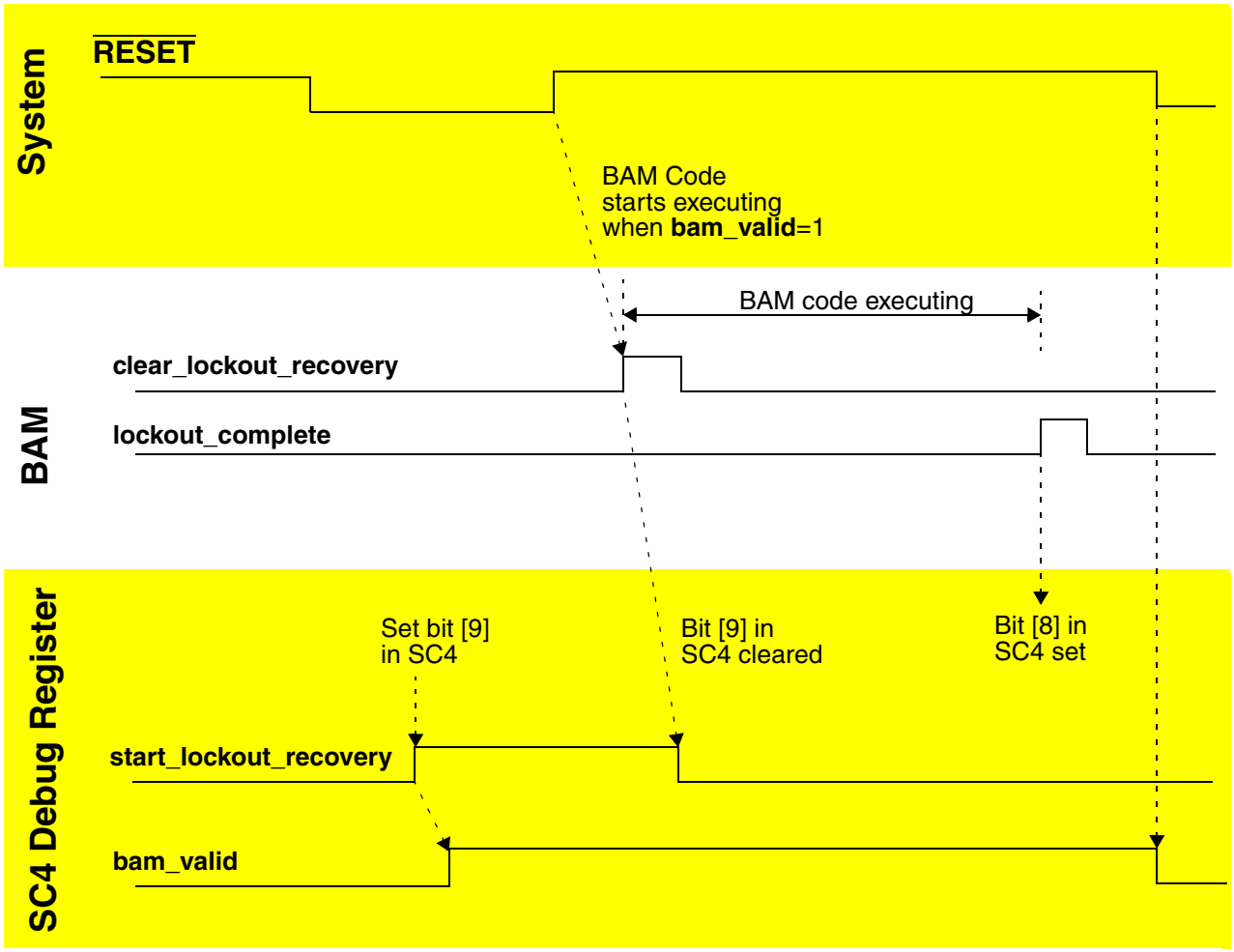


Figure 20-1. BAM Sequence

From a user/tool perspective, the sequence of events looks like the following:

1. If the device is powered off, perform a correct power on sequence first
2. Assert the external RESET of the device
3. While the RESET is asserted, write bit 9 of the SC4 register using the JTAG interface
4. Release the RESET
5. There is no need to clear bit 9 of the SC4 register, as it is self-clearing
6. Continuously poll the SC4 register (READ) until bit 8 (Lockout Recovery Complete) is set.
7. If bit 19 is set (1), it means that the system is secured, and the Lockout Recovery process must have failed. If bit 19 is cleared (0), which is the expected result, then the system is unsecured.
8. Note that it is not necessary to write the Security Word in the system after performing the JTAG Lockout Recovery, as this is done automatically during the procedure.
9. Once the device is unsecured, reset the device by asserting the RESET pin. You may reset into any functional mode.

20.4 BAM External Pins

There are no BAM signals that drive or are driven from MCU pins.

20.5 BAM Bus Aborts

The BAM will signal a bus abort under any of the following conditions:

- Access was not in Supervisor Mode
- Access was 8 or 16 bits
- Access was a WRITE
- Access was a burst

Because the BAM resides as a slave on the crossbar switch, all bus aborts from the BAM will cause either a Prefetch or Data Abort in the ARM7 core.

20.6 BAM Differences from MAC71xx

- This is a new block, that was not present on the MAC71xx family

20.7 BAM Application Usage

Although the BAM is intended primarily for JTAG Lockout Recovery, the code contained in it is completely visible to the application code. If the application code wants to perform a Lockout Recovery, it may simply jump to address $A000 0000. However, the BAM code always end in an endless loop, which can only be exited by resetting the device.

Chapter 21
IPI Subsystem (IPSS)

The IPI Subsystem (IPSS) is responsible for connecting all Peripheral Bus peripherals to the AIPS module.

21.1 Bus Abort handling

Bus aborts on the Peripheral Bus fall into one of two categories:

- Bus aborts produced by peripherals due to an illegal access. This type of abort may be masked with the **REG_ABORT** bit in the **ERROR** register in the SSM module.
- Bus aborts produced accesses to an empty peripheral slot. These aborts are OR'd together and gated by the **PER_ABORT** bit in the **ERROR** register in the SSM module.

21.2 Peripheral Bus Peripheral Clock Frequencies

Half Speed	Full Speed
CAN_A/B	ATD_A
PIT	DSPI_A/B/C
eMIOS	FlexBus (registers and protocol)
SCI_A/B	MCM
Flash (registers only)	INTC
SSM	DMA2
CRG (registers only)	AIPS (registers)
DMA Channel Mux	
PIM	
IIC	

21.3 IPSS Differences from MAC71xx

- Bus aborts for empty peripheral slots may now be masked using a bit in the SSM
- Following peripherals now run at full system speed (was 1/2 system speed on MAC71x1)
  - — ATD_A
  - — DSPI_A/B/C

Chapter 22
Dual-Output Voltage Regulator (VREG_HIP7A)

22.1 Introduction

The Voltage Regulator provides the internal voltage for the core and logic, which enables devices in the MAC7200 family to be supplied with a single 5V power supply source.

NOTE

The Voltage regulator on the MAC7200 family of devices does not implement the VREGEN pin as described in the VREG_3V3 Block Guide. The Voltage Regulator is enabled by the level on the VDDR pin only.

22.1.1 Overview

Block VREG_HIP7A is a dual output voltage regulator providing the following supplies:

- 1.5V (typ) VDD15
- 3.3V (typ) VDD33
- 3.3V (typ) VDDPLL

The regulator input voltage range is 5V (typ).

22.1.2 Features

The block VREG_HIP7A includes these distinctive features:

- Two parallel, linear voltage regulators
  - Bandgap reference
- On-chip Voltage Regulators generate all necessary internal supply voltages from 5V only input voltage, including Flash, Oscillator, PLL and core supply voltage.
- Bypass mode allows off-chip supply of all on-chip voltages.
- Power On Reset (POR) and Low Voltage Reset (LVR) detection with independent flags for full reset source reporting.

22.1.3 Modes of Operation

There are two modes VREG_HIP7A can operate in:

- Full Performance Mode (FPM)

The regulator is active, providing the nominal supply voltage of 1.5V/3.3V with full current sourcing capability at all outputs. Features LVR (Low Voltage Reset) and POR (Power-On Reset) are available.

- Shutdown Mode

  Controlled by VREGEN (see MCU level specification for connectivity of VREGEN).

  This mode is characterized by minimum power consumption. The regulator outputs are in a high impedance state, only the POR feature is available, LVR is disabled.

  This mode must be used to disable the chip internal regulator VREG_HIP7A, i.e. to bypass the VREG_HIP7A to use external supplies.

22.1.4 Block Diagram

Figure 22-1 shows the function principle of VREG_HIP7A by means of a block diagram. The regulator core REG consists of two parallel subblocks, REG1 and REG2, providing two independent output voltages.

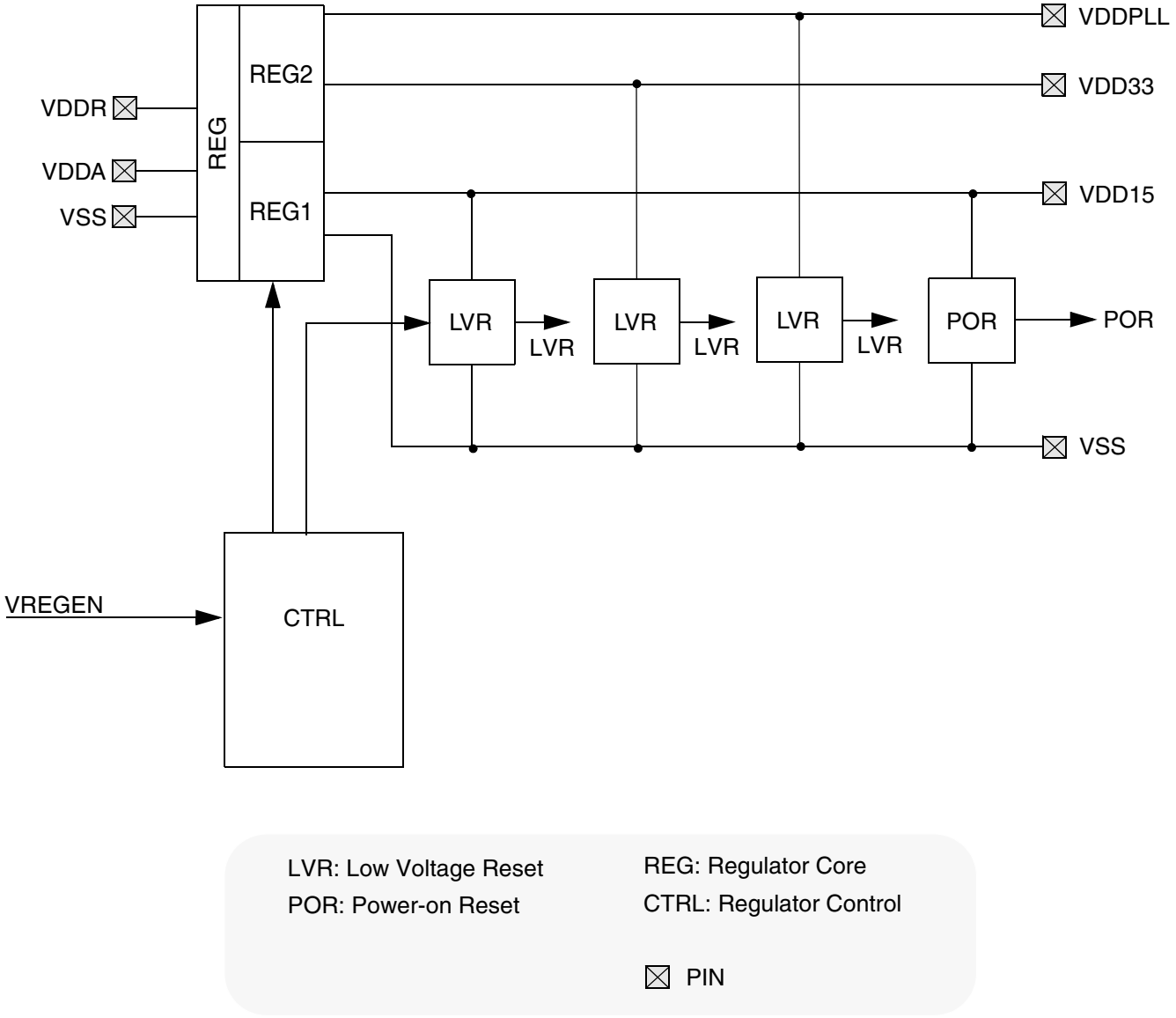


Figure 22-1. VREG_HIP7A Block Diagram

22.2 External Signal Description

22.2.1 Overview

Due to the nature of VREG_HIP7A being a voltage regulator providing the chip internal power supply voltages most signals are power supply signals connected to pads.

Table 22-1 shows all signals of VREG_HIP7A associated with pins.

Table 22-1. VREG_HIP7A - Signal Properties

Name	Function	Reset State	Pull up
VDDR	VREG_HIP7A power input (positive supply)	—	—
VDDA	VREG_HIP7A quiet input (positive supply)	—	—
VSSA	VREG_HIP7A quiet input (ground)	—	—
VDD15	VREG_HIP7A primary 1.5V output (positive supply)	—	—
VSS15	VREG_HIP7A primary 1.5V output (ground)	—	—
VDD33	VREG_HIP7A primary 3.3V output (positive supply)	—	—
VSS33	VREG_HIP7A primary 3.3V output (ground)	—	—
VDDPLL	VREG_HIP7A secondary 3.3V output (positive supply)	—	—
VSSPLL	VREG_HIP7A secondary 3.3V output (ground)	—	—
VREGEN (optional)	VREG_HIP7A (Optional) Regulator Enable	—	—

22.2.2 Detailed Signal Descriptions

Check device level specification for connectivity of the signals.

22.2.2.1 VDDR - Regulator Power Input

Signal VDDR is the power input of VREG_HIP7A. All currents sourced into the regulator loads flow through this pin. A chip external decoupling capacitor (100nF...220nF, X7R ceramic) between VDDR and VSS can smooth ripple on VDDR.

For entering Shutdown Mode pin VDDR should also be tied to ground on devices without VREGEN pin.

22.2.2.2 VDDA, VSSA - Regulator Reference Supply

Signals VDDA/VSSA which are supposed to be relatively quiet are used to supply the analog parts of the regulator. Internal precision reference circuits are supplied from these signals. A chip external decoupling capacitor (100nF...220nF, X7R ceramic) between VDDA and VSSA can further improve the quality of this supply.

22.2.2.3 VDD15, VSS15 - Regulator Output1 (Core Logic)

Signals VDD15/VSS15 are the primary outputs of VREG_HIP7A that provide the power supply for the core logic. These signals are connected to device pins to allow external decoupling capacitors (100nF...220nF, X7R ceramic).

In Shutdown Mode an external supply driving VDD15/VSS15 can replace the voltage regulator.

22.2.2.4 VDD33, VSS33 - Regulator Output1 (3.3V Logic)

Signals VDD33/VSS33 are the primary outputs of VREG_HIP7A that provide the power supply for the 3.3V logic. These signals are connected to device pins to allow external decoupling capacitors (100nF...220nF, X7R ceramic).

In Shutdown Mode an external supply driving VDD33/VSS33 can replace the voltage regulator.

22.2.2.5 VDDPLL, VSSPLL - Regulator Output2 (3.3V PLL)

Signals VDDPLL/VSSPLL are the secondary outputs of VREG_HIP7A that provide the power supply for the PLL and Oscillator. These signals are connected to device pins to allow external decoupling capacitors (100nF...220nF, X7R ceramic).

In Shutdown Mode an external supply driving VDDPLL/VSSPLL can replace the voltage regulator.

22.2.2.6 VREGEN - Optional Regulator Enable

This optional signal is used to shutdown VREG_HIP7A. In that case VDD/VSS and VDDPLL/VSSPLL must be provided externally. Shutdown Mode is entered with VREGEN being low. If VREGEN is high, the VREG_HIP7A is in Full Performance Mode.

VREGEN is connected to VDDR.

NOTE

Switching from FPM to shutdown of VREG_HIP7A and vice versa is not supported while MCU is powered.

22.3 Memory Map and Register Definition

There are no registers available in the VREG_HIP7A block. All control and status registers are located in the Clock and Reset Generation (CRG) module.

22.4 Functional Description

22.4.1 General

Block VREG_HIP7A is a voltage regulator as depicted in Figure 22-1. The regulator functional elements are the regulator core (REG), a low voltage detect module (LVD), a control block (CTRL), a power-on reset module (POR) and a low voltage reset module (LVR).

22.4.2 REG - Regulator Core

VREG_HIP7A, respectively its regulator core has two parallel, independent regulation loops (REG1 and REG2) that differ only in the amount of current that can be delivered.

The regulator is a linear regulator with a bandgap reference when operated in Full Performance Mode. It acts as a voltage clamp in Reduced Power Mode. All load currents flow from input VDDR to VSS or VSSPLL. The reference circuits are supplied by VDDA and VSSA.

22.4.2.1 Full Performance Mode

In Full Performance Mode the output voltage is compared with a reference voltage by an operational amplifier. The amplified input voltage difference drives the gate of an output transistor.

22.4.3 POR - Power-On Reset

This functional block monitors VDD15. If VDD15 is below V_{PORD}, POR is asserted, if VDD15 exceeds V_{PORD}, the POR is de-asserted. POR asserted forces the MCU into Reset. POR De-asserted will trigger the power on sequence.

22.4.4 LVR15 - Low Voltage Reset

Block LVR monitors the primary output voltage VDD15. If it drops below the assertion level (V_{LVR15A}), signal LVR15 asserts. If VDD15 rises above the de-assertion level (V_{LVR15D}), signal LVR15 de-asserts. The LVR function is available only in Full Performance Mode.

22.4.5 LVR33 - Low Voltage Reset

Block LVR monitors the primary output voltage VDD33. If it drops below the assertion level (V_{LVR33A}), signal LVR33 asserts. If VDD33rises above the de-assertion level (V_{LVR33D}), signal LVR33 de-asserts. The LVR function is available only in Full Performance Mode.

22.4.6 LVRPLL - Low Voltage Reset

Block LVR monitors the primary output voltage VDDPLL. If it drops below the assertion level ($V_{LVRPLLA}$), signal LVRPLL asserts. If VDDPLL rises above the de-assertion level ($V_{LVRPLLD}$), signal LVRPLL de-asserts. The LVR function is available only in Full Performance Mode.

22.4.7 Resets

22.4.7.1 General

This section describes how VREG_HIP7A controls the reset of the MCU. Possible reset sources are listed in Table 22-2.

Table 22-2. VREG_HIP7A - Reset Sources

Reset Source	Local Enable
Power-on Reset (POR)	Always active
Low Voltage Reset (LVR)	Available only in Full Performance Mode

22.4.7.2 Description of Reset Operation

22.4.7.2.1 Power-On Reset (POR)

During chip power-up the digital core may not work if its supply voltage VDD15 is below the POR deassertion level (V_{PORD}). Therefore signal POR which forces the other blocks of the device into reset is kept high until VDD15 exceeds V_{PORD}. The MCU will run the start-up sequence after POR deassertion. The power-on reset is active in all operation modes of VREG_HIP7A.

22.4.7.2.2 Low Voltage Reset (LVR)

For details on low voltage reset see Section 22.4.4, “LVR15 - Low Voltage Reset,” Section 22.4.5, “LVR33 - Low Voltage Reset,” and Section 22.4.6, “LVRPLL - Low Voltage Reset.”

22.5 Interrupts

There are no interrupts driven by the VREG_HIP7A block. All voltage related interrupts are driven by the Clock and Reset Generation (CRG) Block.

22.6 VREG Bus Aborts

There is no bus interface on the Voltage Regulator, as all associated control and status bits are located inside the CRG module.

22.7 VREG Differences from MAC71xx

- Removed Standby Mode (Required only for STOP mode)
- Removed RC oscillator/Asynchronous Periodic Interrupt (API)
- Removed Low Voltage interrupt (LVIE,LVIF)
- Moved LVDS register bit functionality into the CRG Module (See reset source reg in CRG)

Chapter 23
Clock and Reset Generator (CRG)

23.1 Introduction

This chapter describes the function of the Clocks and Reset Generator (CRG).

23.1.1 CRG Overview

The CRG module is used to provide the clocks for the MCU, to control its Reset operation and low power operating modes. Included in the module is the PLL, the Clock Generator and the Reset Generator.

The module provides the clock for the Real Time Interrupt counter contained within the PIT, the clock for the Software Watchdog Timer in the platform, the main system clock for the core and the clock used for the peripherals and the Flash memory, which runs at half the speed of the System clock. For more information on the clocks used throughout the MCU refer to Chapter 5, “System Clock Description”. All memory mapped configuration for the PLL is in the CRG module.

The CRG is also used to control entry into the low power operating modes of the device, as well as the features of the device which continue to operate during low power modes. Doze mode is entered by writing to the Doze Control register in the CRG, while a write to the Clock Select Register will control whether the PLL, the Real Time Interrupt (RTI) and the Software Watchdog Timer (SWT) clocks are disabled in Doze mode.

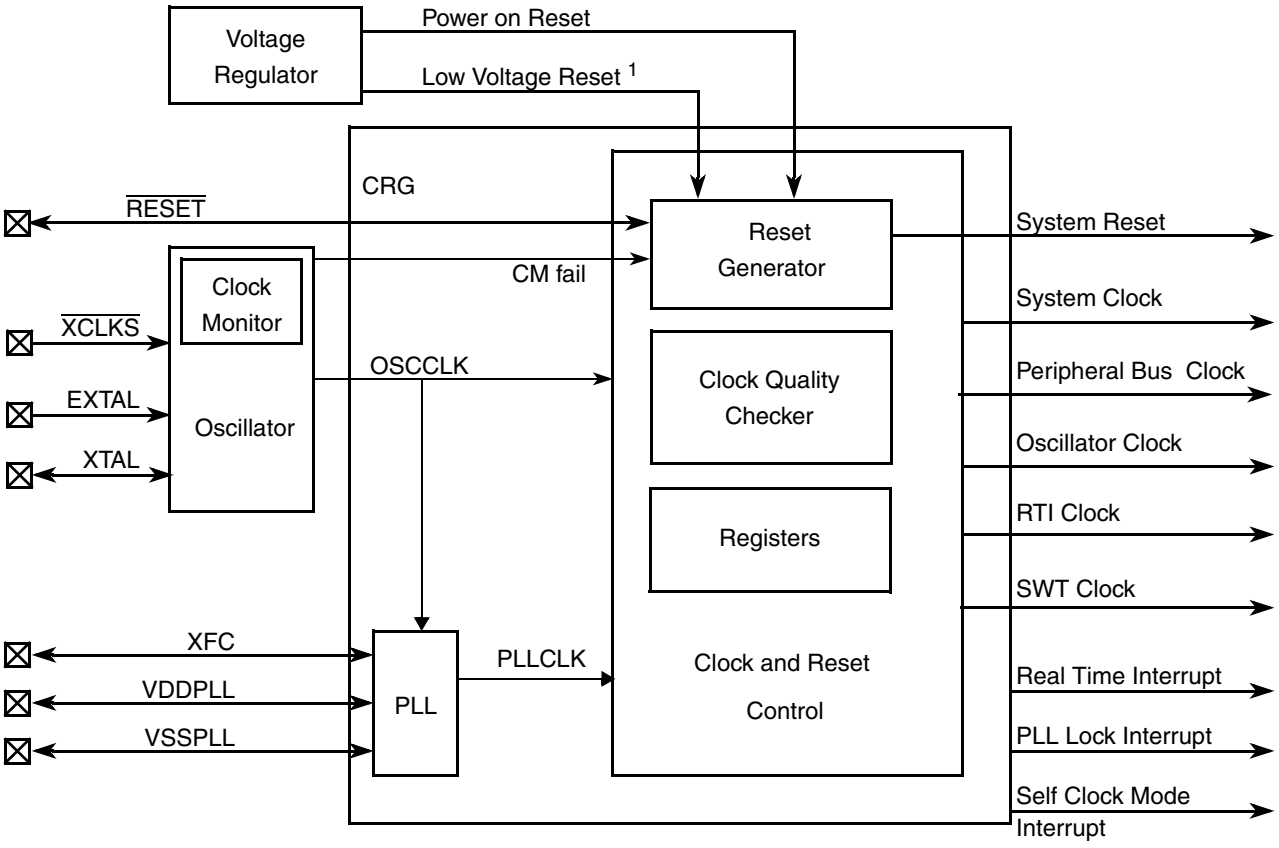
The module provides a Crystal Monitor which detects the presence of the oscillator clock. If the oscillator clock is not present within a defined time, determined by the Crystal Monitor time out period, the module will either generate a Reset, or initiate the PLL Self Clock mode. In Self Clock mode the PLL will generate its own clock based on the minimum VCO frequency. This can be used to clock the device in order to continue some basic operation in the absence of an external clock.

The Clock Quality Checker (CQC) is included in the CRG and provides a more accurate check of the oscillator output clock. The CQC is run following events such as Power On Reset or wake-up from STOP mode, and counts the number of clocks over a defined time window. Failure of the Clock Quality Checker can be used to initiate Self Clocking mode or a Crystal Monitor Reset event.

The CRG module provides information in the form of Flags to help identify the source of a Reset event (see also Chapter 6, “Resets”).

23.1.2 CRG Block Diagram

Figure 23-1 shows a block diagram of the CRG.



1) Refer to device specification for availability of the low voltage reset feature.

Figure 23-1. Block diagram of CRG

23.1.3 Features

The main features of this block are:

- Phase Locked Loop (PLL) frequency multiplier
  - — Reference divider
  - — Automatic bandwidth control mode for low-jitter operation
  - — Automatic frequency lock detector
  - — CPU interrupt on entry or exit from locked condition
  - — Self Clock Mode in absence of reference clock
- System Clock Generator
  - — Clock Quality Check
  - — User selectablefast wake-up from Stop in Self-Clock Mode for power saving and immediate program execution
  - — Clock switch for either Oscillator or PLL based system clocks

— User selectable disabling of clocks during Doze Mode for reduced power consumption

- System Reset generation from the following possible sources:
  — Power on reset
  — Low voltage reset — this will assert if any of the Low Voltage Reset modules detect a voltage below their set point)
  — SWT reset
  — Loss of clock reset
  — External pin reset

23.1.4 Modes of Operation

This subsection lists and briefly describes all operating modes supported by the CRG.

- Run Mode

  All functional parts of the CRG are running during normal Run Mode.
- Doze Mode

  This mode allows to disable the system and peripheral clocks depending on the configuration of the individual bits in the CLKSEL register.
- Self Clock Mode

  Self Clock Mode will be entered if the Clock Monitor Enable Bit (CME) and the Clock Monitor Reset Disable (CMRD) are both asserted and the clock monitor in the oscillator block detects a loss of clock. As soon as Self Clock Mode is entered, the CRG starts to perform a clock quality check. Self Clock Mode remains active until the clock quality check indicates that the required quality of the incoming clock signal is met (frequency above hysteresis level). Self Clock Mode should be used for safety purposes only. It provides reduced functionality to the MCU in case where a loss of clock is causing severe system conditions.

23.2 External Signal Description

Table 23-1 lists and describes the signals that connect off chip.

Table 23-1. Signal Properties

Name	I/O Type	Function
VDDPLL	Input	Operating Voltage
VSSPLL	Input	Ground
XFC	Output/Input	External Loop Filter
$\overline{\text{RESET}}$	Output/Input	Reset Input/Output
CLKOUT/$\overline{\text{XCLKS}}$	Output/Input	Pierce / Square Wave Mode Selection

23.2.1 Detailed Signal Descriptions

23.2.1.1 V_{DDPLL}, V_{SSPLL}

These pins provides operating voltage (VDDPLL) and ground (VSSPLL) for the PLL circuitry. This allows the supply voltage to the PLL to be independently bypassed. Even if PLL usage is not required, VDDPLL and VSSPLL must be connected properly.

23.2.1.2 XFC

A passive external loop filter must be placed on the XFC pin. The filter is a second-order, low-pass filter to eliminate the VCO input ripple. The value of the external filter network and the reference frequency determines the speed of the corrections and the stability of the PLL. Refer to device specification for calculation of PLL Loop Filter (XFC) components. If PLL usage is not required, the XFC pin must be tied to VDDPLL.

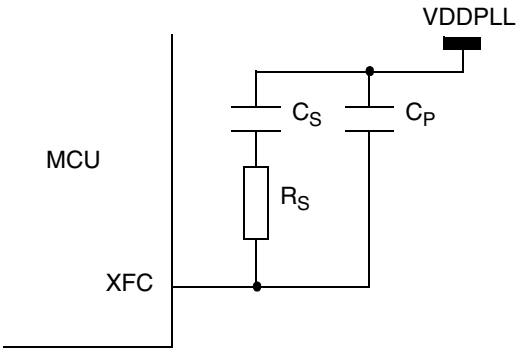


Figure 23-2. PLL Loop Filter Connections

If the XFC pin is shorted to 3.3V, then the VCO will synthesize the lowest possible frequency. Since the minimum VCO locking frequency is 30 MHz, it is expected that this situation would result in the PLL not locking, and thus generating a software detectable condition. Because the Self Clock mode is specified as 2-5.5MHz, the device should continue to operate in Self Clock Mode, with software running and able to detect this condition.

23.2.1.3 RESET

$\overline{\text{RESET}}$ is an active low bidirectional reset pin. As an input, it initializes the MCU asynchronously to a known start-up state. As an open-drain output, it indicates that a system reset (internal to MCU) has been triggered.

23.2.1.4 CLKOUT/$\overline{\text{XCLKS}}$

After reset, CLKOUT provides the reference clock for both the GPIO and FlexBus interfaces. On the xM84D maskset, it is driven from the Peripheral Bus Clock.

$\overline{\text{XCLKS}}$ decides between Pierce ($\overline{\text{XCLKS}}$ = 1) and Square Wave ($\overline{\text{XCLKS}}$ = 0) mode. This pin is sampled during reset.

23.3 Memory Map and Register Definition

This section provides a detailed description of all registers accessible in the CRG.

23.3.1 Memory Map

Figure 23-2 gives an overview of all CRG registers.

Table 23-2. CRG Memory Map

Address	Use	Access
Base + $_00	CRG Synthesizer Register (SYNR)	R/W
Base + $_01	CRG Reference Divider Register (REFDV)	R/W
Base + $_02	CRG Flag Register 1 (CTFLG)	R/W
Base + $_03	CRG Flag Register 2 (CRGFLG)	R/W
Base + $_04	CRG Interrupt Enable Register (CRGINT)	R/W
Base + $_05	CRG Clock Select Register (CLKSEL)	R/W
Base + $_06	CRG PLL Control Register (PLLCTL)	R/W
Base + $_07	CRG STOP/Doze Control Register (SDMCTL)	R/W
Base + $_08	CRG BDM Control Register (BDMCTL)	R/W
Base + $_09- $_0F	Reserved	R

NOTE

Register Address = Base Address + Address Offset, where the Base Address is defined at the MCU level and the Address Offset is defined at the module level.

23.3.2 Register Descriptions

This section describes, in address order, all the CRG registers and their individual bits.

23.3.2.1 CRG Synthesizer Register (SYNR)

The SYNR register controls the multiplication factor of the PLL. If the PLL is on, the count in the loop divider (SYNR) register effectively multiplies up the PLL clock (PLLCLK) from the reference frequency by 2 x (SYNR+1). PLLCLK will not be below the minimum VCO frequency (f_{SCM}).

$$PLLCLK = 2 \times OSCCLK \times \frac{SYNR + 1}{REFDV + 1} \qquad \textit{Eqn. 23-1}$$

NOTE

If PLL is selected (PLLSEL=1), System Clock = PLLCLK
System Clock must not exceed the maximum operating system frequency.

Address: Base + $_00

Access: User read/write
Read: anytime
Write: anytime except if PLLSEL = 1

	7	6	5	4	3	2	1	0
R	0	0	SYN5	SYN4	SYN3	SYN2	SYN1	SYN0
W								
Reset	0	0	0	0	0	0	0	0

Figure 23-3. CRG Synthesizer Register (SYNR)

NOTE

Write to this register initializes the lock detector bit and the track detector bit.

23.3.2.2 CRG Reference Divider Register (REFDV)

The REFDV register provides a finer granularity for the PLL multiplier steps. The count in the reference divider divides the OSCCLK frequency by REFDV+1

Address: Base + $_01

Access: User read/write
Read: anytime
Write: anytime except when PLLSEL = 1

	7	6	5	4	3	2	1	0
R	0	0	0	0	REFDV3	REFDV2	REFDV1	REFDV0
W								
Reset	0	0	0	0	0	0	0	0

Figure 23-4. CRG Reference Divider Register (REFDV)

NOTE

Write to this register initializes the lock detector bit and the track detector bit.

23.3.2.3 CRG ARM Flag Register 1 (CTFLG)

This register provides status bits and flags.

Address: Base + $_02

Access: User read/write
Read: anytime
Write: refer to each bit for individual restrictions.

	7	6	5	4	3	2	1	0
R	0	0	ILR	JTR	SFR	CMR	EXR	WDR
W								
Reset	0	0	0	0	0	0	0	0

Figure 23-5. CRG Flag Register 1 (CTFLG)

Table 23-3. CTFLG Field Descriptions

Field	Descriptions
7–6	Reserved, should be cleared.
5 ILR	Illegal address reset flag. ILR is set to 1 when a bus abort reset occurs. This flag can be cleared only by writing a 1. Writing a 0 has no effect. 0 Bus Abort reset has not occurred 1 Bus Abort reset has occurred
4 JTR	JTAG reset flag. JTR is set to 1 when a JTAG reset occurs. This flag can be cleared only by writing a 1. Writing a 0 has no effect. 0 JTAG reset has not occurred 1 JTAG reset has occurred
3 SFR	Software reset flag. SFR is set to 1 when a software reset occurs. This flag can be cleared only by writing a 1. Writing a 0 has no effect. 0 Software reset has not occurred 1 Software reset has occurred
2 CMR	Clock monitor reset flag. CMR is set to 1 when a clock monitor reset occurs. This flag can be cleared only by writing a 1. Writing a 0 has no effect. 0 Clock Monitor reset has not occurred 1 Clock Monitor reset has occurred
1 EXR	External reset flag. EXR is set to 1 when an external reset occurs. This flag can be cleared only by writing a 1. Writing a 0 has no effect. 0 External reset has not occurred 1 External reset has occurred
0 WDR	Software Watchdog Timer (SWT) Reset Flag. WDR is set to 1 when an SWT reset occurs. This flag can only be cleared by writing a 1. Writing a 0 has no effect. This bit will not be valid if the Voltage Regulator is bypassed. 0 SWT reset has not occurred 1 SWT reset has occurred

NOTE

This flags are only valid if the POR and LVR flags are cleared after reset.

NOTE

Volatile memory may not be valid after a reset, regardless of the reset cause.

23.3.2.4 CRG Flags Register 2 (CRGFLG)

This register provides CRG status bits and flags.

Address: Base + $_03

Access: User read/write
Read: anytime
Write: refer to each bit for individual restrictions

	7	6	5	4	3	2	1	0
R	0	PORF	LVRF	LOCKIF	LOCK	TRACK	SCMIF	SCM
W								
Reset	0	1	2	0	0	0	0	0

1 PORF is set to 1 when a power on reset occurs. Unaffected by system reset.
2 LVRF is set to 1 when a low voltage reset occurs. Unaffected by system reset.

Figure 23-6. CRG Flag Register 2 (CRGFLG)

Table 23-4. CRGFLG Field Descriptions

Field	Description
7	Reserved, should be cleared.
6 PORF	Power on reset flag. PORF is set to 1 when a power on reset occurs. This flag can be cleared only by writing a 1. Writing a 0 has no effect. 0 Power on reset has not occurred 1 Power on reset has occurred **Note:** In order for the other reset source flags to be valid, this flag must be cleared after reset.
5 LVRF	Low voltage reset flag. LVRF is set to 1 when a low voltage reset occurs. This flag can only be cleared by writing a 1. Writing a 0 has no effect. This bit will not be valid if the Voltage Regulator is bypassed. 0 Low voltage reset has not occurred 1 Low voltage reset has occurred **Note:** In order for the other reset source flags to be valid, this flag must be cleared after reset.
4 LOCKIF	PLL lock interrupt flag. LOCKIF is set to 1 when the LOCK status bit changes. This flag can only be cleared by writing a 1. Writing a 0 has no effect. If enabled (LOCKIE=1), LOCKIF causes an interrupt request. 0 No change in LOCK bit 1 LOCK bit has changed
3 LOCK	Lock status bit. LOCK reflects the current state of PLL lock condition. This bit is cleared in Self Clock Mode. Writes have no effect. 0 PLL VCO is not within the desired tolerance of the target frequency 1 PLL VCO is within the desired tolerance of the target frequency
2 TRACK	Track status bit. TRACK reflects the current state of the PLL track condition. This bit is cleared in Self Clock Mode. Writes have no effect. 0 Acquisition mode status 1 Tracking mode status

Table 23-4. CRGFLG Field Descriptions (Continued)

Field	Description
1 SCMIF	Self clock mode interrupt flag. SCMIF is set to 1 when the SCM status bit changes. This flag can only be cleared by writing a 1. Writing a 0 has no effect. If enabled (SCMIE=1), SCMIF causes an interrupt request. 0 No change in SCM bit 1 SCM bit has changed
0 SCM	Self clock mode status bit. SCM reflects the current clocking mode. Writes have no effect. 0 MCU is operating normally with OSCCLK available 1 MCU is operating in Self Clock Mode with OSCCLK in an unknown state. All clocks are derived from PLLCLK running at its minimum frequency f_{SCM}

23.3.2.5 CRG Interrupt Enable Register (CRGINT)

This register enables CRG interrupt requests.

Address: Base + $_04

Access: User read/write
Read: anytime
Write: anytime

	7	6	5	4	3	2	1	0
R	0	0	0	LOCKIE	0	0	SCMIE	0
W								
Reset	0	0	0	0	0	0	0	0

Figure 23-7. CRG Interrupt Enable Register (CRGINT)

Table 23-5. CRGINT Field Descriptions

Field	Description
7–5	Reserved, should be cleared.
4 LOCKIE	Lock interrupt enable bit. 0 LOCK interrupt requests are disabled 1 Interrupt will be requested whenever LOCKIF is set
3–2	Reserved, should be cleared.
1 SCMIE	Self clock mode interrupt enable bit. 0 SCM interrupt requests are disabled 1 Interrupt will be requested whenever SCMIF is set
0	Reserved, should be cleared.

23.3.2.6 CRG Clock Select Register (CLKSEL)

This register controls CRG clock selection. Refer to Figure 23-13 for more details on the effect of each bit.

Address: Base + $_05

Access: User read/write
Read: anytime
Write: refer to each bit for individual write conditions

	7	6	5	4	3	2	1	0
R	PLLSEL	0	0	0	PLLDOZE	0	RTIDOZE	SWTDOZE
W								
Reset	0	0	0	0	0	0	0	0

Figure 23-8. CRG Clock Select Register (CLKSEL)

Table 23-6. CLKSEL Field Descriptions

Field	Description
7 PLLSEL	PLL select bit. Write anytime. Writing a 1 when LOCK=0 and AUTO=1, or TRACK=0 and AUTO=0 has no effect This prevents the selection of an unstable PLLCLK as SYSCLK. The PLLSEL bit is cleared when the MCU enters Self Clock Mode or Doze Mode with the PLLDOZE bit set. 0 System clocks are derived from OSCCLK (System Clock = OSCCLK) 1 System clocks are derived from PLLCLK (System Clock = PLLCLK)
6–4	Reserved, should be cleared.
3 PLLDOZE	PLL stops in doze mode bit. Write: anytime. If PLLDOZE is set, the CRG will clear the PLLSEL bit before entering Doze Mode. The PLLON bit remains set during Doze Mode, but the PLL is powered down. Upon exiting Doze Mode, the PLLSEL bit must be set manually if PLL clock is required. While the PLLDOZE bit is set, the AUTO bit is set to 1 in order to allow the PLL to lock automatically on to the selected target frequency after exiting Doze Mode. 0 PLL keeps running in Doze Mode. 1 PLL stops in Doze Mode.
2	Reserved, should be cleared.
1 RTIDOZE	RTI stops in doze mode bit. Write: anytime 0 RTI keeps running in Doze Mode 1 RTI stops and initializes the RTI dividers whenever the part goes into Doze Mode
0 SWTDOZE	SWT stops in doze mode bit. Normal modes: Write once. Special modes: Write anytime 0 SWT keeps running in Doze Mode 1 SWT stops and initializes the SWT dividers whenever the part goes into Doze Mode

23.3.2.7 CRG PLL Control Register (PLLCTL)

This register controls the PLL functionality.

Address: Base + $_06

Access: User read/write
Read: anytime
Write: refer to each bit for individual write conditions

	7	6	5	4	3	2	1	0
R	CME	PLLON	AUTO	ACQ	0	0	0	CMRD
W								
Reset	1	1	1	1	0	0	0	1

Figure 23-9. CRG PLL Control Register (PLLCTL)

Table 23-7. PLLCTL Field Descriptions

Field	Description
7 CME	Clock monitor enable bit. CME enables the clock monitor. Writeable anytime except when SCM = 1. 0 Clock monitor is disabled 1 Clock monitor is enabled. Slow or stopped clocks will cause a clock monitor reset sequence or Self Clock Mode **Note:** Operating with CME=0 will not detect any loss of clock. In cases of poor clock quality, this could cause unpredictable operation of the MCU.
6 PLLON	Phase lock loop on bit. PLLON turns on the PLL circuitry. In Self Clock Mode, the PLL is turned on, but the PLLON bit reads the last latched value. Write anytime except when PLLSEL = 1. 0 PLL is turned off 1 PLL is turned on. If AUTO bit is set, the PLL will lock automatically
5 AUTO	Automatic bandwidth control bit. AUTO selects either the high bandwidth (acquisition) mode or the low bandwidth (tracking) mode depending on how close to the desired frequency the VCO is running. Write anytime except when PLLDOZE=1, because PLLDOZE sets the AUTO bit to 1. 0 Automatic Mode Control is disabled and the PLL is under software control, using the ACQ bit 1 Automatic Mode Control is enabled and the ACQ bit has no effect
4 ACQ	Acquisition bit. Write anytime. If AUTO=1, writing to this this bit has no effect., but reading this bit indicates which bandwidth filter is selected 0 Low bandwidth filter is selected (Acquisition Mode) 1 High bandwidth filter is selected (Tracking Mode)
3–1	Reserved, should be cleared.
0 CMRD	Clock monitor reset disable bit. Normal modes: Write once. Special modes: Write anytime. CMRD cannot be cleared while operating in Self Clock Mode (SCM=1). 0 Detection of crystal clock failure causes clock monitor reset (see Section 23.4.5.4, “Clock Monitor Reset”) 1 Detection of crystal clock failure forces the MCU into Self Clock Mode (see Section 23.4.3.2, “Self Clock Mode”)

23.3.2.8 CRG DOZE Control Register (SDMCTL)

This register controls the DOZE mode transitioning.

Address: Base + $_07

Access: User read/write
Read: anytime
Write: anytime

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

Figure 23-10. CRG DOZE Control Register (SDMCTL)

Table 23-8. SDMCTL Field Descriptions

Field	Description
7–2	Reserved, should be cleared.
1 DOZE	Doze control bit. This bit determines whether the CRG should transition into DOZE mode. When the CRG receives a wakeup signal, the system will leave DOZE mode and this bit will be cleared. 0 Keep running 1 Activate DOZE sequence
0	Reserved, should be cleared.

23.3.2.9 CRG BDM Control Register (BDMCTL)

This register controls the SWT (Software Watchdog Timer) and RTI clocks in BDM mode.

Address: Base + $_08

Access: User read/write
Read: anytime
Write: RSBCK: once

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

Figure 23-11. CRG BDM Control Register (BDMCTL)

Table 23-9. BDMCTL Field Descriptions

Field	Description
7	Reserved, should be cleared.
6 RSBCK	SWT and RTI stop in active BDM mode (Debug Mode) bit 0 Allows the SWT and RTI to keep running in Active BDM mode 1 Stops the SWT and RTI counters whenever the part is in Active BDM mode
5–0	Reserved, should be cleared.

23.4 Functional Description

23.4.1 General

This section gives detailed information on the internal operation of the design.

23.4.2 Functional Blocks

23.4.2.1 Phase Locked Loop (PLL)

The PLL is used to run the MCU from a different time base than the incoming OSCCLK. For increased flexibility, OSCCLK can be divided in a range of 1 to 16 to generate the reference frequency. This offers

a finer multiplication granularity. The PLL can multiply this reference clock by a multiple of 2, 4, 6,... 126 or 128 based on the SYNR register.

$$PLLCLK = 2 \times OSCCLK \times \frac{SYNR + 1}{REFDV + 1} \qquad \textit{Eqn. 23-2}$$

CAUTION

Although it is possible to choose the two parameters to set a very high clock frequency, do not exceed the specified frequency limit for the MCU. If (PLLSEL=1), System Clock = PLLCLK.

The PLL is a frequency generator that operates in either acquisition mode or tracking mode, depending on the difference between the output frequency and the target frequency. The PLL can change between acquisition and tracking modes either automatically or manually.

The VCO has a minimum operating frequency, which corresponds to the self clock mode frequency f_{SCM}.

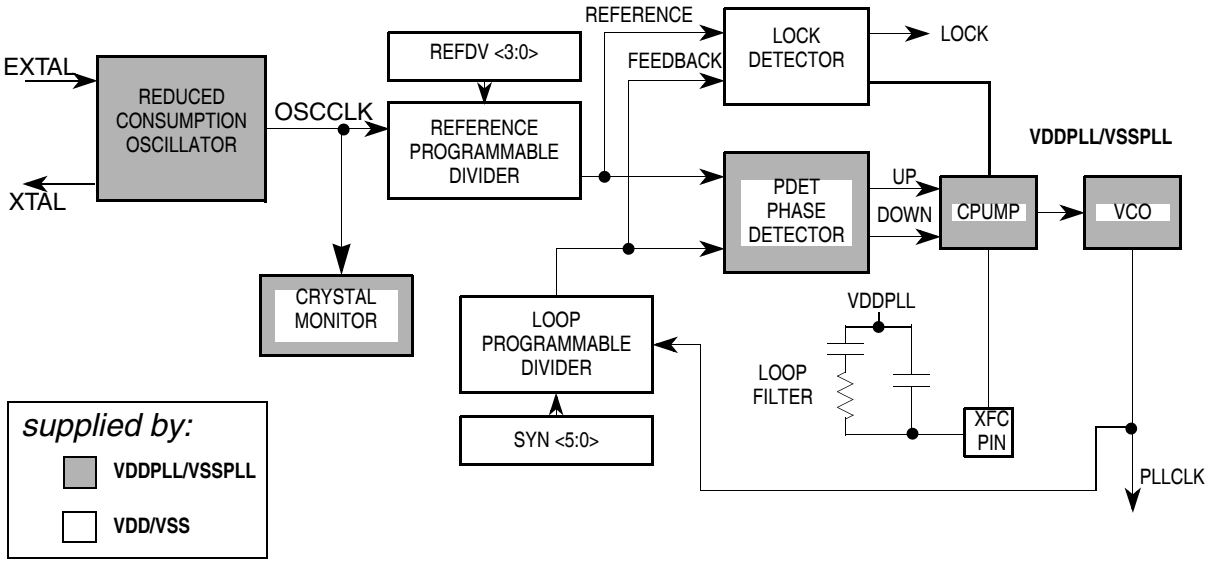


Figure 23-12. PLL Functional Diagram

23.4.2.1.1 PLL Operation

The oscillator output clock signal (OSCCLK) is fed through the reference programmable divider and is divided in a range of 1 to 16 (REFDV+1) to output the REFERENCE clock. The VCO output clock, (PLLCLK) is fed back through the programmable loop divider and is divided in a range of 2 to 128 in increments of [2 x (SYNR +1)] to output the FEEDBACK clock. Figure 23-12.

The phase detector then compares the FEEDBACK clock with the REFERENCE clock. Correction pulses are generated based on the phase difference between the two signals. The loop filter then slightly alters the DC voltage on the external filter capacitor connected to XFC pin, based on the width and direction of the correction pulse. The filter can make fast or slow corrections depending on its mode, as described in the next subsection. The values of the external filter network and the reference frequency determine the speed of the corrections and the stability of the PLL.

23.4.2.1.2 Acquisition and Tracking Modes

The lock detector compares the frequencies of the FEEDBACK clock and the REFERENCE clock. Therefore, the speed of the lock detector is directly proportional to the final reference frequency. The circuit determines the mode of the PLL and the lock condition based on this comparison.

The PLL filter can be manually or automatically configured into one of two possible operating modes:

- Acquisition mode

  In acquisition mode, the filter can make large frequency corrections to the VCO. This mode is used at PLL start-up or when the PLL has suffered a severe noise hit and the VCO frequency is far from the desired frequency. When in acquisition mode, the $\overline{\text{TRACK}}$ status bit is cleared in the CRGFLG register.
- Tracking mode

  In tracking mode, the filter makes only small corrections to the frequency of the VCO. PLL jitter is much lower in tracking mode, but the response to noise is also slower. The PLL enters tracking mode when the VCO frequency is nearly correct and the TRACK bit is set in the CRGFLG register.

The PLL can change the bandwidth or operational mode of the loop filter manually or automatically.

In automatic bandwidth control mode (AUTO = 1), the lock detector automatically switches between acquisition and tracking modes. Automatic bandwidth control mode is also used to determine when the PLL clock (PLLCLK) is safe to use as the source for the system and peripheral clocks. If the PLL LOCK interrupt requests are enabled, the software can wait for an interrupt request and then check the LOCK bit. If CPU interrupts are disabled, software can poll the LOCK bit continuously (usually during PLL start-up) or at periodic intervals. In either case, only when the LOCK bit is set is the PLLCLK clock safe to use as the source for the system and peripheral clocks. If the PLL is selected as the source for the system and peripheral clocks and the LOCK bit is clear, the PLL has suffered a severe noise hit and the software must take appropriate action, depending on the application.

The following conditions apply when the PLL is in automatic bandwidth control mode (AUTO=1):

- The $\overline{\text{TRACK}}$ bit is a read-only indicator of the mode of the filter.
- The $\overline{\text{TRACK}}$ bit is set when the VCO frequency is within a certain tolerance, Δ_{trk}, and is clear when the VCO frequency is out of a certain tolerance, Δ_{unt}.
- The LOCK bit is a read-only indicator of the locked state of the PLL.
- The LOCK bit is set when the VCO frequency is within a certain tolerance, Δ_{Lock}, and is cleared when the VCO frequency is out of a certain tolerance, Δ_{unl}.
- CPU interrupts can occur if enabled (LOCKIE = 1) when the lock condition changes, toggling the LOCK bit.

The PLL can also operate in manual mode (AUTO = 0). Manual mode is used by systems that do not require an indicator of the lock condition for proper operation. Such systems typically operate well below the maximum system frequency (f_{sys}) and require fast start-up. The following conditions apply when in manual mode:

- $\overline{\text{ACQ}}$ is a writable control bit that controls the mode of the filter. Before turning on the PLL in manual mode, the $\overline{\text{ACQ}}$ bit should be asserted to configure the filter in acquisition mode.

- After turning on the PLL by setting the PLLON bit, software must wait a given time (t_{acq}) before entering tracking mode (ACQ = 0).
- After entering tracking mode, software must wait a given time (t_{al}) before selecting the PLLCLK as the source for system and peripheral clocks (PLLSEL = 1).

23.4.2.2 System Clocks Generator

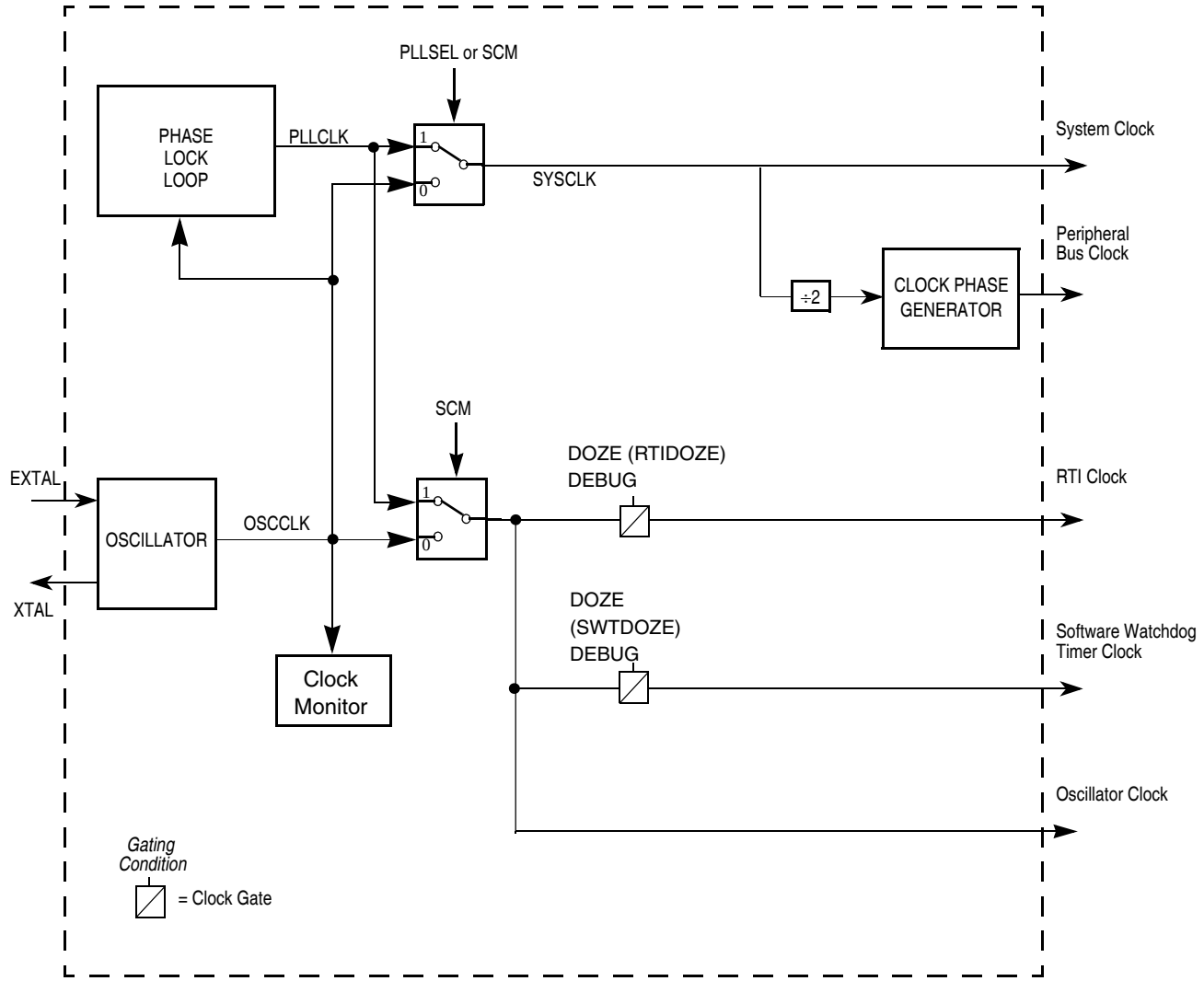


Figure 23-13. System Clocks Generator

The clock generator creates the clocks used in the MCU (see Figure 23-13). The gating condition placed on top of the individual clock gates indicates the dependencies of different modes (DOZE, DEBUG) and the setting of the respective configuration bits.

The CPU uses the System Clock. The peripherals and memory blocks use the Peripheral Bus Clock. Some peripheral modules also use the Oscillator Clock. If the MCU enters Self Clock Mode (see Section 23.4.3.2, “Self Clock Mode”) the Oscillator clock source is switched to PLLCLK running at its

minimum frequency, f_{SCM}. The System Clock is twice the Peripheral Bus Clock as shown in Figure 23-14. Note that a CPU cycle corresponds to one Peripheral Bus Clock.

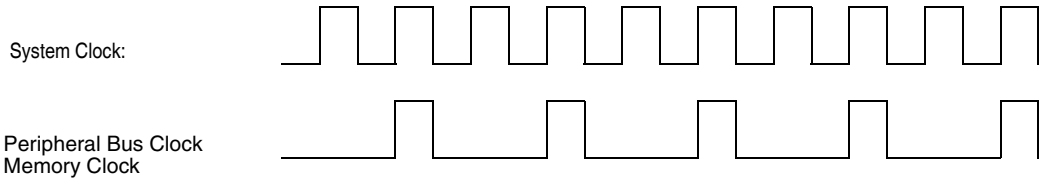


Figure 23-14. System Clock and Peripheral Bus Clock Relationship

PLL clock mode is selected with the PLLSEL bit in the CLKSEL register. When selected, the PLL output clock drives SYSCLK for the main system, including the CPU and peripherals. The PLL cannot be turned off by clearing the PLLON bit if the PLL clock is selected. When PLLSEL is changed, it takes a maximum of 4 OSCCLK plus 4 PLLCLK cycles to make the transition. During the transition, all clocks freeze and CPU activity ceases.

23.4.2.3 Clock Monitor (CM)

If no OSCCLK edges are detected within a certain time, the clock monitor within the oscillator block generates a clock monitor fail event. The CRG then asserts self clock mode or generates a system reset depending on the state of the CMRD bit. If the clock monitor is disabled or the presence of clocks is detected, no failure is indicated by the oscillator block. The clock monitor function is enabled/disabled by the CME control bit.

23.4.2.4 Clock Quality Checker

The clock monitor performs a coarse check on the incoming clock signal. The clock quality checker provides a more accurate check in addition to the clock monitor.

A clock quality check is triggered by any of the following events:

- Power on reset (POR)
- Low voltage reset (LVR)
- Clock Monitor fail indication (CM fail)

A time window of 50000 VCO clock cycles[1] is called *check window.*

A number greater than or equal to 4096 rising OSCCLK edges within a *check window* is called *osc ok*. Note that *osc ok* immediately terminates the current *check window.* See Figure 23-15 as an example.

1. VCO clock cycles are generated by the PLL when running at minimum frequency f_{SCM}.

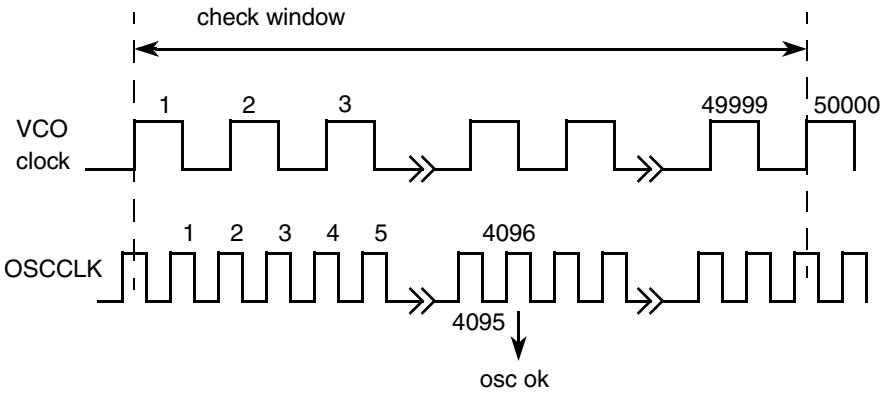


Figure 23-15. Check Window Example

The sequence for clock quality check is shown in Figure 23-16.

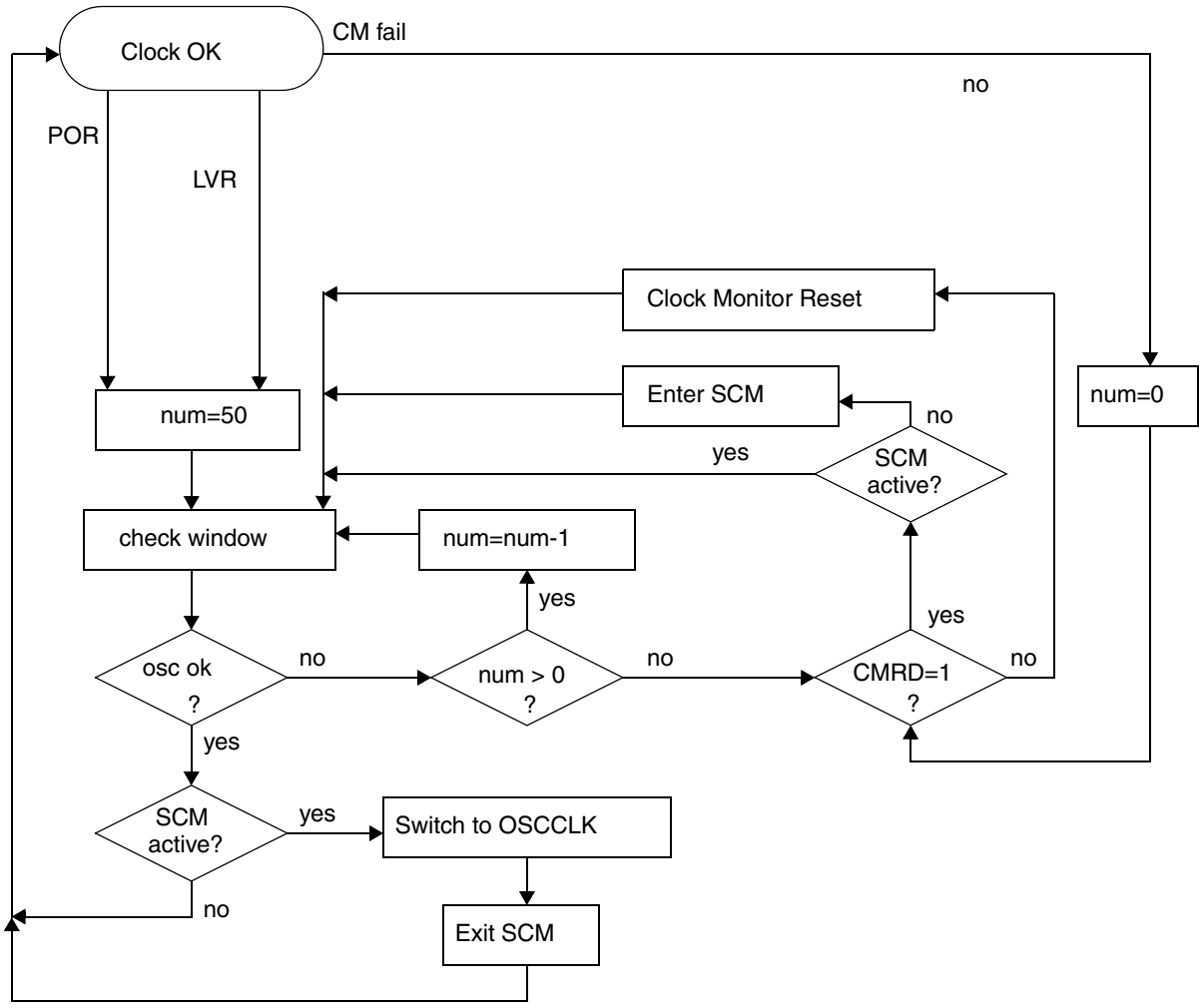


Figure 23-16. Sequence for Clock Quality Check

NOTE

Remember that in parallel to additional actions caused by Self Clock Mode or Clock Monitor Reset[1] handling, the clock quality checker **continues** to check the OSCCLK signal.

NOTE

The Clock Quality Checker enables the PLL and the voltage regulator (VREG) anytime a clock check has to be performed. An ongoing clock quality check could also cause a running PLL (f_{SCM}) and an active VREG during Doze Mode

23.4.2.5 Software Watchdog Timer (SWT)

The SWT (free running watchdog timer) allows the user to check that a program is running and sequencing properly. This watchdog timer resides on the ARM platform. However, the CRG module provides the clock for the watchdog timer.

23.4.2.6 Real Time Interrupt (RTI)

The RTI (real time interrupt) is in a separate module. However, the CRG module provides a special RTI clock. This is a gated OSCCLK.

23.4.3 Operating Modes

23.4.3.1 Normal Mode

The CRG block behaves as described within this specification in all normal modes.

23.4.3.2 Self Clock Mode

The VCO has a minimum operating frequency, f_{SCM}. If the external clock frequency is not available due to a failure or due to long crystal start-up time, the Peripheral Bus Clock and the System Clock are derived from the VCO running at the minimum operating frequency; this mode of operation is called Self Clock Mode. This requires CME=1 and CMRD=1. If the MCU was clocked by the PLL clock prior to entering Self Clock Mode, the PLLSEL bit will be cleared. If the external clock signal has stabilized again, the CRG will automatically select OSCCLK to be the system clock and return to normal mode. Section 23.4.2.4, “Clock Quality Checker,” for more information on entering and leaving Self Clock Mode.

NOTE

In order to detect a potential clock loss, the CME bit should be always enabled (CME=1).

1. A Clock Monitor Reset will always set the CMRD bit to logical’1’

If the CME bit is disabled and the MCU is configured to run on the PLL clock (PLLCLK), a loss of external clock (OSCCLK) will not be detected and will cause the system clock to drift towards the VCO's minimum frequency, f_{SCM}. As soon as the external clock is available again, the system clock ramps up to its PLL target frequency. If the MCU is running on an external clock, any loss of clock will cause the system to go static.

23.4.4 Low Power Options

This section summarizes the low power options available in the CRG.

23.4.4.1 Run Mode

This is the standard mode, all components of the system are clocked.

23.4.4.2 Doze Mode

Writing to the Doze bit puts the system in a low power consumption stand-by mode further controlled by the CLKSEL register settings. This provides enhanced granularity in reducing the level of power consumption. Table 23-10 lists the individual configuration bits and the parts of the MCU that are affected in Doze Mode.

Table 23-10. MCU Configuration During Doze Mode

	PLLDOZE	RTIDOZE	SWTDOZE	
PLL	stopped	–	–	
RTI Clock	–	stopped	–	
SWT Clock	–	–	stopped	

The core requests the CRG to switch into Doze Mode by writing to the DOZE bit. The CRG then checks whether the PLLDOZE bit is asserted (Figure 23-17). Depending on the configuration, the CRG switches the system clock to OSCCLK by clearing the PLLSEL bit and disables the PLL.

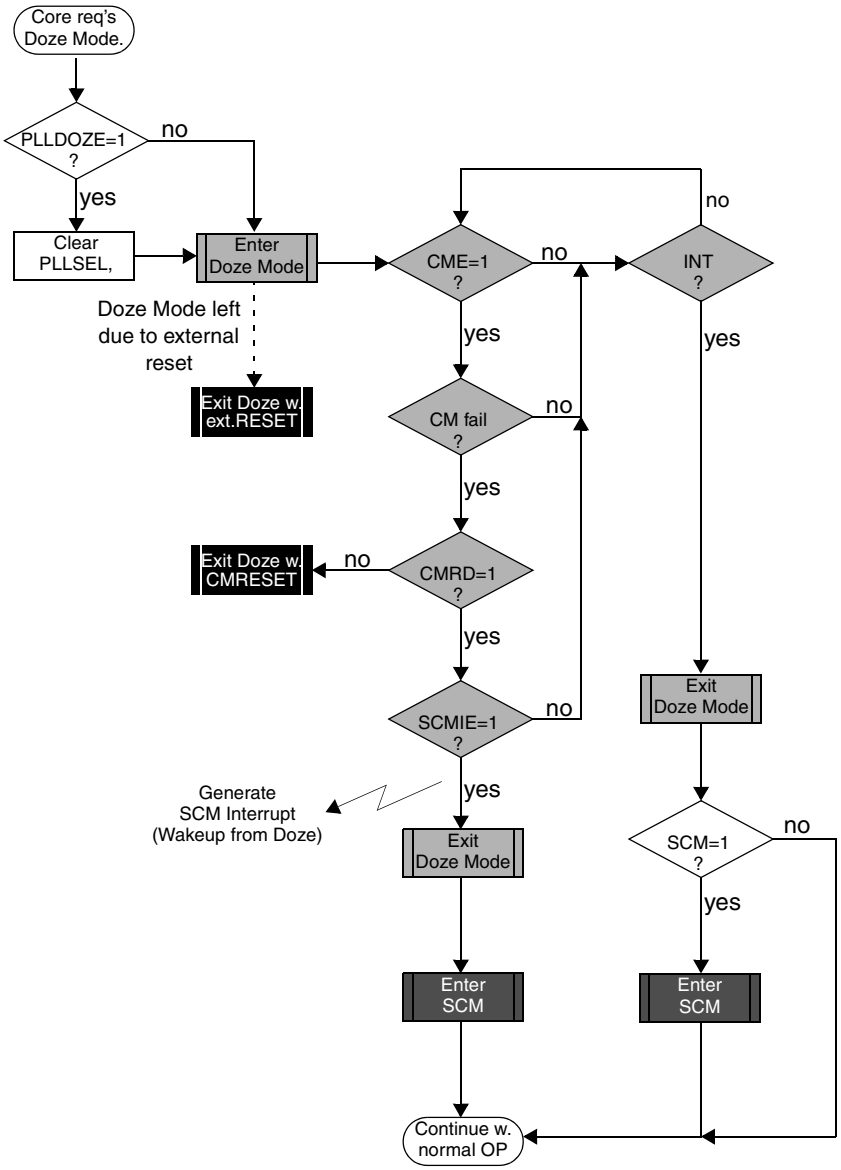


Figure 23-17. Doze Mode Entry/Exit Sequence

There are five different scenarios for the CRG to restart the MCU from Doze Mode:

- External Reset
- Clock Monitor Reset
- SWT Reset
- Self Clock Mode Interrupt
- Wake-Up Interrupt (e.g. RTI)

If the MCU receives an external reset while in Doze Mode, the CRG asynchronously restores all configuration bits in the register space to their default settings and starts the reset generator. After

completing the reset sequence, processing begins by fetching the normal reset vector. Doze Mode is exited and the MCU is in Run Mode again.

If the clock monitor is enabled (CME=1) the MCU is able to exit Doze-Mode when loss of oscillator/external clock is detected by a clock monitor fail. If the CMRD bit is not asserted, the CRG generates a clock monitor fail reset (CMRESET). The CRG's behavior for CMRESET is the same compared to external reset, but another reset vector is fetched after completion of the reset sequence. If the CMRD bit is asserted, the CRG generates an SCM interrupt if enabled (SCMIE=1). After generating the interrupt, the CRG enters Self-Clock Mode and starts the clock quality checker (Section 23.4.2.4, "Clock Quality Checker"). Then the MCU continues with normal operation. If the SCM interrupt is blocked by SCMIE=0, the SCMIF flag will be asserted and clock quality checks will be performed, but the MCU will not wake-up from Doze-Mode.

If any other interrupt source (e.g. RTI) triggers the exit from Doze Mode, the MCU immediately continues with normal operation. If the PLL has been powered-down during Doze-Mode, the PLLSEL bit is cleared and the MCU runs on OSCCLK after leaving Doze-Mode. The software must manually set the PLLSEL bit again, in order to switch system and peripheral clocks to the PLLCLK.

If Doze Mode is entered from Self-Clock Mode, the CRG will continue to check the clock quality until the clock check is successful. The PLL and voltage regulator (VREG) will remain enabled.

Table 23-11 summarizes the outcome of a clock loss while in Doze Mode.

Table 23-11. Outcome of Clock Loss in Doze Mode

CME	CMRD	SCMIE	CRG Actions
0	X	X	Clock failure → No action, clock loss not detected.
1	0	X	Clock failure → CRG performs Clock Monitor Reset immediately

Table 23-11. Outcome of Clock Loss in Doze Mode (Continued)

CME	CMRD	SCMIE	CRG Actions
1	1	0	Clock failure → Scenario 1: OSCCLK **recovers** prior to exiting Doze Mode – MCU remains in Doze Mode – VREG enabled – PLL enabled – SCM activated – Start Clock Quality Check – Set SCMIF interrupt flag *Some time later OSCCLK recovers* – CM no longer indicates a failure – 4096 OSCCLK cycles later Clock Quality Check indicates clock ok – SCM deactivated – PLL disabled depending on PLLDOZE – VREG remains enabled *(never gets disabled in Doze Mode)* – MCU remains in Doze Mode *Some time later either a wakeup interrupt occurs (no SCM interrupt)* – Exit Doze Mode using OSCCLK as system clock (SYSCLK) – Continue normal operation *or an External Reset is applied* – Exit Doze Mode using OSCCLK as system clock – Start reset sequence Scenario 2: OSCCLK **does not recover** prior to exiting Doze Mode – MCU remains in Doze Mode – VREG enabled – PLL enabled – SCM activated – Start Clock Quality Check – Set SCMIF interrupt flag – Keep performing Clock Quality Checks (could continue infinitely) while in Doze Mode *Some time later either a wakeup interrupt occurs (no SCM interrupt)* – Exit Doze Mode in SCM using PLL clock (f_{SCM}) as system clock – Continue to perform additional Clock Quality Checks until OSCCLK is ok again or an External RESET is applied – Exit Doze Mode in SCM using PLL clock (f_{SCM}) as system clock – Start reset sequence – Continue to perform additional Clock Quality Checks until OSCCLK is ok again

Table 23-11. Outcome of Clock Loss in Doze Mode (Continued)

CME	CMRD	SCMIE	CRG Actions
1	1	1	Clock failure → – VREG enabled – PLL enabled – SCM activated – Start Clock Quality Check – SCMIF set SCMIF generates Self Clock Mode wakeup interrupt – Exit Doze Mode in SCM using PLL clock (f_{SCM}) as system clock – Continue to perform a additional Clock Quality Checks until OSCCLK is ok again

23.4.5 Resets

Table 23-12. Entering CRG Modes

Mode	Sequence
Doze	– Doze register bit in the CRG is written – CRG indicates Doze mode – The system will turn off peripherals' clocks according to their DOZE bits – RTI and SWT clocks will be turned off according to their DOZE bits

23.4.5.1 General

This section describes how to reset the CRG and how the CRG controls the reset of the MCU. It explains all special reset requirements. Since the reset generator for the MCU is part of the CRG, this section also describes all automatic actions that occur during or as a result of individual reset conditions. The reset values of registers and signals are provided in Section 23.3, "Memory Map and Register Definition." All reset sources are listed in Table 23-13. Refer to the MCU specification for related vector addresses and priorities.

Table 23-13. Reset Summary

Reset Source	Local Enable
Power on Reset	None
Low Voltage Reset	None
External Reset	None
Clock Monitor Reset	PLLCTL (CME=1, CMRD=0)
SWT Watchdog Reset	None

23.4.5.2 Description of Reset Operation

The reset sequence is initiated by any of the following events:

- Low level is detected at the $\overline{\text{RESET}}$ pin (External Reset)

- Power on is detected
- Low voltage is detected
- SWT watchdog times out
- Clock monitor failure is detected and Self-Clock Mode was disabled (CMRD=0)

Upon detection of any reset event, an internal circuit drives the RESET pin low for 256 SYSCLK cycles (see Figure). Since entry into reset is asynchronous, it does not require a running SYSCLK. However, the internal reset circuit of the CRG cannot sequence out of the current reset condition without a running SYSCLK. The number of 256 SYSCLK cycles might be increased by n=3 to 6 additional SYSCLK cycles depending on the internal synchronization latency. After 256+n SYSCLK cycles the RESET pin is released. The reset generator of the CRG waits for an additional 64 SYSCLK cycles and then samples the RESET pin to determine the originating source. Figure 23-14 shows which vector will be fetched.

Table 23-14. Reset Vector Selection

sampled RESET pin (64 cycles after release)	Clock Monitor Reset pending	SWT Reset pending	Vector fetch
1	0	0	POR / LVR / External Reset
1	1	X	Clock Monitor Reset
1	0	1	SWT Reset
0	X	X	POR / LVR / External Reset with rise of RESET pin

NOTE

External circuitry connected to the RESET pin should not include a large capacitance that would interfere with the ability of this signal to rise to a valid logic one within 64 SYSCLK cycles after the low drive is released.

The internal reset of the MCU remains asserted while the reset generator completes the 320 SYSCLK long reset sequence. The reset generator circuitry always makes sure the internal reset is deasserted synchronously after completion of the 320 SYSCLK cycles. When the RESET pin is externally driven low for more than these 320 SYSCLK cycles (External Reset), the internal reset also remains asserted.

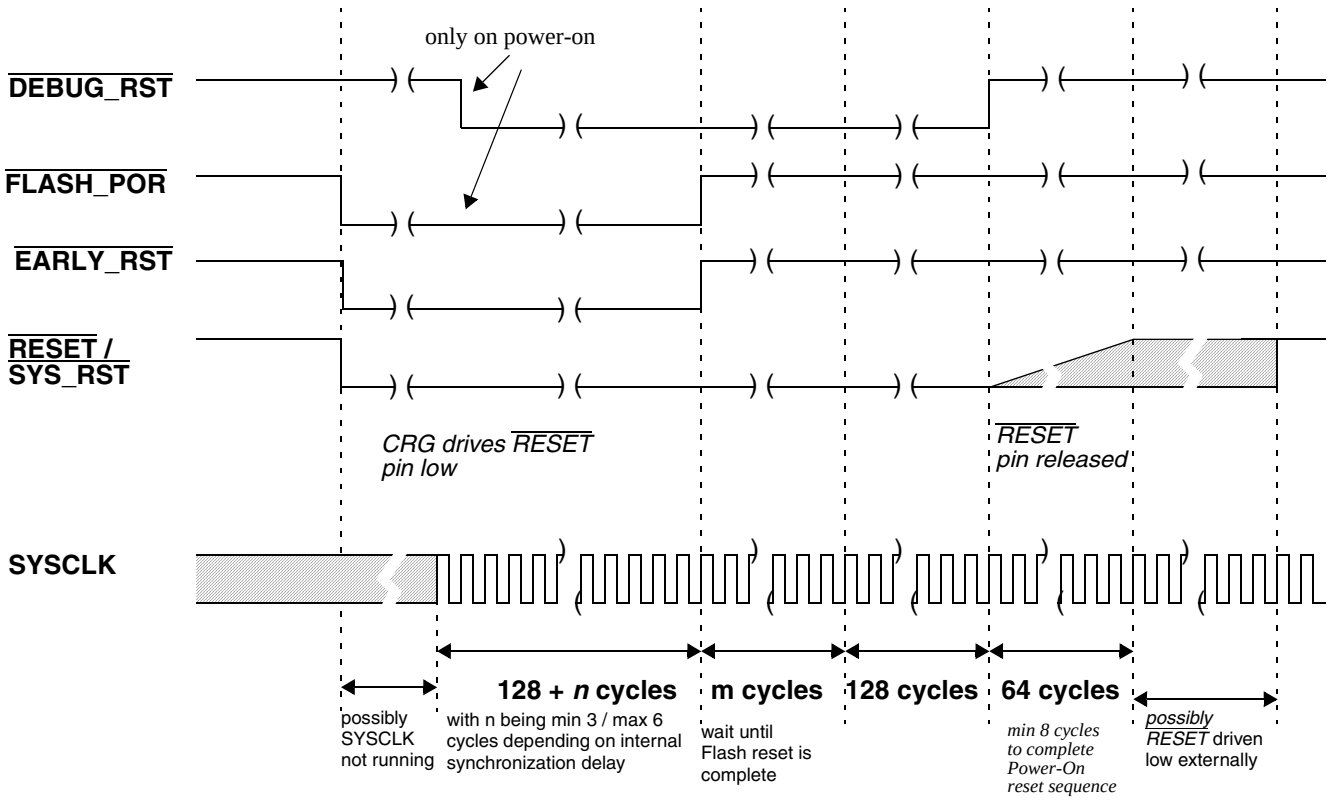


Figure 23-18. RESET Timing

The debug block reset is asserted together with the internal reset, but will not be extended by the external $\overline{\text{RESET}}$ pin. In addition, the debug block reset will be asserted only if the reset sequence was initiated by a power-on or low-voltage indication.

The $\overline{\text{Flash\_POR}}$ signal is asserted on POR or LVR conditions. It will be synchronously released after 128+n clock cycles.

The reset sequence will be extended by *m* cycles till the Flash block acknowledges that it has completed its reset. The number m will depend on the external crystal as well as on the characteristics of the Flash. It will vary for each device.

This enables the user to activate the debugger in two ways: either by holding the reset input asserted while the debugger is being activated or by setting up the debugger after a power-on reset, then applying an additional reset that will not affect the debug logic (i.e. not a power-on reset).

Note that the Power-On reset sequence is not complete until least 8 cycles after the CRG releases the $\overline{\text{RESET}}$ line. If another external reset occurs before the 8 cycles have expired, that new reset will also be treated as a Power-On-Reset, so DEBUG_RESET will be asserted again.

23.4.5.3 JTAG Reset

The DEBUG_RESET line can also be asserted via a JTAG command (see Figure 23-19). This causes a much shorter reset sequence, solely for the DEBUG_RESET - other parts of the device are not affected.

NOTE

The JTAG reset sequence can not be mixed with another reset sequence.

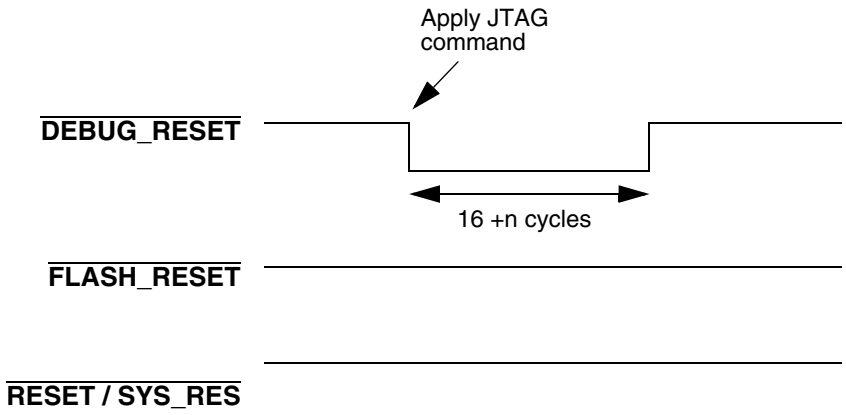


Figure 23-19. RESET Timing controlled by JTAG

23.4.5.4 Clock Monitor Reset

The CRG generates a Clock Monitor Reset when all of the following conditions are true:

- Clock monitor is enabled (CME=1)
- Loss of clock is detected
- Self-Clock Mode is disabled (CMRD=0)

The reset event asynchronously forces the configuration registers to their default settings (see Section 23.3, “Memory Map and Register Definition).” The CME and the CMRD are reset to logical ‘1’ (which doesn’t change the state of the CME bit, because it has already been set). As a consequence, the CRG immediately enters Self Clock Mode and starts its internal reset sequence. In parallel, the clock quality check starts. As soon as the clock quality check indicates a valid Oscillator Clock, the CRG switches to OSCCLK and leaves Self Clock Mode. Since the clock quality checker is running in parallel with the reset generator, the CRG may leave Self Clock Mode while still completing the internal reset sequence. When the reset sequence is finished, the CRG checks the internally latched state of the clock monitor fail circuit.

23.4.5.5 Software Watchdog Timer (SWT) Reset

The CRG will generate a reset if the Watchdog reset is indicated by the ARM platform.

23.4.5.6 Power On Reset, Low Voltage Reset

The on-chip voltage regulators detect when VDDs to the MCU have reached a certain level and assert a power on reset or low voltage reset or both. As soon as a power on reset or low voltage reset is triggered,

the CRG performs a quality check on the incoming clock signal. As soon as clock quality check indicates a valid Oscillator Clock signal, the reset sequence starts using the Oscillator clock. If after 50 check windows the clock quality check indicates a non-valid Oscillator Clock, the reset sequence starts using Self-Clock Mode.

Figure 23-20 and Figure 23-21 show the power-up sequence for cases when the $\overline{\text{RESET}}$ pin is tied to VDD and when the $\overline{\text{RESET}}$ pin is held low.

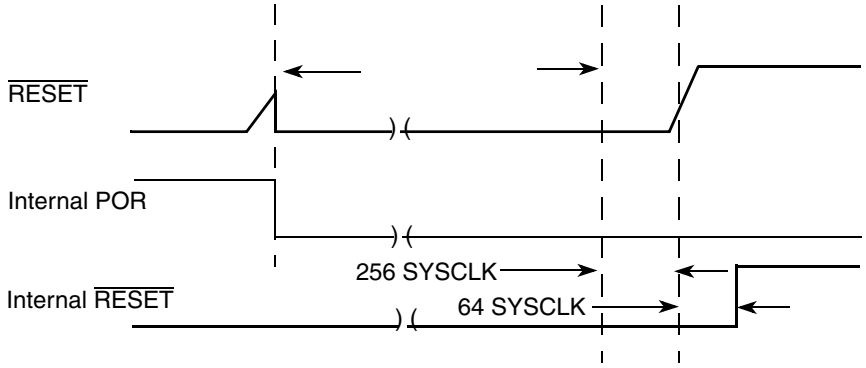


Figure 23-20. $\overline{\text{RESET}}$ Pin Tied to VDD (by a pull-up resistor)

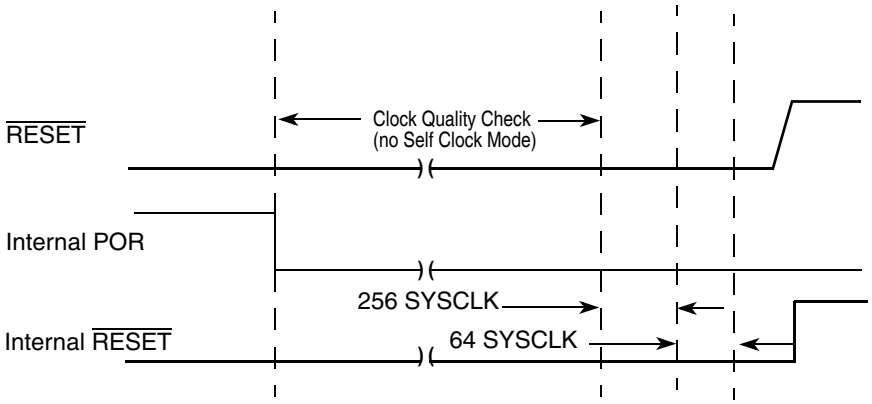


Figure 23-21. $\overline{\text{RESET}}$ Pin Held Low Externally

23.4.6 Interrupts

23.4.6.1 General

The interrupts/reset vectors requested by the CRG are listed in Figure 23-15. Refer to the MCU specification for related vector addresses and priorities.

Table 23-15. CRG Interrupt Vectors

Interrupt Source	CCR Mask	Local Enable
LOCK interrupt	I bit	CRGINT (LOCKIE)
SCM interrupt	I bit	CRGINT (SCMIE)

23.4.6.2 PLL Lock Interrupt

The CRG generates a PLL Lock interrupt when the LOCK condition of the PLL has changed, either from a locked state to an unlocked state or vice versa. Lock interrupts are locally disabled by clearing the LOCKIE bit to zero. The PLL Lock interrupt flag (LOCKIF) is set to 1 when the LOCK condition has changed and is cleared to 0 by writing a 1 to the LOCKIF bit.

23.4.6.3 Self Clock Mode Interrupt

The CRG generates a Self Clock Mode interrupt when the SCM condition of the system has changed; either entered or exited Self Clock Mode. SCM conditions can only change if the Clock Monitor Reset Disable (CMRD) bit is set to 1. SCM conditions are caused by a failing clock quality check after power on reset (POR) or low voltage reset (LVR) or Clock Monitor failure. For details on the clock quality check, refer to Section 23.4.2.4, “Clock Quality Checker.” If the clock monitor is enabled (CME=1) a loss of external clock will also cause a SCM condition (CMRD=1).

SCM interrupts are disabled locally by setting the SCMIE bit to zero. The SCM interrupt flag (SCMIF) is set to1 when the SCM condition has changed, and is cleared to 0 by writing a 1 to the SCMIF bit.

23.5 CRG Bus Aborts

The CRG module supports Peripheral Bus bus aborts, and enforces the following memory map:

Table 23-16. CRG Bus Aborts

Abort	Allowed	
	$0000-$000b	
$000c-$3fff		

Supervisor Access: Unused.

23.6 CRG Differences from MAC71xx

- Fixed MUCts01648: Write on bus abort still writes the register
- Added single register with all possible reset sources
  - — Added the ILR, JTR, SFR, CMR, EXR and WDR bits to the CTFLG register
- Removed Pseudo-STOP/STOP modes (DOZE still implemented)
  - — Removed the STPEF bit from the CRGFLG register
  - — Removed the PSTP bit from the CLKSEL register

  - Removed the FSTWKP, PRE and PWE bits from the PLLCTL register
  - Removed the STOP bit from the SDMCTL register
- Renamed the SCME bit in the PLLCTL register to CMRD (no functionality change)
- Removed the DOZE_ROA bit from the CLKSEL register (The oscillator does not support this feature)
- Changed the latching of the oscillator mode (**XCLKS**)
  - The oscillator mode (**XCLKS**) is latched only during a power-on reset and during a reset while a loss of clock has occurred. It is no longer latched during a normal system reset.
- XFC changed from 2.5V to 3.3V signal
- Changed max PLL frequency from 40MHz to 70MHz. Minimum VCO lock changed to 30MHz
- PLL Loop filter calculations in the documentation *may* be revised slightly

23.7 CRG Application Usage

23.7.1 Enabling the CRG and PLL

It is not necessary to enable the CRG before it can be used.

Note that the PLL is bypassed when the device comes out of reset. It is the responsibility of the boot code to configure and enable the PLL. This should be done as early in the boot process as possible (after the PLL has locked) in order to minimize the time required for boot.

23.7.2 Crystal Monitor

The MAC72xx provides an on-chip Crystal Monitor that detects a loss of Oscillator Clock. If no OSCCLK edges are detected within a certain time, referred to as the Crystal Monitor Timeout, the Crystal Monitor within the oscillator block generates a Crystal Monitor fail event. Depending on the state of the **CMRD** bit in the **PLLCTL** register and the **SCMIE** bit in the **CRGINT** register, one of several events can occur when the Crystal Monitor detects a loss of Oscillator Clock.

- The CRG generates a Crystal Monitor Reset, which causes a System Reset. The **CMR** reset status bit in the **CTFLG** register is set, and after the reset sequence has completed, may be read to determine that a Crystal Monitor Reset has occurred. (**CMRD** = 0)
- The system enters Self Clock Mode, the CRG raises an SCM interrupt, and the **SCMIF** and **SCM** bits in the **CRGFLG** register are set. (**CMRD** = 1, **SCMIE** = 1)
- The system enters Self Clock Mode and the **SCMIF** and **SCM** bits in the **CRGFLG** register are set.(**CMRD** = 1, **SCMIE** = 0)

If the **CME** bit in the **PLLCTL** register is cleared, then the Crystal Monitor is disabled, and the loss of Oscillator Clock becomes essentially undetectable in software. For this reason, it is highly recommended to enable this functionality. Note that the terms “Crystal Monitor” and “Clock Monitor” are used interchangeably in the MAC72xx documentation.

Chapter 24
Oscillator (OSC)

24.1 Introduction

The MAC7200 family of devices features an internal Automatic Level Control (ALC) oscillator. The oscillator is designed for optimal startup margin with typical crystal oscillators. Selection of the oscillator type is performed at Reset based on the value of the $\overline{\text{XCLKS}}$ pin.

After a Power on Reset (POR) or a system reset, the quality of the oscillation is checked by in the Clock and Reset Generator (CRG) using the Clock Quality Checker before the oscillator is connected to the internal system clocks. In the event that a stable oscillator output is not detected within a predefined time, the MCU will be switched to its internal self clock mode.

Oscillator power is supplied from its own 3.3V PLL supply voltage generated by the Voltage Regulator in order to minimize noise. The Oscillator continues to run in Doze mode.

A square wave input can be supplied to the device through the oscillator by connecting the external clock source to the EXTAL pin with the Oscillator operating in External Clock mode.

Consult Chapter 23, “Clock and Reset Generator (CRG) for more information about the operation of reset and clocks.

NOTE

The oscillators on the MAC71xx and MAC72xx are different designs. You should characterize your oscillator design if you switch between the two devices.

24.1.1 Features

The OSC_ALC_HIP7A contains the following features:

- Low RF emissions with peak to peak swing limited dynamically.
- Transconductance (gm) sized for optimum start-up margin for typical oscillators.
- Integrated resistor eliminates the need for external bias resistor.
- Low power consumption:
  - Operates from 3.3 V (nominal) supply
  - Amplitude control limits power
- Oscillator reference output (f_{osc} - ALC mode)
- External clock input (f_{osc} - External Clock mode)
- Clock monitor

24.1.2 Block diagram

Figure 24-1 shows a block diagram of the OSC_ALC_HIP7A.

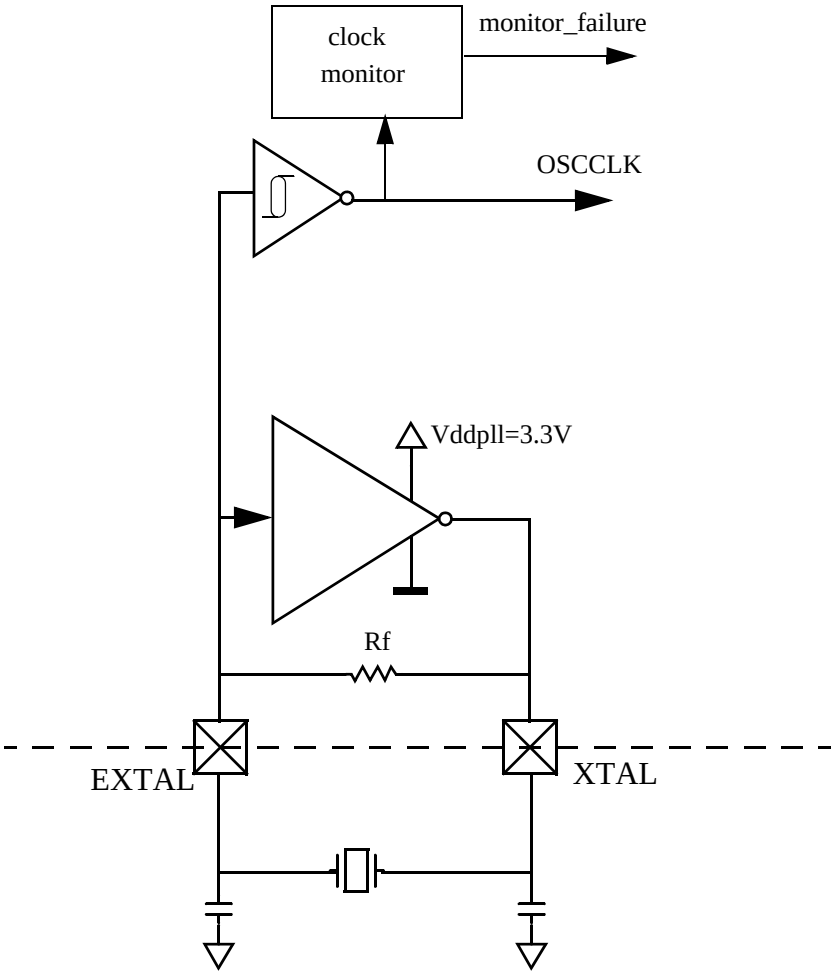


Figure 24-1. OSC_ALC_HIP7A Block Diagram

24.1.3 Modes of Operation

Two modes of operation exist:

- ALC oscillator mode
- External square wave mode

Selection between the modes is done at reset, using the $\overline{\text{XCLKS}}$ hardware configuration pin. Please refer to the MCU level documentation for a description of this pin.

24.2 External Signal Description

This section lists and describes the signals that connect off chip

24.2.1 V_{DDPLL}, V_{SSPLL}

These pins provides operating voltage (VDDPLL) and ground (VSSPLL) for the OSC_ALC_HIP7A circuitry. This allows the supply voltage to the OSC_ALC_HIP7A to be independently bypassed.

24.2.2 EXTAL, XTAL

These pins provide the interface for either a crystal or a CMOS compatible clock to control the internal clock generator circuitry. EXTAL is the external clock input or the input to the crystal oscillator amplifier. XTAL is the output of the crystal oscillator amplifier. The MCU internal system clock is derived from the EXTAL input frequency.

NOTE

Freescale recommends an evaluation of the application board and chosen resonator or crystal by the resonator or crystal supplier.

NOTE

Loop controlled circuit is not suited for overtone resonators and crystals.

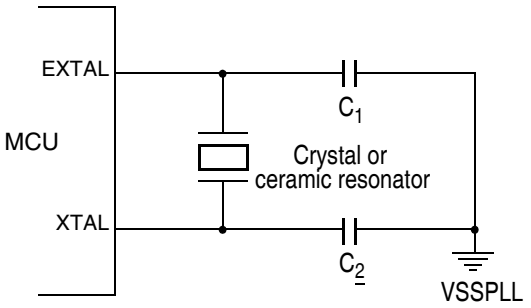


Figure 24-2. ALC Oscillator Connections ($\overline{XCLKS}$=1)

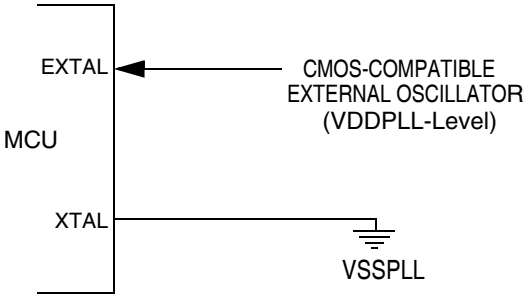


Figure 24-3. External Clock Connections ($\overline{XCLKS}$=0)

24.2.3 $\overline{XCLKS}$ (eXternal CLocK Select)

The $\overline{XCLKS}$ is an active low input signal which controls whether a crystal in combination with the internal loop controlled oscillator is used or whether external clock circuitry is used. The $\overline{XCLKS}$ signal is sampled during reset with the rising edge of $\overline{RESET}$. Table 24-1 lists the state coding of the sampled XCLKS signal.

Table 24-1. Clock Selection Based on $\overline{\text{XCLKS}}$

$\overline{\text{XCLKS}}$	Description
0	External Clock Mode (External 3.3V clock required)
1	ALC Mode (Internal oscillator clock)

24.3 Memory Map/Register Definition

The CRG contains the registers and associated bits for controlling and monitoring the oscillator module.

24.4 Functional Description

The OSC_ALC_HIP7A module has control circuitry to maintain the crystal oscillator circuit voltage level to an optimal level which is determined by the amount of hysteresis being used and the maximum oscillation range.

The oscillator block has two external pins, EXTAL and XTAL. The oscillator input pin, EXTAL, is intended to be connected to either a crystal or an external clock source. The selection of ALC oscillator mode or external clock depends on the $\overline{\text{XCLKS}}$ signal which is sampled during reset. The XTAL pin is an output signal that provides crystal circuit feedback.

A buffered EXTAL signal becomes the internal clock. To improve noise immunity, the oscillator is powered by the VDDPLL and VSSPLL power supply pins.

24.4.1 Gain control

A closed loop control system will be utilized whereby the amplifier is modulated to keep the output waveform sinusoidal and to limit the oscillation amplitude. The output peak to peak voltage will be kept above twice the maximum hysteresis level of the input buffer. Electrical specification details are provided in Section A.8, “Oscillator Characteristics”.

24.4.2 Clock Monitor

The clock monitor circuit is based on an internal RC time delay so that it can operate without any MCU clocks. If no OSCCLK edges are detected within this RC time delay, the clock monitor indicates failure which asserts self clock mode or generates a system reset depending on the state of SCME bit. If the clock monitor is disabled or the presence of clocks is detected no failure is indicated.The clock monitor function is enabled/disabled by the CME control bit, described in Section Chapter 23, “Clock and Reset Generator (CRG).

24.4.3 Doze Mode Operation

When the system is in Doze Mode, the oscillator operates as normal.

24.5 Initialization/Application Information

If the oscillator is used in External Clock mode, the XTAL pin should be grounded.

24.6 OSC Bus Aborts

There is no bus interface for the oscillator. All oscillator control registers reside in the CRG module.

24.7 OSC Differences from MAC71xx

- No Reduced Oscillator Amplitude (ROA) mode
- EXTAL and XTAL changed from 2.5V to 3.3V signals
- Colpitts mode removed
- Frequency range for loop controlled Pierce mode changed
- Frequency range for external clock mode changed

24.8 OSC Application Usage

24.8.1 OSC Mode Selection

The oscillator mode is chosen by the value of the $\overline{\text{XCLKS}}$ hardware configuration pin at reset, as follows:

Table 24-2. Oscillator Modes

XCLKS	Mode
0	External Clock Mode. In this mode, the EXTAL pin drives the Oscillator reference clock directly.
1	ALC Mode. In this mode, the EXTAL signal is inverted and fed back out to the crystal (via XTAL). The Oscillator reference clock is driven by the oscillator.

NOTE

Unlike the rest of the hardware configuration pins, the $\overline{\text{XCLKS}}$ pin is not latched during every system reset. It is only latched during a Power On Reset or during a reset after a loss of clock has occurred.

24.8.2 OSC Mode - ALC Mode ($\overline{\text{XCLKS}}$ = 1)

In this mode, the Oscillator reference clock is fed by the oscillator. Refer to Figure 24-4.

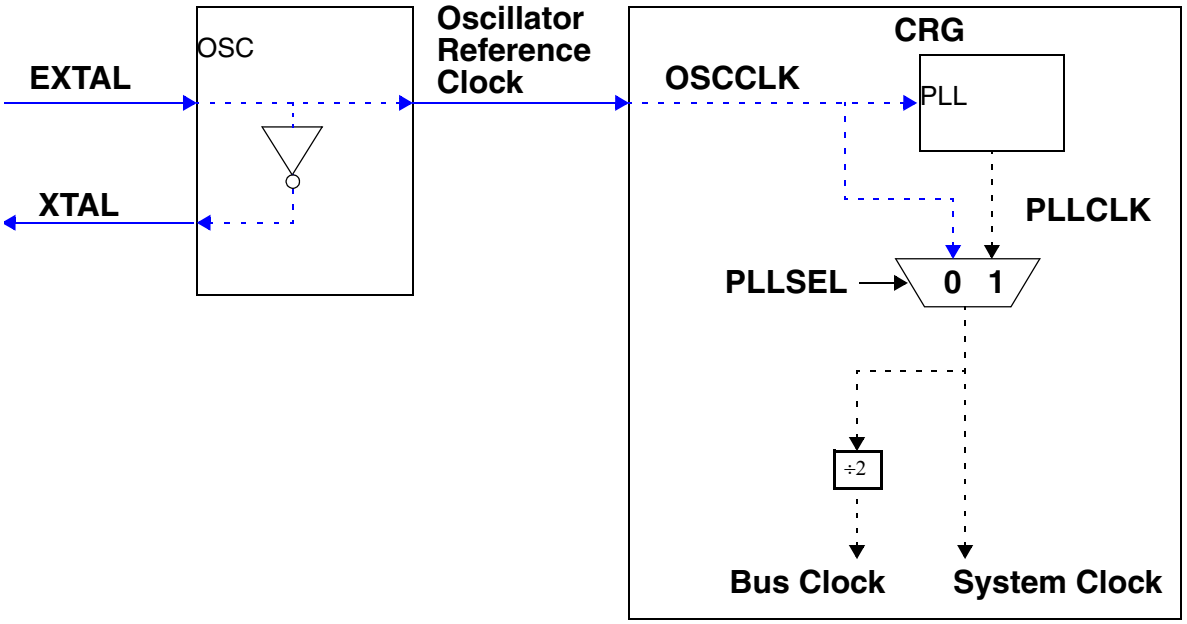


Figure 24-4. ALC Mode Clock Path

24.8.3 OSC Mode - External Clock Mode ($\overline{\text{XCLKS}}$ = 0)

In this mode, the Oscillator reference clock is fed directly from the EXTAL pin. Refer to Figure 24-5.

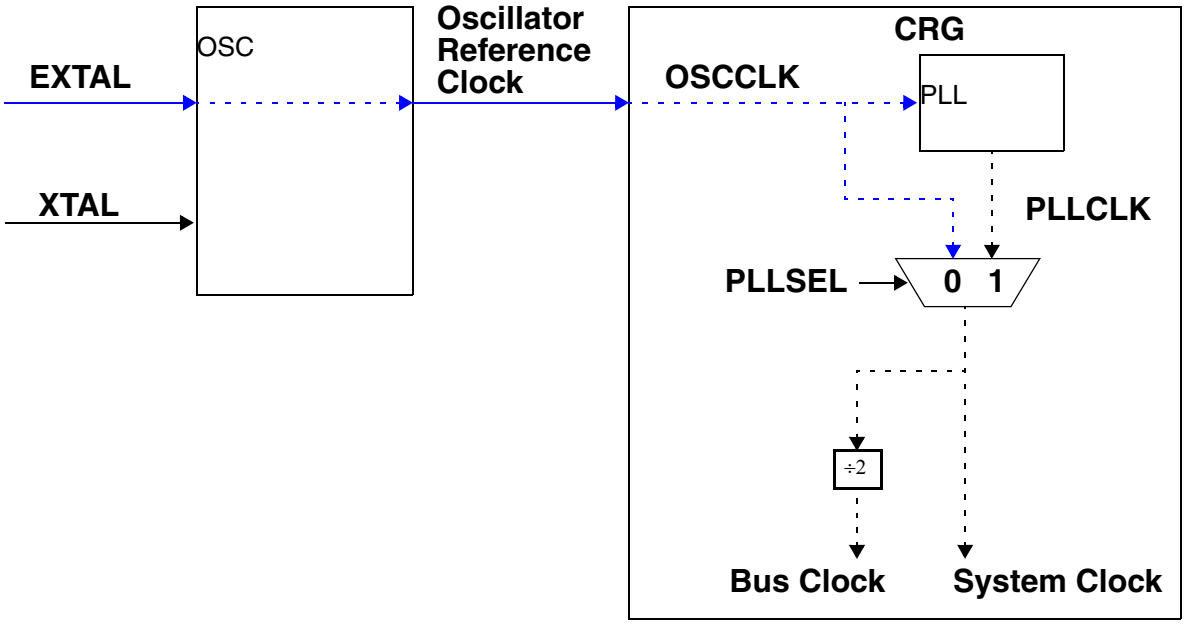


Figure 24-5. External Clock Mode Clock Path

Chapter 25
System Service Module (SSM_MAC7202)

25.1 Introduction

The System Services Module (SSM) provides a global location for system level configuration and status information.

25.1.1 Overview

The System Service Module (SSM), pictured in Figure 25-1, provides system level configuration registers.

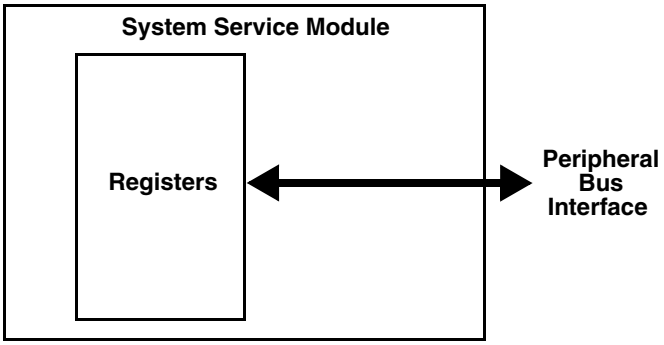


Figure 25-1. System Service Module Block Diagram

25.1.2 Features

The SSM includes these distinctive features:

- System Configuration and Status
  - Memory sizes/status
  - Device Mode and Security Status
  - DMA Status
- Software reset
- Debug Status Port enable and selection
- Bus abort masking

25.1.3 Modes of Operation

The SSM operates identically in all system modes.

25.2 External Signal Description

There are no SSM signals that drive or are driven from MCU pins.

25.3 Memory Map/Register Definition

This section provides a detailed description of all memory-mapped registers in the SSM.

Table 25-1 shows the memory map for the SSM. Note that all addresses are offsets; the absolute address may be calculated by adding the specified offset to the base address of the SSM.

Table 25-1. Module Memory Map

Address	Register	Size	Access	Mode[1]
Base + 0x0000	Reserved	16 bits	Reads/Writes have no effect.	A
Base + 0x0002	System Status (STATUS)	16 bits	R	A
Base + 0x0004	System Memory Configuration (MEMCONFIG)	16 bits	R	A
Base + 0x0006	Debug Status Port (DEBUGPORT)	16 bits	R/W	A
Base + 0x0008	Error Configuration (ERROR)	16 bits	R/W	A
Base + 0x000A	Reserved	16 bits	Reads/Writes have no effect.	A
Base + 0x000C	System Reset (SYSRESET)	16 bits	R/W	A
Base + 0x000E	Reserved	16 bits	Reads/Writes have no effect.	A
Base + 0x0010 to Base + 0x3FFF	Reserved	See Note [2]		

1. **U** = User Mode, **S** = Supervisor Mode, **A** = All (No restrictions)
2. If enabled at the SoC level, accessing these register addresses will cause bus aborts.

All registers are accessible via 8-bit, 16-bit or 32-bit accesses. However, 16-bit accesses must be aligned to 16-bit boundaries, and 32-bit accesses must be aligned to 32-bit boundaries. As an example, the STATUS register is accessible by a 16-bit READ/WRITE to address ’Base + 0x0002’, but performing a 16-bit access to ’Base + 0x0003’ is illegal.

25.3.1 Register Descriptions

The following memory-mapped registers are available in the SSM. Those bits that are shaded out are read-only, meaning that writes to that bit will have no effect. Those bits that are shaded out and marked **rsvd** are reserved for future use. To optimize future compatibility, these bits should be masked out during any read/write operations to avoid conflict with future implementations.

25.3.1.1 System Status Register

The System Status register is a read-only register that reflects the current state of the system.

Address Refer to Table 25-1 Access: Refer to Table 25-2

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	NEXUS		0	0	0	SEC	MODE		DMAIDLE	DMAACTCH			
W	rsvd	rsvd	rsvd													
RESET:	0	0	0	0	0	0	0	0	0/1	0/1	0/1	0	0	0	0	0

= Writes have no effect on this bit rsvd = Reserved for future use

Figure 25-2. Status (STATUS) Register

Table 25-2. STATUS Allowed Register Accesses

	8-bit	16-bit	32-bit[1]
READ	Allowed	Allowed	Not Allowed
WRITE	Not Allowed	Not Allowed	Not Allowed

1. All 32-bit accesses must be aligned to 32-bit addresses (i.e. 0x0, 0x4, 0x8 or 0xC).

Table 25-3. STATUS Field Descriptions

Field	Description
12–11 NEXUS	Nexus Status. This field reflects the current status of the Nexus Port. 00 No Nexus hardware attached 01 No Nexus hardware attached 10 Nexus hardware attached to Nexus Primary Port 11 Nexus hardware attached to Nexus Secondary Port
7 SEC	Security Status. This bit reflects the current security state of the Flash. 1 The Flash is secured 0 The Flash is not secured
6–5 MODE	Device Mode. This field reflects the current mode of the device, and is only valid when the TEST pin is driven low. 00 Single Chip 01 Reserved 10 Primary Bootloader 11 Single Chip

Table 25-3. STATUS Field Descriptions (Continued)

Field	Description
4 DMAIDLE	DMA Is Idle. This bit reflects the current status of the DMA. It is primarily used for debug purposes. 1 The DMA is idle 0 The DMA is performing a bus read or write
3–0 DMAACTCH	Active DMA Channel. This field identifies which DMA channel, if any, is currently active (i.e. executing bus transactions). Note that this value is only valid if DMAIDLE = 0. These bits are primarily used for debug purposes. 0x0 Channel 0 active 0x1 Channel 1 active 0x2 Channel 2 active 0x3 Channel 3 active 0x4 Channel 4 active 0x5 Channel 5 active 0x6 Channel 6 active 0x7 Channel 7 active 0x8 Channel 8 active 0x9 Channel 9 active 0xA Channel 10 active 0xB Channel 11 active 0xC Channel 12 active 0xD Channel 13 active 0xE Channel 14 active 0xF Channel 15 active

25.3.1.2 System Memory Configuration Register

The System Memory Configuration register is a read-only register that reflects the memory configuration of the system.

Address Refer to Table 25-1 Access: Refer to Table 25-5

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	SRAMSIZE					IFLASHSIZE					IFLASHVALID	DFLASHSIZE				DFLASHVALID
W																
RESET:	1	0	1	1	1	1	0	1	0	1	1	0	1	0	0	1

= Writes have no effect on this bit rsvd = Reserved for future use

Figure 25-3. System Memory Configuration (MEMCONFIG) Register

Table 25-4. MEMCONFIG Field Descriptions

Field	Description
15–11 SRAM SIZE	SRAM Size. This field identifies the size of the on-chip SRAM memory. 0x00–0x03 No SRAM 0x04 4 Kb 0x05 8 Kb 0x06 16 Kb 0x07 32 Kb 0x10–0x13 No SRAM 0x14 2 Kb 0x15 6 Kb 0x16 12 Kb 0x17 20 Kb
10–8 IFLASH SIZE	Instruction Flash Size. This field identifies the size of the on-chip Instruction (Program) Flash memory. 0x00–0x0F No Instruction Flash 0x10 64 Kb 0x11 96 Kb 0x12 128 Kb 0x13 192 Kb 0x14 256 Kb 0x15 384 Kb 0x16 512 Kb 0x17 768 Kb 0x18 1024 Kb
5 IFLASH VALID	Instruction Flash Valid. This bit identifies whether or not the on-chip Instruction Flash is visible in the system memory map. 1 Instruction Flash is visible 0 Instruction Flash is not visible
4–1 DFLASH SIZE	Data Flash Size. This field identifies the size of the on-chip Data Flash memory. 0x0–0x3 No Data Flash 0x4 4 Kb 0x5 8 Kb 0x6 16 Kb 0x7 32 Kb 0x9–0xF No Data Flash
0 DFLASH VALID	Data Flash Valid. This bit identifies whether or not the on-chip Data Flash is visible in the system memory map 1 Data Flash is visible 0 Data Flash is not visible

Table 25-5. MEMCONFIG Allowed Register Accesses

	8-bit	16-bit	32-bit
READ	Allowed	Allowed	Allowed (slso reads DEBUGPORT)
WRITE	Not Allowed	Not Allowed	Not Allowed

25.3.1.3 Debug Status Port Register

The Debug Status Port register is used to (optionally) select Port F as a debug status port.

Address Refer to Table 25-1 Access: Refer to Table 25-8

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	DEBUG_MODE		
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Reserved for future use

Figure 25-4. Debug Status Port (DEBUGPORT) Register

Table 25-6. DEBUGPORT Field Descriptions

Field	Description
2–0 DEBUG_ MODE	Debug Status Port Mode. This field selects the alternate debug functionality for Port F 000 No alternate functionality selected 001 Mode 1 Selected 010 Mode 2 Selected 011 Mode 3 Selected 100 Mode 4 Selected 101 Mode 5 Selected 110 Mode 6 Selected 111 Mode 7 Selected Table 25-7 describes the functionality of the Debug Status Port in each mode.

Table 25-7. Debug Status Port Modes

Pin[1]	Mode 1	Mode 2	Mode 3	Mode 4	Mode 5	Mode 6	Mode 7
PF0	JTAG Lockout Recovery started	Reserved	STATUS[0]	STATUS[8]	MEMCONFIG[0]	MEMCONFIG[8]	Reserved
PF1	JTAG Lockout Recovery with PTIMER load started	Reserved	STATUS[1]	STATUS[9]	MEMCONFIG[1]	MEMCONFIG[9]	Reserved
PF2	JTAG Lockout Recovery running	Reserved	STATUS[2]	STATUS[10]	MEMCONFIG[2]	MEMCONFIG[10]	Unused (low)
PF3	System has entered DOZE mode	Reserved	STATUS[3]	STATUS[11]	MEMCONFIG[3]	MEMCONFIG[11]	Unused (low)
PF4	System has entered DEBUG mode	Reserved	STATUS[4]	STATUS[12]	MEMCONFIG[4]	MEMCONFIG[12]	Reserved
PF9	Core is held (not running)	Reserved	STATUS[5]	STATUS[13]	MEMCONFIG[5]	MEMCONFIG[13]	Reserved
PF6	Unused (low)	Unused (low)	STATUS[6]	STATUS[14]	MEMCONFIG[6]	MEMCONFIG[14]	Unused (low)
PF7	Unused (low)	Unused (low)	STATUS[7]	STATUS[15]	MEMCONFIG[7]	MEMCONFIG[15]	Unused (low)

1. All signals are active high, unless otherwise noted.

Table 25-8. DEBUGPORT Allowed Register Accesses

	8-bit	16-bit	32-bit[1]
READ	Allowed	Allowed	Not Allowed
WRITE	Allowed	Allowed	Not Allowed

1. All 32-bit accesses must be aligned to 32-bit addresses (i.e. 0x0, 0x4, 0x8 or 0xC).

25.3.1.4 Error Configuration

The Error Configuration register is a read-write register that controls the error handling of the system.

Address Refer to Table 25-1

Access: Refer to Table 25-10

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	PER_ABORT	REG_ABORT
W	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd		
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Writes have no effect on this bit

rsvd = Reserved for future use

Figure 25-5. Error Configuration (ERROR) Register

Table 25-9. ERROR Field Descriptions

Field	Description
1 PER_ ABORT	Peripheral Bus Abort Enable. This bit enables bus aborts on any access to a peripheral slot that is not used on the device. This feature is intended to aid in debugging when developing application code. 1 Illegal accesses to non-existing peripherals produce a Prefetch or Data Abort exception 0 Illegal accesses to non-existing peripherals do not produce a Prefetch or Data Abort exception; This includes the following address spaces: 0xFC09 C000 – 0xFC0A BFFF 0xFC0C 0000 – 0xFC0C 3FFF 0xFC0C C000 – 0xFC0D BFFF 0xFC0E 4000 – 0xFC0E 7FFF 0xFC0E C000 – 0xFC0F 0000 0xFC0F 4000 – 0xFFFF FFFF **Note:** Accesses to the following address spaces will produce a Prefetch or Data Abort exception, regardless of the value of the PER_ABORT bit: 0xFC00 C000 – 0xFC03 FFFF 0xFC04 C000 – 0xFC07 FFFF
0 REG_ ABORT	Register Bus Abort Enable. This bit enables bus aborts on illegal accesses to off-platform peripherals. Illegal accesses are defined as reads or writes to reserved addresses within the address space for a particular peripheral. This feature is intended to aid in debugging when developing application code. 1 Illegal accesses to peripherals produce a Prefetch or Data Abort exception 0 Illegal accesses to peripherals do not produce a Prefetch or Data Abort exception This includes the following peripherals: • SSM • DMA Channel Mux • CRG • PIT_RTI • VREG_3V3 • FLEXCAN_A/FLEXCAN_B • IIC • DSPI_A/DSPI_B • SCI_A/SCI_B • eMIOS • ATD • CFM (registers) • PIM All other modules (DMA, MCM, AEIM, etc.) are not affected by this register, and will produce Prefetch or Data Abort exceptions for illegal accesses, regardless of the value of the REG_ABORT bit. **Note:** Transfers to Peripheral Bus resources may be aborted even before they reach the Peripheral Bus (i.e. at the AIPS level). In this case, the PER_ABORT and REG_ABORT register bits will have no effect on the abort.

Table 25-10. ERROR Allowed Register Accesses

	8-bit	16-bit	32-bit
READ	Allowed	Allowed	Allowed
WRITE	Allowed	Allowed	Allowed

25.3.1.5 System Reset Register

The System Reset register provides a hardware interlocked method of resetting the system via software.

Address Refer to Table 25-1

Access: Refer to Table 25-12

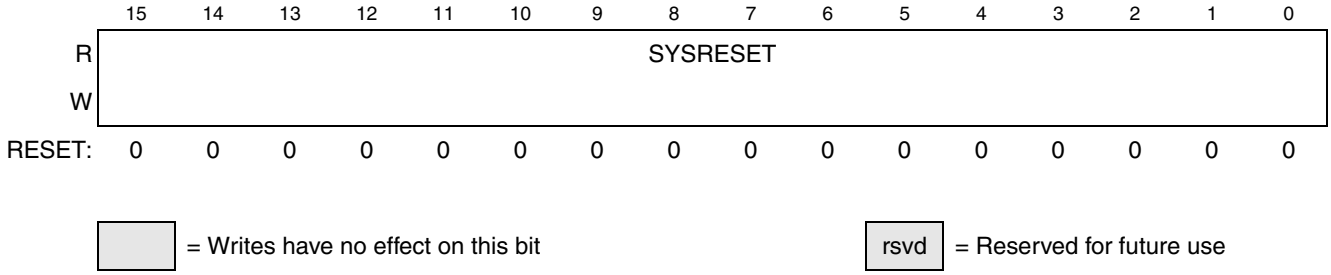


Figure 25-6. System Reset (SYSRESET) Register

Table 25-11. SYSRESET Field Descriptions

Field	Description
15–0 SYS RESET	Execute a system reset. This register enables software to reset the system. In order to prevent spurious resets, a hardware interlock has been implemented. In order to reset the system, the following sequence of writes must be executed: • 16-bit Write of 0xb6b6 • 16-bit Write of 0x3535 Once the final write to the SYSRESET register is performed, the system will reset itself. If any value other than 0x3535 is written to the register after writing 0xb6b6, or an 8-bit write of the register is performed, the sequence must be started again from the beginning. Once the final write is performed, the system will reset itself, with all further writes to the register being ignored until the reset sequence is complete.

Table 25-12. SYSRESET Allowed Register Accesses

	8-bit	16-bit	32-bit
READ	Allowed	Allowed	Allowed
WRITE	Not Allowed	Allowed	Not Allowed

25.4 Functional Description

This section provides a complete functional description of the System Services Module. The primary purpose of the SSM is to provide information about the current state and configuration of the system that may be useful for configuring application software and for debug of the system.

25.4.1 System Configuration/Status

The MAC7200 provides three registers in the SSM to aid in configuring software and debugging your system:

- STATUS
- MEMCONFIG

The STATUS register is a 16-bit read-only register that reflects the current status of the DMA, and is primarily used for debug. By reading the STATUS register, you can determine which DMA channel, if any, is currently active. This information should not be used as part of the execution of application code, but

may be useful for debug purposes. Bits [15:5] of the STATUS register are reserved for future use, and a read from these bits may return any value. Therefore, it is necessary to mask out these bits when reading from this register (refer to Section 25.5.3, “Using the STATUS register).

The MEMCONFIG register is a 16-bit read-only register that reflects the configuration of the various memories on the SoC. The PAC7202 device, for example, has the following memories:

- 256 Kb Instruction Flash
- 8 Kb Data Flash
- 16 Kb SRAM

With the above memories, we would have the following MEMCONFIG register valid (assuming all memories are visible in the memory map):

- **M**EMCONFIG[4:1] = DFLASHSIZE = 0x4 (4 Kb)
- MEMCONFIG[9:6] = IFLASHSIZE = 0xB (256 Kb)
- MEMCONFIG[14:11] = SRAMSIZE = 0x6 (16 Kb)

25.5 Initialization/Application Information

25.5.1 Enabling the SSM

It is not necessary to enable the SSM before it can be used.

25.5.2 Reset

The reset state of each individual bit is shown within the Register Description section (see Section 25.3.1, “Register Descriptions).

25.5.3 Using the STATUS register

Bits [15:5] of the STATUS register are reserved for future use; therefore you should mask these bits out when reading this register, as shown in the following code example:

```
In File registers.h:
        #define SSM_BASE_ADDRESS     0xFC008000/* Example only ! */
        /* Following example assumes short is 16-bits */
        volatile unsigned short *STATUS = (volatile unsigned short *)
(SSM_BASE_ADDRESS+0x0002);

In File main.c:
        #include "registers.h"
                :
                :
        unsigned char dma_active;
        unsigned char dma_channel;
        /* Do single read of STATUS register to "capture" state */
        dma_channel = (unsigned char) *STATUS & 0x001f;
        dma_active  = dma_channel >> 4;
        dma_channel = dma_channel & 0x0f;
```

Because the DMA is executing independently of the processor core, using the STATUS register in software is discouraged, as the latency for the execution of the above code may be longer than the DMA is actually active. Use of this register is intended for debug purposes only.

25.5.4 Using the MEMCONFIG register

Bit 15 of the MEMCONFIG register is reserved for future use; therefore you should mask this bit out when reading this register, as shown in the following code example:

```
In File registers.h:
        #define SSM_BASE_ADDRESS     0xFC008000/* Example only ! */
        /* Following example assumes short is 16-bits */
        volatile unsigned short *MEMCONFIG = (volatile unsigned short *)
(SSM_BASE_ADDRESS+0x0004);

In File main.c:
        #include "registers.h"
                :
                :
        unsigned short memconfig;
        unsigned char sram_valid;
        unsigned char sram_size;
        unsigned char iflash_valid;
        unsigned char iflash_size;
        unsigned char dflash_valid;
        unsigned char dflash_size;
        memconfig = *MEMCONFIG & 0x7fff;
        sram_valid = (unsigned char) (memconfig & 0400) >> 10;
        iflash_valid = (unsigned char) (memconfig & 0020) >> 5;
        dflash_valid = (unsigned char) (memconfig & 0001);
        sram_size = (unsigned char) (memconfig & 7800) >> 11;
        iflash_size = (unsigned char) (memconfig & 02C0) >> 6;
        dflash_size = (unsigned char) (memconfig & 001e) >> 1;
```

25.6 SSM Bus Aborts

The SSM module supports Peripheral Bus bus aborts, and enforces the following memory map:

Table 25-13. SSM Bus Aborts

Abort	Allowed	
	$0000-$000f	
$0010-$3fff		

Supervisor Access: Unused.

25.7 SSM Differences from MAC71xx

- Added PER_ABORT field to the ERROR register
- Added Nexus information to STATUS register
- Changed SRAMSIZE field from 4 bits to 5 bits and re-encoded

- Changed IFLASHSIZE encoding
- Added Debug Status Port and associated DEBUGPORT register
- Added Software Reset and associated SYSRESET register
- Changed DFV Mode register access and functionality (non-customer feature)

Chapter 26
Periodic Interrupt Timer (PIT_RTI)

26.1 Introduction

This section describes the function of the Periodic Interrupt Timer block (PIT_RTI) on the MAC7200. The PIT is an array of timers that can be used to raise interrupts and trigger DMA channels. It also provides a dedicated Real Time Interrupt Timer (RTI), which runs on a separate clock and can be used for system wakeup.

26.1.1 Overview

The Periodic Interrupt Timer provides ten programmable 32-bit timers offering a range of functions plus one programmable 24-bit Real Timer Interrupt (RTI). Once reaching zero, a trigger can be generated and the counter is reloaded with the start value and continues to be decremented. If required, the value of each of the timers down count can be read by software at any time.

The PIT has one Real Time Interrupt (RTI), four general purpose timers, four dedicated timers used to trigger DMA transfers and two timers used to trigger ATD conversions.

PIT 0 is used to provide the RTI and can generate an interrupt which may also be inhibited. The four general purpose timers, PIT 1 to PIT4 can be used to generate interrupts or to trigger DMA channels 4 to 7. PIT 5 to PIT 8 are used to trigger DMA channels 0 to 3. PIT 9 and 10 are used for the ATD triggers SYSTRIG0 and SYSTRIG1 respectively. Note that PIT4 and the RTI share the same interrupt vector. Therefore, to use just the RTI as an interrupt, you must disable the interrupt for PIT4. Conversely, to use just PIT4 as an interrupt source, you must disable the interrupt for the RTI.

All of the PITs take their clock from the system clock frequency FSYS with the exception of the RTI which uses the oscillator clock as its source. This allows the RTI to continue operating while in low power modes.

The PIT module can be independently disabled by writing to the MDIS bit in the modules PITCTRL register. Disabling the module will turn off all clocks to the module with the exception of the RTI clock. This will disable PIT 1 to 10, but will allow the RTI to continue operating and for most of the module registers to remain accessible by the core across the peripheral bus. The MDIS bit is intended to be used when the module is not required in the application. By default the PIT is disabled after reset (MDIS=1), so, prior to use, the MDIS bit must be cleared. Alternatively, each timer may be independently disabled.

26.1.2 Block Diagram

Figure 26-1 shows the PIT block diagram.

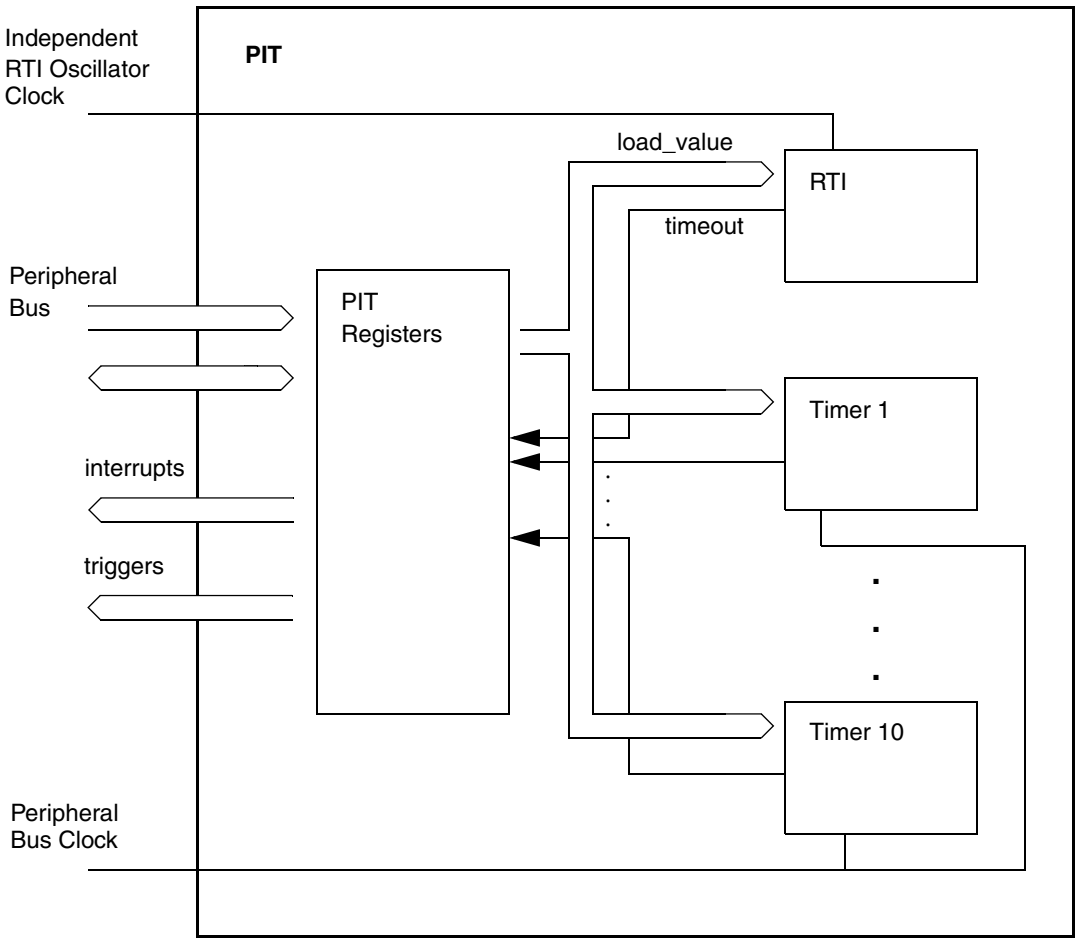


Figure 26-1. Block diagram of PIT

26.2 PIT Features

- Memory Map: 32-bit peripheral, byte/halfword/word addressable
- Independent timeout period for each timer.
- Four 32-bit general purpose PIT timers, configurable to generate DMA trigger pulses.
- Four 32-bit PITs to generate DMA trigger pulse.
- Two 32-bit timers that can be configured to generate ATD trigger pulses.
- One 24-bit real-time interrupt (RTI) timer.
- RTI operates from oscillator output clock.
- Can be completely turned off
- Timeout sets register flag and optional interrupt
- Selectable divider (from Oscillator Clock) from 2^{10} to $16x2^{16}$

The PIT is used primarily for 4 things:

- Timer 0 will serve as a Real-Time Interrupt (RTI) that can be used to wakeup the system out of low power modes. For this reason, it is run off of a separate clock.
- Timers 1-4 may be used to generate general purpose periodic interrupts
- Timers 1-8 may be used as DMA triggers
- Timers 9-10 may be used as on-chip sample triggers for the ATD

Table 26-1. PIT Timer Usage

Timer	Interrupt	DMA trigger (DMA channel)	ATD Trigger
0 (RTI)	X		
1	X	X (#0)	
2	X	X (#1)	
3	X	X (#2)	
4	X	X (#3)	
5		X (#4)	
6		X (#5)	
7		X (#6)	
8		X (#7)	
9			X (SYSTRIG1)
10			X (SYSTRIG0)

26.2.1 Modes of Operation

This subsection describes briefly all operating modes supported by the PIT.

- Run Mode

  All functional parts of the PIT are running during normal Run Mode.
- Doze Mode

  The PIT's RTI timer can run in Doze mode depending on the settings in the CRG (Clock and Reset Generator Module).

26.3 PIT Implementation

The PIT on the MAC72xx is an enhanced version of the PIT on the MAC71xx family of devices.

26.4 PIT External Pins

There are no PIT signals that drive or are driven from MCU pins.

26.5 PIT Bus Aborts

The PIT module supports Peripheral Bus bus aborts, and enforces the following memory map:

Table 26-2. PIT Bus Aborts

Abort	Allowed
	$0000-$01ff
$0200-$3fff	

Supervisor Access: Unused

26.6 PIT Differences from MAC71xx

- Changed Timer 1-10 from 24-bit timers to 32-bit timers. Note that the width of the RTI was not changed.
  - Changed TLVAL1-TLVAL10 from 24-bit to 32-bit registers
- Fixed MUCts01646 (Write on bus abort still writes the register)

26.7 PIT Application Usage

26.7.1 Enabling the PIT

Before the PIT can be used, it must be explicitly enabled by clearing the **MDIS** bit.

26.7.2 SYSTRIG Order

Note that the order of the SYSTRIG channels (on-chip ATD triggers) is reversed with respect to the order of the timers themselves. In order words, the first ATD trigger (SYSTRIG0) corresponds to Timer 10, not Timer 9 as would be expected. This is done solely to keep backwards compatibility with the MAC71xx family of devices.

26.7.3 Interrupts

For flexibility, the RTI interrupt may be used by either the RTI or Timer #4, as follows:

Table 26-3. PIT Interrupt Sources

TIE4	RTIE	Interrupt Source
0	0	Disabled
0	1	RTI only
1	0	Timer #4 only
1	1	RIT *or* Timer #4 (Either source will trigger an interrupt)

26.8 Memory Map and Register Description

This section provides a detailed description of all registers accessible in the PIT_RTI module.

26.8.1 Memory Map

Figure 26-4 gives an overview on all PIT_RTI registers. The first timer (timer 0) is the RTI timer.

Table 26-4. PIT_RTI Memory Map

Address Offset	Use	Access
0x00	PIT RTI Load Value Register	R/W
0x04	PIT Timer Load Value Register 1	R/W
0x08	PIT Timer Load Value Register 2	R/W
0x0C	PIT Timer Load Value Register 3	R/W
0x10	PIT Timer Load Value Register 4	R/W
0x14	PIT Timer Load Value Register 5	R/W
0x18	PIT Timer Load Value Register 6	R/W
0x1C	PIT Timer Load Value Register 7	R/W
0x20	PIT Timer Load Value Register 8	R/W
0x24	PIT Timer Load Value Register 9	R/W
0x28	PIT Timer Load Value Register 10	R/W
0x32–0x7F	Reserved	
0x80	PIT Current RTI Value	R
0x84	PIT Current Timer Value 1	R
0x88	PIT Current Timer Value 2	R
0x8C	PIT Current Timer Value 3	R
0x90	PIT Current Timer Value 4	R
0x94	PIT Current Timer Value 5	R
0x98	PIT Current Timer Value 6	R
0x9C	PIT Current Timer Value 7	R
0xA0	PIT Current Timer Value 8	R
0xA4	PIT Current Timer Value 9	R
0xA8	PIT Current Timer Value 10	R
0xAC–0x9F	Reserved	
0x100	PIT Interrupt Flags Register	R/W
0x104	PIT Interrupt Enable Register	R/W
0x108	PIT Interrupt/DMA Select Register	R/W
0x10C	PIT Timer Enable Register	R/W
0x110	PIT Control Register	R/W
0x114–0x1FC	Reserved	

NOTE

Register Address = Base Address + Address Offset, where the Base Address is defined at the MCU level and the Address Offset is defined at the module level.

NOTE

Reserved registers will read as 0, writes will have no effect.

26.8.2 Register Descriptions

This section describes in address order all the PIT_RTI registers and their individual bits.

NOTE

The RTI registers should be set only when the RTI clock is running - e.g. not if the system is in doze mode and the RTI clock is switched off in doze mode.

26.8.2.1 PIT RTI / Timer Load Value Register (TLVAL)

These registers select the timeout period for the timer interrupts. In the case of the RTI, it will take several cycles until this value is synchronized into the RTI clock domain. For all other timers the value change is visible immediately.

Address Offset: 0x00 Access: User read/write

	31	30	29	28	27	26	25	24
R	0	0	0	0	0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0

	23	22	21	20	19	18	17	16
R	TSV23	TSV22	TSV21	TSV20	TSV19	TSV18	TSV17	TSV16
W								
Reset	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8
R	TSV15	TSV14	TSV13	TSV12	TSV11	TSV10	TSV9	TSV8
W								
Reset	0	0	0	0	0	0	0	0

	7	6	5	4	3	2	1	0
R	TSV7	TSV6	TSV5	TSV4	TSV3	TSV2	TSV1	TSV0
W								
Reset	0	0	0	0	0	0	0	0

Figure 26-2. PIT RTI Load Value Register (TLVAL0)

Table 26-5. TLVAL0 Field Descriptions

Field	Description
31–24	Reserved, should be cleared.
23–0 TSV*n*	Time Start Value Bits. These bits set the timer start value. The timer will count down until it reaches 0, then it will generate an interrupt and load this register value again. Writing a new value to this register will not restart the timer, instead the value will be loaded once the timer expires. To abort the current cycle and start a timer period with the new value, the timer must be disabled and enabled again (see Figure 26-9). NOTE: For the RTI, the timer should not be set to a value lower than 32 cycles, otherwise interrupts may be lost, as it takes several cycles to clear the RTI interrupt. For the other timers, this limit does not apply, however there will be practical limits, since the processor will require several cycles to service an interrupt.

Address Offset: 0x04–0x2B Access: User read/write

	31	30	29	28	27	26	25	24
R W	TSV31	TSV30	TSV29	TSV28	TSV27	TSV26	TSV25	TSV24
Reset	0	0	0	0	0	0	0	0

	23	22	21	20	19	18	17	16
R W	TSV23	TSV22	TSV21	TSV20	TSV19	TSV18	TSV17	TSV16
Reset	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8
R W	TSV15	TSV14	TSV13	TSV12	TSV11	TSV10	TSV9	TSV8
Reset	0	0	0	0	0	0	0	0

	7	6	5	4	3	2	1	0
R W	TSV7	TSV6	TSV5	TSV4	TSV3	TSV2	TSV1	TSV0
Reset	0	0	0	0	0	0	0	0

Figure 26-3. PIT Timer Load Value Registers (TLVAL1–10)

Table 26-6. TLVAL1–10 Field Descriptions

Field	Description
31–0 TSV*n*	Time Start Value Bits. These bits set the timer start value. The timer will count down until it reaches 0, then it will generate an interrupt and load this register value again. Writing a new value to this register will not restart the timer, instead the value will be loaded once the timer expires. To abort the current cycle and start a timer period with the new value, the timer must be disabled and enabled again (see Figure 26-9). NOTE: For the RTI, the timer should not be set to a value lower than 32 cycles, otherwise interrupts may be lost, as it takes several cycles to clear the RTI interrupt. For the other timers, this limit does not apply, however there will be practical limits, since the processor will require several cycles to service an interrupt.

26.8.2.2 PIT Current RTI / Timer Values (TVAL0–10)

These registers indicate the current timer position. In the case of the RTI, this will show a value which is several cycles old, since it originates from a potentially different clock domain.

Address: Offset: 0x80 Access: User read

	31	30	29	28	27	26	25	24
R	0	0	0	0	0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0

	23	22	21	20	19	18	17	16
R	TVL23	TVL22	TVL21	TVL20	TVL19	TVL18	TVL17	TVL16
W								
Reset	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8
R	TVL15	TVL14	TVL13	TVL12	TVL11	TVL10	TVL9	TVL8
W								
Reset	0	0	0	0	0	0	0	0

	7	6	5	4	3	2	1	0
R	TVL7	TVL6	TVL5	TVL4	TVL3	TVL2	TVL1	TVL0
W								
Reset	0	0	0	0	0	0	0	0

Figure 26-4. PIT Current RTI Value (TVAL0)

Table 26-7. TVAL0 Field Descriptions

Field	Description
31–24	Reserved, should be cleared.
23–0 TVL*n*	Current Timer Value. These bits represent the current timer value. Note that the timer uses a downcounter. NOTE: The timer values will be frozen in Debug mode

Address Offset: 0x84–0xA

Access: User read

	31	30	29	28	27	26	25	24
R	TVL31	TVL30	TVL29	TVL28	TVL27	TVL26	TVL25	TVL24
W								
Reset	0	0	0	0	0	0	0	0

	23	22	21	20	19	18	17	16
R	TVL23	TVL22	TVL21	TVL20	TVL19	TVL18	TVL17	TVL16
W								
Reset	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8
R	TVL15	TVL14	TVL13	TVL12	TVL11	TVL10	TVL9	TVL8
W								
Reset	0	0	0	0	0	0	0	0

	7	6	5	4	3	2	1	0
R	TVL7	TVL6	TVL5	TVL4	TVL3	TVL2	TVL1	TVL0
W								
Reset	0	0	0	0	0	0	0	0

Figure 26-5. PIT CurrentTimer Values (TVAL1–10)

Table 26-8. TVAL1–10 Field Descriptions

Field	Description
31–0 TVL*n*	Current Timer Value. These bits represent the current timer value. Note that the timer uses a downcounter. NOTE: The timer values will be frozen in Debug mode.

26.8.2.3 Interrupt Flags Register (PITFLG)

These registers hold the PIT interrupt flags. Timer 0 is the special timer RTI, which can be used to wake-up the device.

Address Offset: **0x**100 Access: User read/write (write to clear)

	31	30	29	28	27	26	25	24
R	0	0	0	0	0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0

	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8
R	0	0	0	0	0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0

	7	6	5	4	3	2	1	0
R	0	0	0	TIF4	TIF3	TIF2	TIF1	RTIF
W								
Reset	0	0	0	0	0	0	0	0

Figure 26-6. APIT Interrupt Flags Register (PITFLG)

Table 26-9. PITFLG Field Desciptions

Field	Description
31–5	Reserved, should be cleared.
4–1 TIF*n*	Real Time Interrupt Flags for Timer 1–4. TIF*n* is set to 1 at the end of the timer period. This flag can be cleared only by writing a 1. Writing a 0 has no effect. If enabled (TIEx = 1 and ISELx = 1), TIF*n* causes an interrupt request. 0 Time-out has not yet occurred 1 Time-out has occurred
0 RTIF	Real Time Interrupt Flag. RTIF is set to 1 at the end of the RTI period. This flag can be cleared only by writing a 1. Writing a 0 has no effect. If enabled (RTIE = 1), RTIF causes an interrupt request. 0 RTI time-out has not yet occurred 1 RTI time-out has occurred

26.8.2.4 PIT Interrupt Enable Register (PITINTEN)

This register enables PIT interrupts.

Address Offset: 0x104 Access: User read/write

	31	30	29	28	27	26	25	24
R	0	0	0	0	0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0

	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8
R	0	0	0	0	0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0

	7	6	5	4	3	2	1	0
R	0	0	0	TIE4	TIE3	TIE2	TIE1	RTIE
W								
Reset	0	0	0	0	0	0	0	0

Figure 26-7. PIT Interrupt Enable Register (PITINTEN)

Table 26-10. PITINTEN Field Descriptions

Field	Description
31–5	Reserved, should be cleared.
4–1 TIE*n*	Timer Interrupt Enable Bit. 0 Interrupt requests from Timer x are disabled 1 Interrupt will be requested whenever TIFx is set When an interrupt is pending (TIF/RTIF set), enabling the interrupt will immediately cause an interrupt event. To avoid this, the associated TIF/RTIF flag must be cleared first.
0 RTIE	Real Time Interrupt Enable Bit. 0 Interrupt requests from RTI are disabled 1 Interrupt will be requested whenever RTIF is set

26.8.2.5 PIT Interrupt/DMA Select Registers (PITINTSEL)

This register decides whether a channel generates an interrupt or is used for DMA triggering.

Address Offset: 0x108 Access: User read/write

	31	30	29	28	27	26	25	24
R	0	0	0	0	0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0

	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8
R	0	0	0	0	0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0

	7	6	5	4	3	2	1	0
R	0	0	0	ISEL4	ISEL3	ISEL2	ISEL1	1
W								
Reset	0	0	0	0	0	0	0	1

Figure 26-8. PIT Interrupt/DMA Select Registers (PITINTSEL)

Table 26-11. PITINTSEL Field Descriptions

Field	Description
31–5	Reserved, should be cleared.
4–1 ISEL*n*	Interrupt Selector. 0 The timer will trigger a DMA channel 1 The timer will generate an interrupt if enabled
0	Reserved, should be cleared.

26.8.2.6 PIT Timer Enable Register (PITEN)

This register enables the PIT timers.

Address Offset: 0x10C Access: User read/write

	31	30	29	28	27	26	25	24
R	0	0	0	0	0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0

	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8
R	0	0	0	0	0	PEN10	PEN9	PEN8
W								
Reset	0	0	0	0	0	0	0	0

	7	6	5	4	3	2	1	0
R	PEN7	PEN6	PEN5	PEN4	PEN3	PEN2	PEN1	PEN0
W								
Reset	0	0	0	0	0	0	0	0

Figure 26-9. PIT Timer Enable Register (PITEN)

Table 26-12. PITEN Field Descriptions

Field	Description
31–11	Reserved, should be cleared.
10–0 PEN*n*	Timer Enable Bit. 0 Timer will be disabled 1 Timer will be active

26.8.2.7 PIT Control Register (PITCTRL)

This register controls whether the clock for the timers 1-10 is enabled. The RTI timer (timer 0) runs on a separate clock which is controlled by the CRG.

Address Offset: 0x110 Access: User read/write

	31	30	29	28	27	26	25	24
R	0	0	0	0	0	0	DOZE	MDIS
W								
Reset	0	0	0	0	0	0	0	1

	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8
R	0	0	0	0	0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0

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

Figure 26-10. PIT Control Registers (PITCTRL)

Table 26-13. PITCTRL Field Descriptions

Field	Description
31–26	Reserved, should be cleared.
25 DOZE	Disable Module in Doze Mode. This is used to disable timers 1–10 in Doze Mode, the RTI (timer 0) is not affected by this bit. 0 Clock for Timers 1–10 stays active in Doze Mode (default) 1 Clock for Timers 1–10 is turned off in Doze Mode
24 MDIS	Module Disable. This is used to disable timers 1–10. The RTI (timer 0) is not affected by this bit. The module should be enabled before any setup is done. 0 Clock for Timers 1–10 is enabled 1 Clock for Timers 1–10 is disabled (default)
23–0	Reserved, should be cleared.

26.9 Functional Description

26.9.1 General

This section gives detailed information on the internal operation of the module. The PIT block has 11 timers: one RTI timer especially to wakeup the processor and 10 additional timers for general-purpose use (e.g. DMA triggering, ATD triggering).

26.9.1.1 Timer / RTI

The timers generate triggers at periodic intervals, when enabled. They load their start values, as specified in their TLVAL registers, then count down until they reach 0. This creates a trigger then they load their respective start value again. Each time a timer reachers 0, it will generate a trigger pulse, and set the interrupt flag.

All interrupts can be enabled or masked (by setting the TIE/RTIE bits in the PITINTEN register and selecting interrupts in the PITINTSEL register). A new interrupt can be generated only after the previous one is cleared. Since in the case of the RTI, clearing the interrupt crosses clock domains, a minimum load value of 32 should be maintained.

If desired, the current counter value of the timer can be read via the TVAL registers. The value of the RTI counter can be delayed considerably, as it is synchronized to the bus clock from the RTI clock domain.

The counter period can be restarted, by first disabling, then enabling the timer with the PITEN registers (see Figure 26-11).

The counter period of a running timer can be modified, by first disabling the timer, setting a new load value and then enabling the timer again (see Figure 26-12).

It is also possible to change the counter period without restarting the timer by writing the TLVAL register with the new load value. This value will then be loaded after the next trigger event (see Figure 26-13).

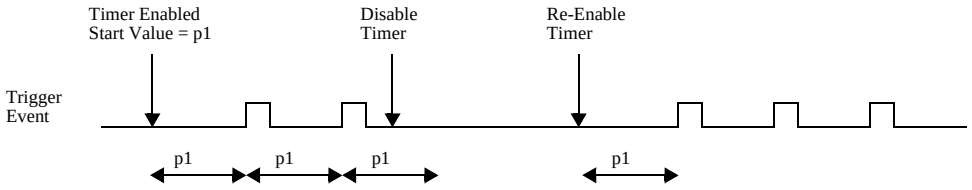


Figure 26-11. Stopping and Starting a Timer

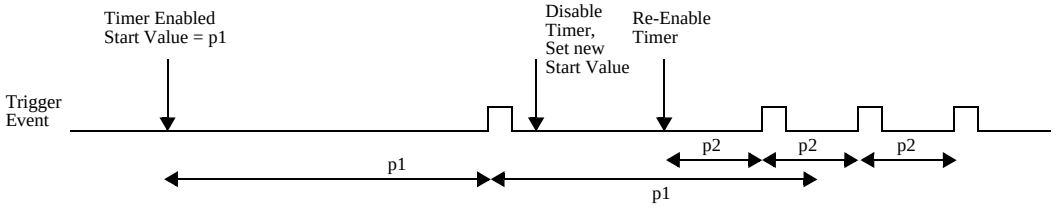


Figure 26-12. Modifying Running Timer Period

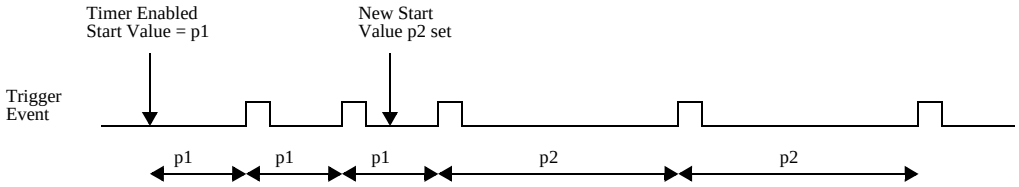


Figure 26-13. Dynamically Setting a New Load Value

Four of the timers and the RTI support interrupts. The RTI is used for system wakeup only. The other interrupts are general-purpose.

26.9.1.2 Debug Mode

In Debug Mode the timers will be frozen - this is intended to aid software development, allowing the developer to halt the processor, investigate the current state of the system (e.g. the timer values) and then continue the operation.

26.9.2 Interrupts

The interrupts generated by the PIT are listed in Figure 26-14. Refer to the MCU specification for related vector addresses and priorities.

Table 26-14. PIT Interrupt Vectors

Interrupt Source	Local Enable
Real time interrupt	RTIE
Timer Interrupts	TIE[4:1]

26.9.2.1 Real Time Interrupt

The PIT_RTI generates a real time interrupt when the selected interrupt time period elapses. The RTI interrupt is disabled locally by setting the RTIE bit to zero. The real time interrupt flag (RTIF) is set to 1 when a timeout occurs, and is cleared to 0 by writing a 1 to the RTIF bit.

Since the RTI operates on the oscillator clock, it can be used for periodic wakeup from WAIT mode providede the CRG's PRE bit is set to 1 and the RTI interrupt is enabled. It can also be used to generate a general purpose interrupt.

26.9.2.2 Timer Interrupts

The PIT can also generate timer interrupts on the first four timer channels, when the timer's selected interrupt period elapses. Timer interrupts are disabled locally by setting the TIE bits to zero. The timer interrupt flags (TIF) are set to 1 when a timeout occurs on the associated timer, and are cleared to 0 by

writing a 1 to that TIF bit. To activate a timer interrupt it must also be switched from trigger mode into interrupt mode, using the PITINTSEL register.

The timer interrupts are general-purpose interrupts.

26.10 Initialization and Application Information

26.10.1 Example Configuration

In the example configuration:

- the PIT clock has a frequency of 50 MHz
- the RTI clock has a frequency of 10 MHz
- the RTI timer shall be set up to create a wakeup interrupt every 500 ms
- timer 1 shall create an interrupt every 5.12 ms
- timer 8 shall create a trigger event every 30 ms.

First the PIT module needs to be activated by writing a 0 to the MDIS bit in the PITCTRL register.

The 50 MHz clock frequency equates to a clock period of 20 ns and the 10 MHz frequency equates to a clock period of 100 ns. Therefore the RTI timer needs to trigger every 500 ms/100 ns = 5000000 cycles, timer 1 needs to trigger every 5.12 ms/20 ns = 256000 cycles and timer 8 every 30 ms/20 ns = 1500000 cycles. The value for the TVAL register trigger would be calculated as (period / clock period) -1.

This means that TVAL0 will be written with 004C4B3F hex, TVAL1 with 00003E7FF hex and TVAL8 with 0016E35F hex.

To generate the interrupt, the interrupt line must be enabled by writing a 1 to the RTIE bit in the PITINTEN register. There is no need to modify PITINTSEL, as the RTI timer is always used for interrupts and never for trigger events. To start the RTI, PEN0 in the PIT Timer Enable register 0 (PITEN0) is set.

The interrupt for Timer 1 is enabled by setting TIE1 in the PITINTEN register and the Interrupt/DMA selector ISEL1 (in PITINTSEL) is set to 1. The timer is started by writing a 1 to bit PEN1 in the PITEN register.

Timer 8 shall be used only for triggering. Only timers 0–4 have interrupt capability. Therefore Timer 8 is started by writing a 1 to bit PEN8 in the PITEN1 register.

It is also possible to setup all timers and start them simultaneously by writing to the PITEN registers. However the RTI still cannot start in synchronisation as it is running on a separate clock.

The following example code matches the described setup:

```
// turn on PIT
PIT_REG_P->pit_CTRL = 0x00;

// RTI
CRG_REG_P->crg_CTL    |= 1<<2;  // Set RTI bit in CLKSEL
PIT_REG_P->pit_TLVAL0 = 0x004C4B3F; // setup RTI for 5000000 cycles
PIT_REG_P->pit_INTEN = 0x00000001; // let RTI generate interrupts
// writing INTSEL is unnecessary
PIT_REG_P->pit_EN |= 1<<0; // start RTI
```

```
// Timer 1
PIT_REG_P->pit_TLVAL1 = 0x00003E7FF; // setup timer 1 for 256000 cycles
PIT_REG_P->pit_INTEN |= 1<<1; // enable Timer 1 interrupts
PIT_REG_P->pit_INTSEL |= 1<<1; // select Timer 1 for interrupts
PIT_REG_P->pit_EN |= 1<<1; // start timer 1

// Timer 8
PIT_REG_P->pit_TLVAL8 = 0x0016E35F; // setup timer 8 for 1500000 cycles
// timer 8 can’t generate interrupts -> no settings needed for trigger
PIT_REG_P->pit_EN |= 1<<8; // start timer 8
```

Chapter 27
Enhanced Direct Memory Access (DMA Channel MUX)

27.1 Introduction

The Enhanced Direct Memory Access (eDMA) controller implemented on the MAC7200 family of devices has 16 channels, but it is possible to perform Direct Memory Accesses between considerably more than 16 sources. The DMA Channel Mux module enables full flexibility in the assignment of DMA channels to sources, with any source being able to be connected to any DMA channel. The DMA Channel Mux additionally allows DMA channels 0 to 7 to be triggered either by the source, or from the Programmable Interrupt Timers (PIT). This allows the DMA to be used to perform periodic transfers to peripherals such as the Port Integration Module or external bus interface.

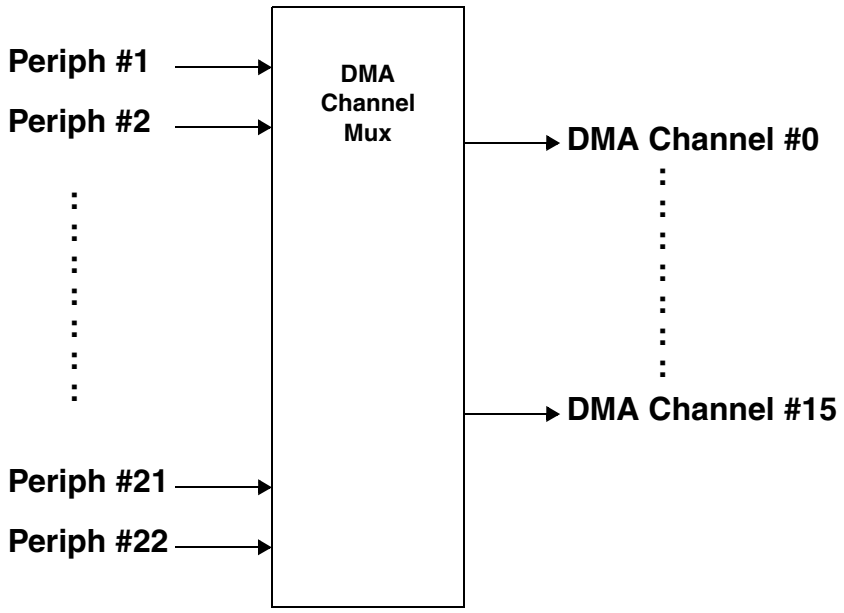


Figure 27-1. DMA Channel Mux

Table 27-1. DMA Request Sources

Peripheral	Number of DMA Channels
SPI_A	2
SPI_B	2
SPI_C	2
IIC	2

Table 27-1. DMA Request Sources (Continued)

Peripheral	Number of DMA Channels
SCI_A	2
SCI_B	2
ATD_A	2
eMIOS	8 channels on MAC72x2 16 channel on MAC72x1 (1 per channel)
PIM (GPIO)	0 (Use “always enabled” functionality)
TOTAL	**22**

27.1.1 Features

The DMA Mux includes these distinctive features:

- Programmable DMA Channel Mux allows assignment of any DMA source to any of the 16 available DMA channels.
- Periodic triggering of up to 8 channels.

27.1.2 Modes of Operation

DMA Channels 0–7 may be used in the following modes, while Channels 8–15 may only be configured to Disabled or Normal Mode.

- Disabled Mode

  In this mode, the DMA channel is disabled. Since disabling and enabling of DMA channels is done primarily via the DMA registers, this mode is used mainly as the reset state for a DMA channel in the DMA Channel Mux. It may also be used to temporarily suspend a DMA Channel while reconfiguration of the system takes place (changing the period of a DMA trigger, for example).
- Normal Mode

  In this mode, a DMA source (such as SCI transmit or SCI receive for example) is routed directly to the specified DMA channel. The operation of the DMA Mux in this mode is completely transparent to the system.
- Periodic Trigger Mode

  In this mode, a DMA source may only request a DMA transfer (such as when a transmit buffer becomes empty or a receive buffer becomes full) periodically. Configuration of the period is done in the registers of the Periodic Interrupt Timer.

27.2 External Signal Description

The DMA Mux has no external pins.

27.3 Memory Map and Register Definition

This section provides a detailed description of all memory-mapped registers in the DMA Mux.

Table 27-2 shows the memory map for the DMA Mux. Note that all addresses are offsets; the absolute address may be computed by adding the specified offset to the base address of the DMA Mux.

Table 27-2. Module Memory Map

Address	Use	Access
Base + $0x00	Channel #0 Configuration (CHCONFIG0)	R/W
Base + $0x01	Channel #1 Configuration (CHCONFIG1)	R/W
Base + $0x02	Channel #2 Configuration (CHCONFIG2)	R/W
Base + $0x03	Channel #3 Configuration (CHCONFIG3)	R/W
Base + $0x04	Channel #4 Configuration (CHCONFIG4)	R/W
Base + $0x05	Channel #5 Configuration (CHCONFIG5)	R/W
Base + $0x06	Channel #6 Configuration (CHCONFIG6)	R/W
Base + $0x07	Channel #7 Configuration (CHCONFIG7)	R/W
Base + $0x08	Channel #8 Configuration (CHCONFIG8)	R/W
Base + $0x09	Channel #9 Configuration (CHCONFIG9)	R/W
Base + $0x0A	Channel #10 Configuration (CHCONFIG10)	R/W
Base + $0x0B	Channel #11 Configuration (CHCONFIG11)	R/W
Base + $0x0C	Channel #12 Configuration (CHCONFIG12)	R/W
Base + $0x0D	Channel #13 Configuration (CHCONFIG13)	R/W
Base + $0x0E	Channel #14 Configuration (CHCONFIG14)	R/W
Base + $0x0F	Channel #15 Configuration (CHCONFIG15)	R/W

All registers are accessible via 8-bit, 16-bit or 32-bit accesses. However, 16-bit accesses must be aligned to 16-bit boundaries, and 32-bit accesses must be aligned to 32-bit boundaries. As an example, CHCONFIG0 through CHCONFIG4 are accessible by a 32-bit READ/WRITE to address ‘Base + 0x00’, but performing a 32-bit access to address ‘Base + 0x01’ is illegal.

27.3.1 Register Descriptions

The following memory-mapped registers are available in the DMA Channel Mux.

27.3.1.1 Channel Configuration Registers

Each of the total of 16 DMA channels can be independently enabled/disabled and associated with 1 of the 30 total DMA sources in the system.

Address: Base + xx – 1 Access: User read/write

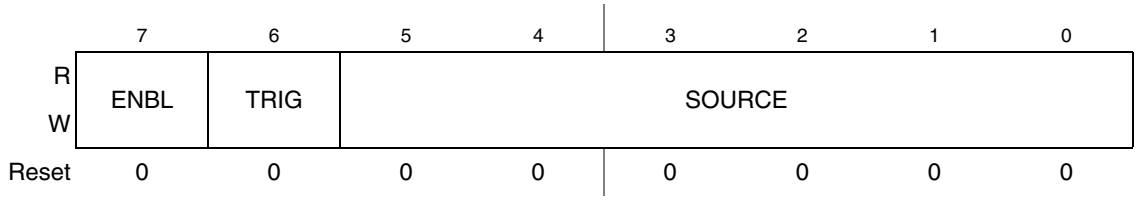


Figure 27-2. Channel Configuration Registers (CHCONFIG*xx*)

Table 27-3. CHCONFIG*xx* Field Descriptions

Field	Description
7 ENBL	DMA Channel Enable. ENBL enables the DMA Channel 0 DMA channel is disabled. This mode is primarily used during configuration of the DMA Mux. The DMA has separate channel enables/disables, which should be used to disable or re-configure a DMA channel. 1 DMA channel is enabled
6 TRIG	DMA Channel Trigger Enable (Channels 0–7 only). TRIG enables the periodic trigger capability for the DMA Channel 0 Triggering is disabled. If triggering is disabled, and the ENBL bit is set, the DMA Channel will simply route the specified source to the DMA channel. 1 Triggering is enabled
5–0 SOURCE	DMA Channel Source. SOURCE specifies which DMA source, if any, is routed to a particular DMA channel, according to Table 27-5.

Table 27-4. Channel and Trigger Enabling

ENBL	TRIG	Function	Mode
0	X	DMA Channel is disabled	Disabled Mode
1	0	DMA Channel is enabled with no triggering (transparent)	Normal Mode
1	1	DMA Channel is enabled with triggering	Periodic Trigger Mode

Table 27-5. SOURCE Configuration

Source	Value
IIC Transmit	0x01
IIC Receive	0x02
DSPI_A Transmit	0x03
DSPI_A Receive	0x04
DSPI_B Transmit	0x05
DSPI_B Receive	0x06
DSPI_C Transmit	0x23
DSPI_C Receive	0x24
ESCI_A Transmit	0x07

Table 27-5. SOURCE Configuration

Source	Value
ESCI_A Receive	0x08
ESCI_B Transmit	0x09
ESCI_B Receive	0x0A
eMIOS Channel 1	0x0F
eMIOS Channel 2	0x10
eMIOS Channel 3	0x11
eMIOS Channel 4	0x12
eMIOS Channel 5	0x13
eMIOS Channel 6	0x14
eMIOS Channel 7	0x15
eMIOS Channel 8	0x16
eMIOS Channel 9	0x17
eMIOS Channel 10	0x18
eMIOS Channel 11	0x19
eMIOS Channel 12	0x1A
eMIOS Channel 13	0x1B
eMIOS Channel 14	0x1C
eMIOS Channel 15	0x1D
eMIOS Channel 16	0x1E
ATD_A Command	0x1F
ATD_A Result	0x20
Always Enabled	0x25
Always Enabled	0x26
Always Enabled	0x27
Always Enabled	0x28
Always Enabled	0x29
Always Enabled	0x2A
Always Enabled	0x2B
Always Enabled	0x2C
DMA Channel is not used (disabled)	All other values

NOTE

Setting multiple CHCONFIG registers with the same Source value will result in unpredictable behavior.

27.4 Functional Description

This section provides a complete functional description of the DMA Mux. The primary purpose of the DMA Mux is to provide flexibility in the system's use of the available DMA channels. As such, configuration of the DMA Mux is intended to be a static procedure done during execution of the system boot code. However, if the procedure outlined in Section 27.7.5, "Enabling and Configuring Sources," is followed, the configuration of the DMA Mux may be changed during the normal operation of the system.

Functionally, the DMA Mux channels may be divided into two classes: Channels 0–7, which implement the normal routing functionality plus periodic triggering capability, and channels 8–15, which implement only the normal routing functionality.

27.4.1 DMA Channels 0-7

Besides the normal routing functionality, channels 0–7 of the DMA Mux provide a special periodic triggering capability that can be used to provide an automatic mechanism to transmit bytes, frames or packets at fixed intervals without the need for processor intervention. The trigger is generated by the Periodic Interrupt Timer (PIT); as such, the configuration of the periodic triggering interval is done via configuration registers in the PIT. Please refer to Chapter 26, "Periodic Interrupt Timer (PIT_RTI)" for more information on this topic.

NOTE

Because of the dynamic nature of the system (i.e. DMA channel priorities, bus arbitration, interrupt service routine lengths, etc.), the number of clock cycles between a trigger and the actual DMA transfer cannot be guaranteed.

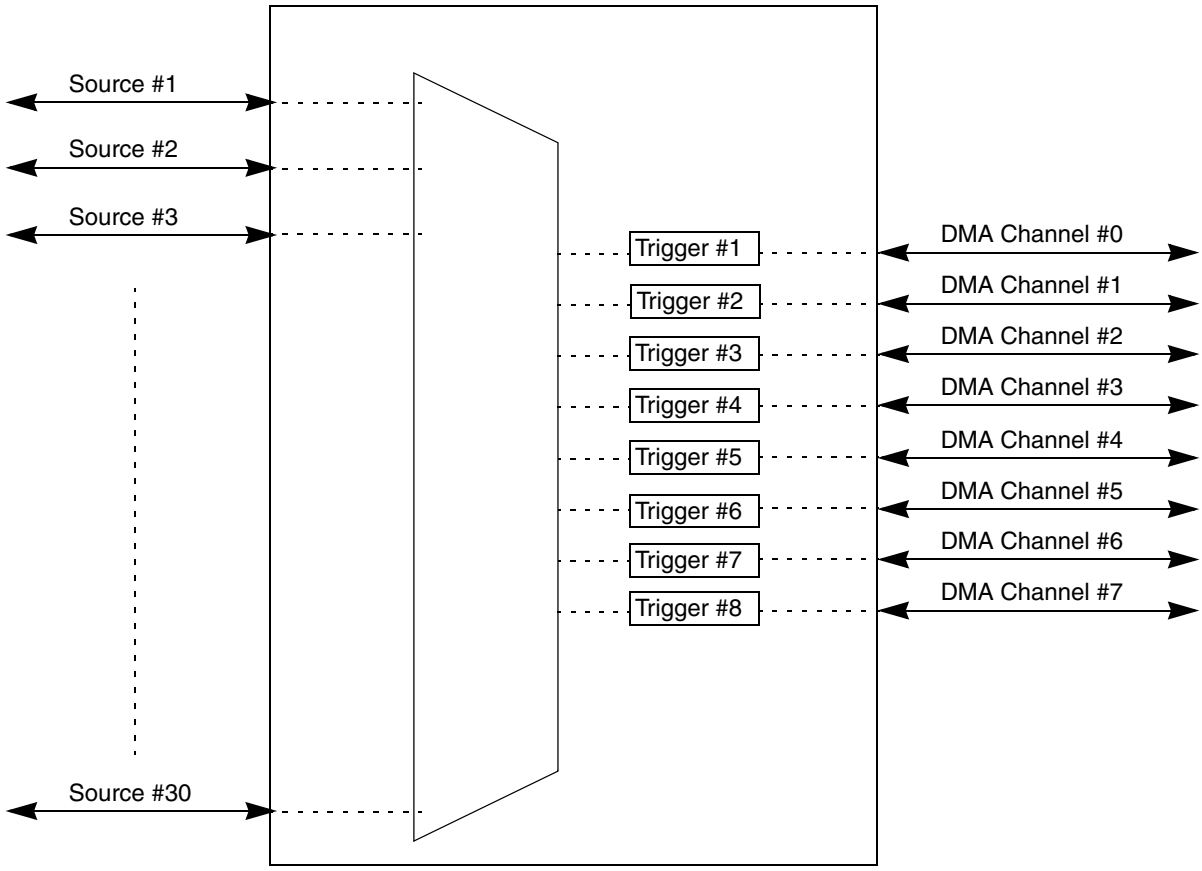


Figure 27-3. DMA Mux Channel 0-7 Block Diagram

The DMA channel triggering capability allows the system to “schedule” regular DMA transfers, usually on the transmit side of certain peripherals, without the intervention of the processor. This trigger works by gating the request from the Peripheral to the DMA until a trigger event has been seen. This is illustrated in Figure 27-4.

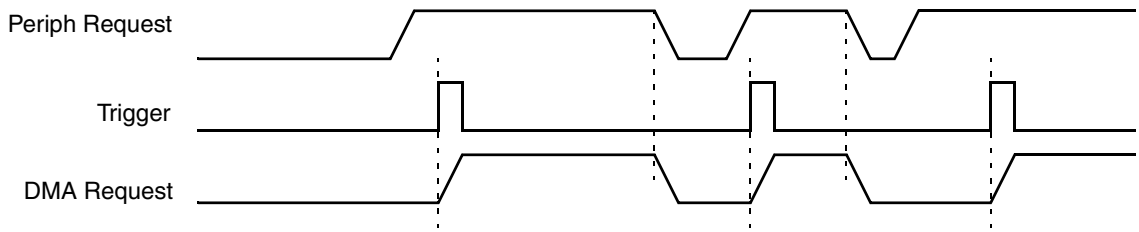


Figure 27-4. DMA Mux Channel Triggering: Normal Operation

Once the DMA request has been serviced, the peripheral will negate its request, effectively resetting the gating mechanism until the peripheral re-asserts its request AND the next trigger event is seen. This means that if a trigger is seen, but the peripheral is not requesting a transfer, that triggered will be ignored. This situation is illustrated in Figure 27-5.

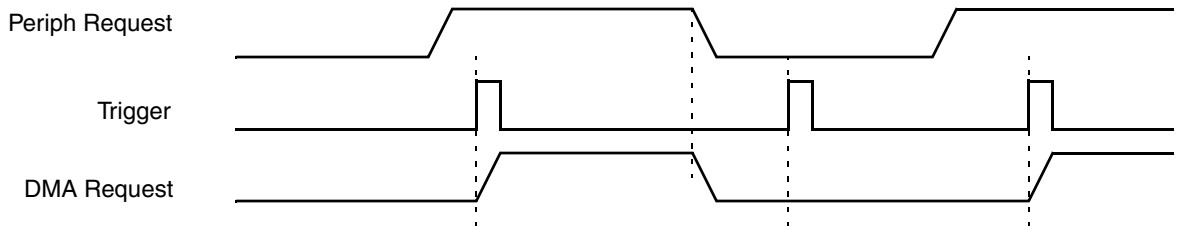


Figure 27-5. DMA Mux Channel Triggering: Ignored Trigger

This triggering capability may be used with any peripheral that supports DMA transfers, and is most useful for two types of situations:

- Periodically polling external devices on a particular bus. As an example, the transmit side of an SPI is assigned to a DMA channel with a trigger, as described above. Once setup, the SPI will request DMA transfers (presumably from memory) as long as its transmit buffer is empty. By using a trigger on this channel, the SPI transfers can be automatically performed every 5µs (as an example). On the receive side of the SPI, the SPI and DMA can be configured to transfer receive data into memory, effectively implementing a method to periodically read data from external devices and transfer the results into memory without processor intervention.
- Using the GPIO Ports to drive or sample waveforms. By configuring the DMA to transfer data to one or more GPIO ports, it is possible to create complex waveforms using tabular data stored in on-chip memory. Conversely, using the DMA to periodically transfer data from one or more GPIO ports, it is possible to sample complex waveforms and store the results in tabular form in on-chip memory.

A more detailed description of the capability of each trigger (i.e.-resolution, range of values, etc.) may be found in Chapter 26, “Periodic Interrupt Timer (PIT_RTI)”.

27.4.2 DMA Channels 8-15

Channels 9–16 of the DMA Mux provide the normal routing functionality as described in Section 27.1.2, “Modes of Operation.”

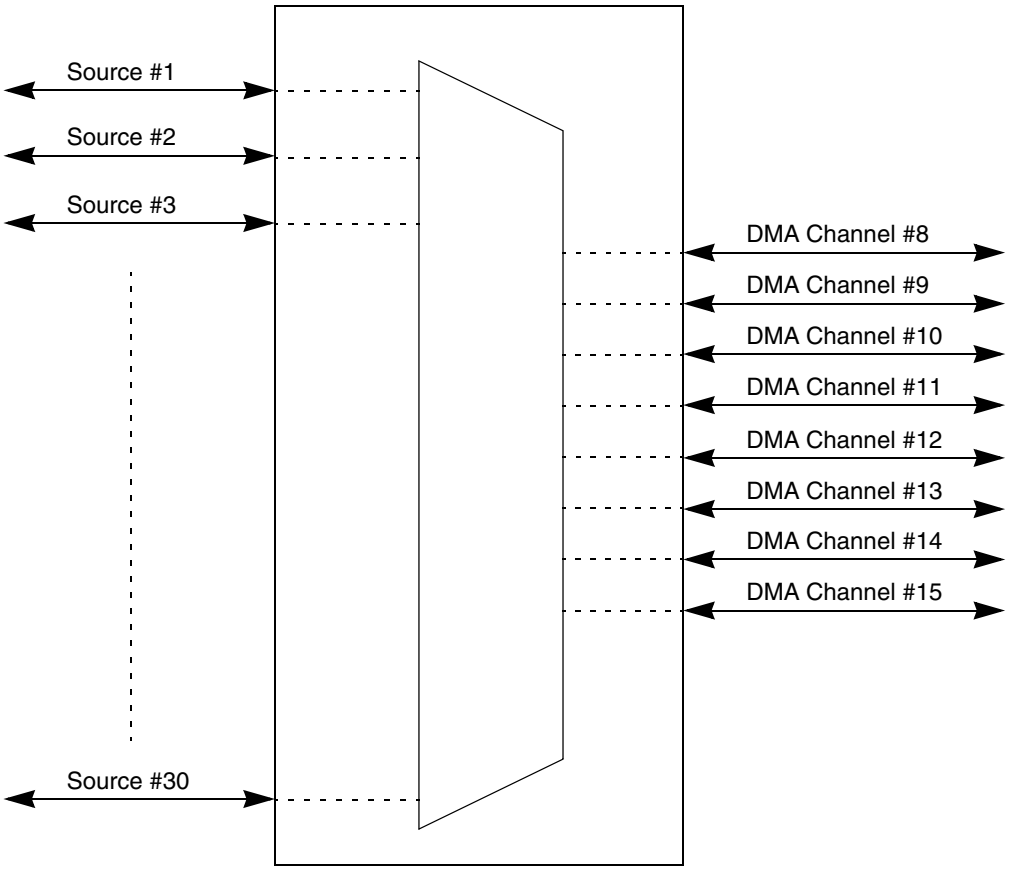


Figure 27-6. DMA Mux Channel 8-15 Block Diagram

27.4.3 "Always Enabled" DMA Sources

In addition to the 30 peripherals that can be used as DMA sources, there are 8 additional DMA sources that are "always enabled". Unlike the peripheral DMA sources, where the peripheral controls the flow of data during DMA transfers, the "always enabled" sources provide no such "throttling" of the data transfers. These sources are most useful in the following cases:

- Doing DMA transfers to/from GPIO - Moving data from/to one or more GPIO pins, either un-throttled (i.e.-as fast as possible), or periodically (using the DMA triggering capability).
- Doing DMA transfers from memory to memory - Moving data from memory to memory, typically as fast as possible, sometimes with software activation.
- Doing DMA transfers from memory to the external bus (or vice-versa) - Similar to memory to memory transfers, this is typically done as quickly as possible.
- Any DMA transfer that requires software activation - Any DMA transfer that should be explicitly started by software.

In cases where software should initiate the start of a DMA transfer, a "always enabled" DMA source can be used to provide maximum flexibility. When activating a DMA channel via software, subsequent

executions of the minor loop require a new "start" event be sent. This can either be a new software activation, or a transfer request from the DMA Channel Mux. The options for doing this are:

- Transfer all data in a single minor loop. By configuring the DMA to transfer all of the data in a single minor loop (i.e.-major loop counter = 1), no re-activation of the channel is necessary. The disadvantage to this option is the reduced granularity in determining the load that the DMA transfer will incur on the system. For this option, the DMA channel should be disabled in the DMA Channel Mux.
- Use explicit software re-activation. In this option, the DMA is configured to transfer the data using both minor and major loops, but the processor is required to re-activate the channel (by writing to the DMA registers) *after every minor loop*. For this option, the DMA channel should be disabled in the DMA Channel Mux.
- Use a "always enabled" DMA source. In this option, the DMA is configured to transfer the data using both minor and major loops, and the DMA Channel Mux does the channel re-activation. For this option, the DMA channel should be enabled and pointing to an "always enabled" source. Note that the re-activation of the channel can be continuous (DMA triggering is disabled) or can use the DMA triggering capability. In this manner, it is possible to execute periodic transfers of packets of data from one source to another, without processor intervention.

27.5 DMA_CH_MUX Bus Aborts

The DMA_CH_MUX module supports Peripheral Bus bus aborts, and enforces the following memory map:

Table 27-6. DMA Channel Mux Bus Aborts

Abort	Allowed
	$0000-$000f
$0010-$3fff	

Supervisor Access: Unused

27.6 DMA_CH_MUX Differences from MAC71xx

- Fixed MUCts01645 (Write on bus abort still writes register)
- Re-allocated peripherals
  - DSPI_C located at $23/$24
  - "Always Enabled" moves from $23-$2A to $25-$2C

Table 27-7. MAC71xx versus MAC72xx DMA Channel Mux Assignment

Value	MAC71x1	MAC72x2	MAC72x1
$00	Disabled	Disabled	Disabled
$01	IIC Transmit	IIC Transmit	IIC Transmit
$02	IIC Receive	IIC Receive	IIC Receive

Table 27-7. MAC71xx versus MAC72xx DMA Channel Mux Assignment (Continued)

Value	MAC71x1	MAC72x2	MAC72x1
$03	DSPI_A Transmit	DSPI_A Transmit	DSPI_A Transmit
$04	DSPI_A Receive	DSPI_A Receive	DSPI_A Receive
$05	DSPI_B Transmit	DSPI_B Transmit	DSPI_B Transmit
$06	DSPI_B Receive	DSPI_B Receive	DSPI_B Receive
$07	ESCI_A Transmit	ESCI_A Transmit	ESCI_A Transmit
$08	ESCI_A Receive	ESCI_A Receive	ESCI_A Receive
$09	ESCI_B Transmit	ESCI_B Transmit	ESCI_B Transmit
$0A	ESCI_B Receive	ESCI_B Receive	ESCI_B Receive
$0B	ESCI_C Transmit	Reserved	Reserved
$0C	ESCI_C Receive	Reserved	Reserved
$0D	ESCI_D Transmit	Reserved	Reserved
$0E	ESCI_D Receive	Reserved	Reserved
$0F	eMIOS Channel 1	eMIOS Channel 1	eMIOS Channel 1
$10	eMIOS Channel 2	eMIOS Channel 2	eMIOS Channel 2
$11	eMIOS Channel 3	eMIOS Channel 3	eMIOS Channel 3
$12	eMIOS Channel 4	eMIOS Channel 4	eMIOS Channel 4
$13	eMIOS Channel 5	eMIOS Channel 5	eMIOS Channel 5
$14	eMIOS Channel 6	eMIOS Channel 6	eMIOS Channel 6
$15	eMIOS Channel 7	eMIOS Channel 7	eMIOS Channel 7
$16	eMIOS Channel 8	eMIOS Channel 8	eMIOS Channel 8
$17	eMIOS Channel 9	Reserved	eMIOS Channel 9
$18	eMIOS Channel 10	Reserved	eMIOS Channel 10
$19	eMIOS Channel 11	Reserved	eMIOS Channel 11
$1A	eMIOS Channel 12	Reserved	eMIOS Channel 12
$1B	eMIOS Channel 13	Reserved	eMIOS Channel 13
$1C	eMIOS Channel 14	Reserved	eMIOS Channel 14
$1D	eMIOS Channel 15	Reserved	eMIOS Channel 15
$1E	eMIOS Channel 16	Reserved	eMIOS Channel 16
$1F	ATD_A Command	ATD_A Command	ATD_A Command
$20	ATD_A Result	ATD_A Result	ATD_A Result
$21	ATD_B Command	Reserved	Reserved
$22	ATD_B Result	Reserved	Reserved
$23	Always Enabled	DSPI_C Transmit	DSPI_C Transmit

Table 27-7. MAC71xx versus MAC72xx DMA Channel Mux Assignment (Continued)

Value	MAC71x1	MAC72x2	MAC72x1
$24	Always Enabled	DSPI_C Receive	DSPI_C Receive
$25	Always Enabled	Always Enabled	Always Enabled
$26	Always Enabled	Always Enabled	Always Enabled
$27	Always Enabled	Always Enabled	Always Enabled
$28	Always Enabled	Always Enabled	Always Enabled
$29	Always Enabled	Always Enabled	Always Enabled
$2A	Always Enabled	Always Enabled	Always Enabled
$2B	Disabled	Always Enabled	Always Enabled
$2C	Disabled	Always Enabled	Always Enabled
$2D - $FF	Disabled	Disabled	Disabled

27.7 Initialization/Application Information

27.7.1 Reset

The reset state of each individual bit is shown within the Register Description section (See Section 27.3.1, “Register Descriptions”). In summary, after reset, all channels are disabled and must be explicitly enabled before use.

27.7.2 Enabling the DMA_CH_MUX

It is not necessary to enable the DMA_CH_MUX before it can be used. However, it must be configured before any DMA transfers can occur, as all DMA Channel Mux channels are disabled by default.

27.7.3 Simple Setup from AG

1. Set up the desired channel in the DMA
2. Connect the channel from step#1 to the desired peripheral by writing the **CHCONFIG*xx*** register in the DMA Channel Mux
3. Enable the desired peripheral by clearing its **MDIS** bit
4. Set up the DMA interface of the peripheral by writing the appropriate register(s) in the peripheral. Please refer to the section for the particular peripheral for more information.

Example 27-1. Configure DMA Channel #0 to be used with ESCI_A (transmit)

1. Set up the desired channel in the DMA
   — Configure **TCD00** in the DMA to transfer from the ESCI_A’s **SCIDRH/L** registers to memory (probably SRAM or Flash)

— Configure error and interrupt handling for DMA Channel #0 by writing the appropriate registers in the DMA

— Write $00 to the **DMASERQ** register in the DMA to enable Channel #0

2. Connect the channel from step#1 to the desired peripheral by writing the **CHCONFIG*xx*** register in the DMA Channel Mux

— Write $87 to **CHCONFIG0** to setup Channel #0 with ESCI_A Transmit

3. Enable the desired peripheral by clearing its **MDIS** bit

— Turn on the module by writing the **MDIS** bit to 0

4. Setup the DMA interface of the peripheral by writing the appropriate register(s) in the peripheral. Please refer to the section for the particular peripheral for more information.

— Select a baud rate. Write this value to the ESCI baud registers (**SCIBDH/L**) to begin the baud rate generator. Remember that the baud rate generator is disabled when the baud rate is zero. Writing to the SCIBDH has no effect without also writing to **SCIBDL**

— Write to **SCICR1** to configure word length, parity, and other configuration bits (**LOOPS,RSRC,M,WAKE,ILT,PE,PT**)

— Enable the transmitter, interrupts, receive, and wake up as required, by writing to the **SCICR2** register bits (**TIE,TCIE,RIE,ILIE,TE,RE,RWU,SBK**). The **TXDMA** bit should be set when the ESCI is used with a DMA. A preamble or idle character will now be shifted out of the transmitter shift register

27.7.4 Using the “Always Enabled” Feature to Periodically Drive GPIO Pins

1. Temporarily disable the desired DMA Channel in the DMA Channel Mux by writing $00 to the corresponding **CHCONFIGxx** register
2. Set up the correct timer interval by writing the correct value into the **TLVAL** register in the PIT

   The timer reload value for step PIT2) can be calculated with the following formula:

$$\frac{\text{trigger period}}{\text{clock period}} - 1 = \text{timer reload value} \qquad \textit{Eqn. 27-1}$$

   For example: The system clock is 66.67Mhz ($T_{SYSCLOCK}$ = 15ns), which gives us a bus clock of 33.33Mhz ($T_{BUSCLOCK}$ = 30ns). The desired trigger period is 30µs.

$$\frac{30\mu s}{30ns} - 1 = 999 = 0x0000\_03E7 \qquad \textit{Eqn. 27-2}$$

NOTE

All PIT timers, except the RTI, run off of the Bus Clock, which is 1/2 the frequency of the System Clock.

3. Enable the PIT trigger by setting the corresponding bit in the **PITEN** register in the PIT
4. Configure the desired pin(s) to be General Purpose Outputs by writing $40 to the corresponding **CONFIGxx** register(s) in the PIM
5. Configure the Transfer Control Descriptor (TCD) for the given DMA Channel by writing to the **TCDxx** registers in the DMA. The Source Address will probably be a location in the Flash or SRAM memories, and the Destination Address will be the **PORTDATA** register (with a minor

counter = 1) or the **PINDATAxx** registers (with a minor counter > 1). Note that if a pin is configured as either Peripheral Mode or General Purpose Input mode, then writing to the **PORTDATA** register will have no effect on that particular pin. In this manner, it is possible to simultaneously write to less than 16 pins in a port, using the **PORTDATA** register. If, however, you have General Purpose Output pins in the port that you do not wish to change, you must use the **PINDATAxx** registers. In this case, the pins must be contiguous (i.e.-PA3,PA4,PA5,PA6 for instance)

6. If any interrupt or error handling needs to be configured for the DMA Channel, do this by writing the appropriate registers in the DMA
7. Write the appropriate **DMASERQ** register in the DMA to enable the channel
8. Enable the DMA Channel in the DMA Channel Mux (with triggering) by setting the **ENBL** and **TRIG** bits in the appropriate **CHCONFIGxx** register in the DMA Channel Mux. Choose an “Always Enabled” source (i.e.-$25 to $2C). This can be done by writing $E5...$EC to the **CHCONFIGxx** register. You should only use an “Always Enabled” source for a single DMA Channel. It is recommended to use $25 for Channel #0, $26 for Channel#1, etc.

27.7.5 Enabling and Configuring Sources

27.7.5.1 Enabling a Source with Periodic Triggering

1. Determine with which DMA channel the source will be associated. Remember that only DMA channels 0–7 have periodic triggering capability
2. Clear the ENBL and TRIG bits of the DMA channel
3. Ensure that the DMA channel is properly configured in the DMA. The DMA channel may be enabled at this point
4. In the PIT, configure the corresponding timer
5. Select the source to be routed to the DMA channel. Write to the corresponding CHCONFIG register, ensuring that the ENBL and TRIG bits are set

Example 27-2. Configure DSPI_B Transmit for use with DMA Channel 2, with periodic triggering capability

1. Write 0x00 to CHCONFIG2 (Base Address + 0x02)
2. Configure Channel 2 in the DMA, including enabling the channel
3. Configure Timer 3 in the Periodic Interrupt Timer (PIT) for the desired trigger interval
4. Write 0xC5 to CHCONFIG2 (Base Address + 0x02)

The following code example illustrates steps #1 and #4 above:

```
In File registers.h:
#define DMAMUX_BASE_ADDR     0xFC084000/* Example only ! */
/* Following example assumes char is 8-bits */
volatile unsigned char *CHCONFIG0 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0000);
volatile unsigned char *CHCONFIG1 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0001);
volatile unsigned char *CHCONFIG2 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0002);
volatile unsigned char *CHCONFIG3 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0003);
volatile unsigned char *CHCONFIG4 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0004);
volatile unsigned char *CHCONFIG5 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0005);
```

```
volatile unsigned char *CHCONFIG6 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0006);
volatile unsigned char *CHCONFIG7 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0007);
volatile unsigned char *CHCONFIG8 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0008);
volatile unsigned char *CHCONFIG9 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0009);
volatile unsigned char *CHCONFIG10= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000A);
volatile unsigned char *CHCONFIG11= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000B);
volatile unsigned char *CHCONFIG12= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000C);
volatile unsigned char *CHCONFIG13= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000D);
volatile unsigned char *CHCONFIG14= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000E);
volatile unsigned char *CHCONFIG15= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000F);

In File main.c:
#include "registers.h"
         :
         :
*CHCONFIG2 = 0x00;
*CHCONFIG2 = 0xC5;
```

27.7.5.2 Enabling a Source without Periodic Triggering

1. Determine with which DMA channel the source will be associated. Remember that only DMA channels 0–7 have periodic triggering capability
2. Clear the ENBL and TRIG bits of the DMA channel
3. Ensure that the DMA channel is properly configured in the DMA. The DMA channel may be enabled at this point
4. Select the source to be routed to the DMA channel. Write to the corresponding CHCONFIG register, ensuring that the ENBL is set and the TRIG bit is cleared

Example 27-3. Configure DSPI_B Transmit for use with DMA Channel 2, with no periodic triggering capability.

1. Write 0x00 to CHCONFIG2 (Base Address + 0x02)
2. Configure Channel 2 in the DMA, including enabling the channel
3. Write 0x85 to CHCONFIG2 (Base Address + 0x02)

The following code example illustrates steps #1 and #3 above:

```
In File registers.h:
#define DMAMUX_BASE_ADDR     0xFC084000/* Example only ! */
/* Following example assumes char is 8-bits */
volatile unsigned char *CHCONFIG0 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0000);
volatile unsigned char *CHCONFIG1 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0001);
volatile unsigned char *CHCONFIG2 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0002);
volatile unsigned char *CHCONFIG3 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0003);
volatile unsigned char *CHCONFIG4 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0004);
volatile unsigned char *CHCONFIG5 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0005);
volatile unsigned char *CHCONFIG6 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0006);
volatile unsigned char *CHCONFIG7 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0007);
volatile unsigned char *CHCONFIG8 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0008);
volatile unsigned char *CHCONFIG9 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0009);
volatile unsigned char *CHCONFIG10= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000A);
volatile unsigned char *CHCONFIG11= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000B);
```

```
volatile unsigned char *CHCONFIG12= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000C);
volatile unsigned char *CHCONFIG13= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000D);
volatile unsigned char *CHCONFIG14= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000E);
volatile unsigned char *CHCONFIG15= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000F);

In File main.c:
#include "registers.h"
         :
         :
*CHCONFIG2 = 0x00;
*CHCONFIG2 = 0x85;
```

27.7.6 Disabling a Source

A particular DMA source may be disabled by not writing the corresponding source value into any of the CHCONFIG registers. Additionally, some module specific configuration may be necessary. Please refer to the appropriate section for more details.

27.7.6.1 Switching the Source of a DMA Channel

1. Disable the DMA channel in the DMA and re-configure the channel for the new source
2. Clear the ENBL and TRIG bits of the DMA channel
3. Select the source to be routed to the DMA channel. Write to the corresponding CHCONFIG register, ensuring that the ENBL and TRIG bits are set

Example 27-4. Switch DMA Channel 8 from DSPI_A transmit to ESCI_A transmit

1. In the DMA configuration registers, disable DMA channel 8 and re-configure it to handle the DSPI_A transmits
2. Write 0x00 to CHCONFIG8 (Base Address + 0x08)
3. Write 0x87 to CHCONFIG8 (Base Address + 0x08). In this case, setting the TRIG bit would have no effect, because channels 8–15 do not support the periodic triggering functionality

The following code example illustrates steps #2 and #4 above:

```
In File registers.h:
#define DMAMUX_BASE_ADDR     0xFC084000/* Example only ! */
/* Following example assumes char is 8-bits */
volatile unsigned char *CHCONFIG0 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0000);
volatile unsigned char *CHCONFIG1 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0001);
volatile unsigned char *CHCONFIG2 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0002);
volatile unsigned char *CHCONFIG3 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0003);
volatile unsigned char *CHCONFIG4 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0004);
volatile unsigned char *CHCONFIG5 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0005);
volatile unsigned char *CHCONFIG6 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0006);
volatile unsigned char *CHCONFIG7 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0007);
volatile unsigned char *CHCONFIG8 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0008);
volatile unsigned char *CHCONFIG9 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0009);
volatile unsigned char *CHCONFIG10= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000A);
volatile unsigned char *CHCONFIG11= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000B);
volatile unsigned char *CHCONFIG12= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000C);
volatile unsigned char *CHCONFIG13= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000D);
volatile unsigned char *CHCONFIG14= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000E);
```

```
volatile unsigned char *CHCONFIG15= (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x000F);

In File main.c:
#include "registers.h"
         :
         :
*CHCONFIG8 = 0x00;
*CHCONFIG8 = 0x87;
```

Chapter 28
FlexCAN2

28.1 FlexCAN2 Implementation on the MAC7200

28.1.1 Introduction

The FlexCAN2 module complies to the CAN2.0B specification and therefore supports transfer rates of up to 1Mbit for high speed CAN networks.

On the MAC7200 family of devices, the clock source for the FlexCAN2 can be selected to be either derived from the PLL clock, or the oscillator clock. Each CAN module in the MAC7200 family of devices contains 32 mailboxes.

Each of the FlexCAN2 modules can be independently disabled by writing to the MDIS bit in the modules MCR register. Disabling the module will turn off the clock to the modules protocol engine and message buffers, although most of the modules registers remain available to be accessed by the core across the peripheral bus. The MDIS bit is intended to be used when the module is not required in the application. By default the FlexCANs are disabled after reset (MDIS=1), so, prior to use, the MDIS bit must be cleared.

Unlike most of the other peripherals on the MAC72xx, the CAN modules are not masters. This means that data coming into the MAC72xx on a CAN interface may not be under the software control of the MAC72xx. For this reason, it is necessary for the CAN Message ID (i.e.-IDENTIFIER) to be extracted from a CAN frame for the processor to react to it correctly. The FlexCAN module does this, and stores the message corresponding to a particular Message ID in a particular mailbox (up to a limit of 32 mailboxes).

See Section 28.2, “The Generic FlexCAN2 Module” for generic information on the Flexible CAN Module.

28.1.2 Features of FlexCAN2 on MAC7200

- Memory Map: 32-bit peripheral with 8 bytes, byte/halfword/word addressable
- Full implementation of the CAN 2.0 protocol specification.
- Programmable bit rate up to 1Mbps.
- 32 flexible Mail Boxes of 0-8 bytes data length on all modules.
- All Mail Boxes configurable for either Rx/Tx.
- Unused Mail Boxes space can be used as general purpose RAM.
- Supports standard or extended messages.
- “Time Stamp”, based on a 16-bit free-running counter.

- Maskable interrupts.
- Programmable I/O modes.
- External transceiver assumed.

28.1.3 CAN Protocol

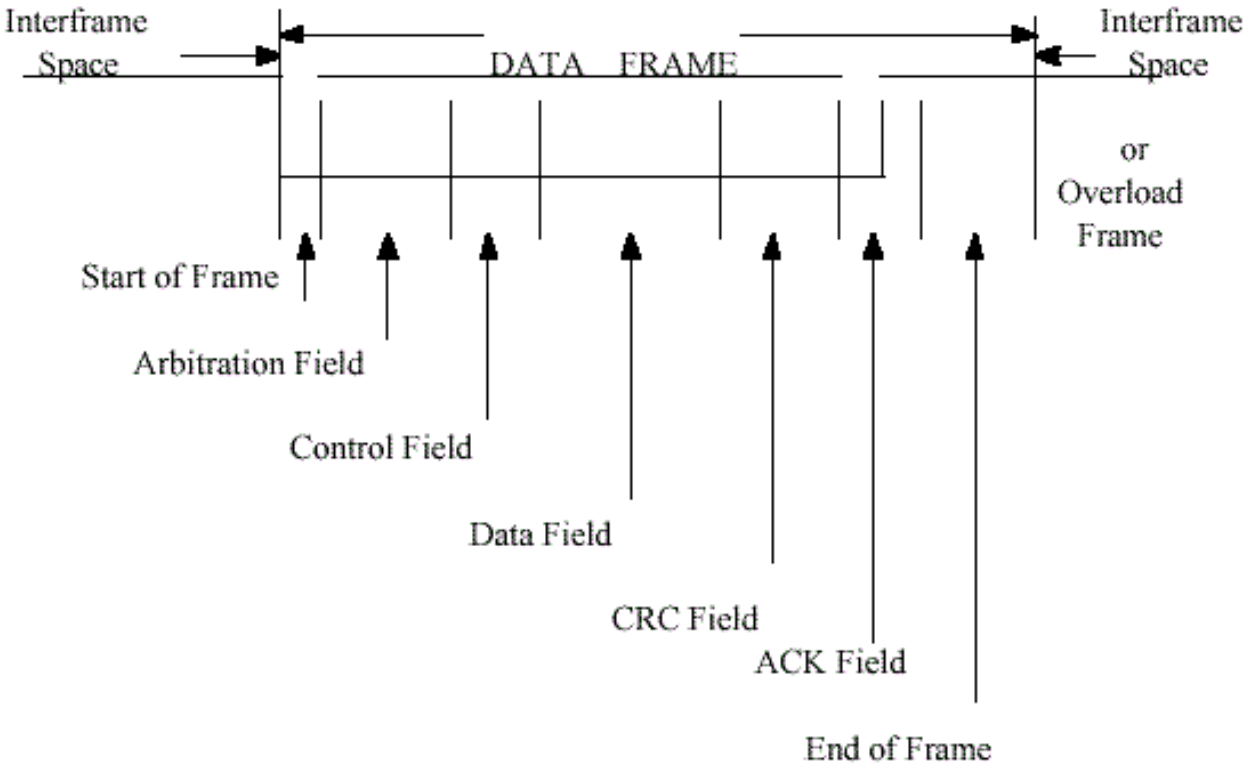
28.1.3.1 Terminology

- **Dominant** — Typically logic level ‘0’
- **Recessive** — Typically logic level ‘1’

28.1.3.2 Data Frame

A Data Frame carries data from a transmitter to the receivers. Length varies from 45 to 109 bits, depending on the number of bytes in the data payload (including minimum interframe space).

- INTERFRAME SPACE - Minimum of 1 recessive bit [1 bit]
- START OF FRAME - 1 dominant bit [1 bit]
- ARBITRATION FIELD - [12 bits]
  - IDENITIFIER - The Message ID [11 bits]
  - RTR-bit - 1 dominant bit [1 bit]
- CONTROL FIELD - [6 bits]
  - RESERVED - Reserved for future use [2 bits]
  - DATA LENGTH CODE - Specifies the number of data bytes in the payload [4 bits]
- DATA FIELD - The data payload, ranging from 0 to 8 bytes [0 to 64 bits]
- CRC FIELD - [16 bits (?)]
  - CRC SEQUENCE - The frame check sequence [15 bits (?)]
  - CRC DELIMITER - 1 recessive bit [1 bit]
- ACK FIELD - [2 bits]
  - ACK SLOT - Acknowledge reception of the CRC sequence with 1 dominant bit [1 bit]
  - ACK DELIMITER - 1 recessive bit [1 bit]
- END OF FRAME - [7 bits]

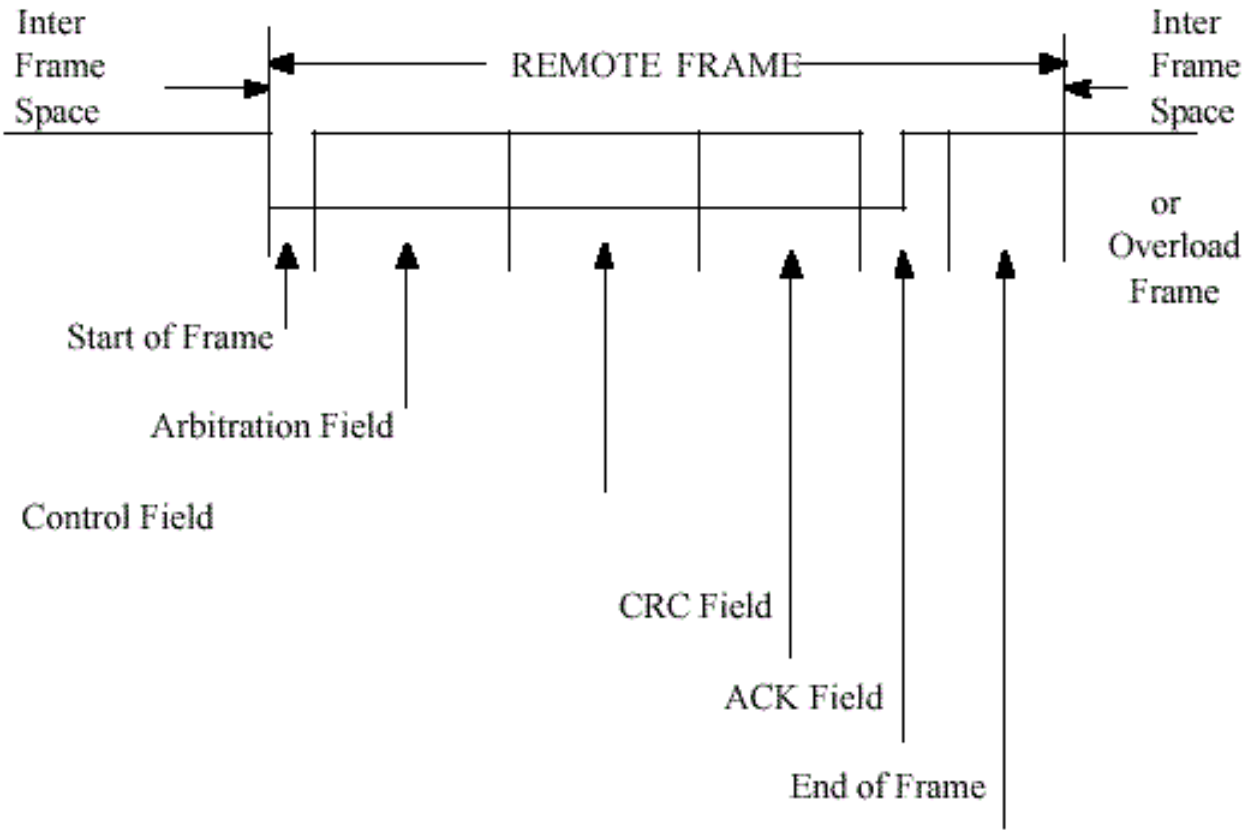


28.1.3.3 Remove Frame

A Remote Frame is transmitted by a bus unit to request the transmission of the DATA FRAME with the same IDENTIFIER. The length is 45 bits (including minimum interframe space).

- INTERFRAME SPACE - Minimum of 1 recessive bit [1 bit]
- START OF FRAME - 1 dominant bit [1 bit]
- ARBITRATION FIELD - [12 bits]
  - IDENITIFIER - The Message ID [11 bits]
  - RTR-bit - 1 recessive bit [1 bit]
- CONTROL FIELD - [6 bits]
  - RESERVED - Reserved for future use [2 bits]
  - DATA LENGTH CODE - Specifies the number of data bytes in the payload [4 bits]
- CRC FIELD - [16 bits (?)]
  - CRC SEQUENCE - The frame check sequence [15 bits (?)]
  - CRC DELIMITER - 1 recessive bit [1 bit]
- ACK FIELD - [2 bits]
  - ACK SLOT - Acknowledge reception of the CRC sequence with 1 dominant bit [1 bit]
  - ACK DELIMITER - 1 recessive bit [1 bit]

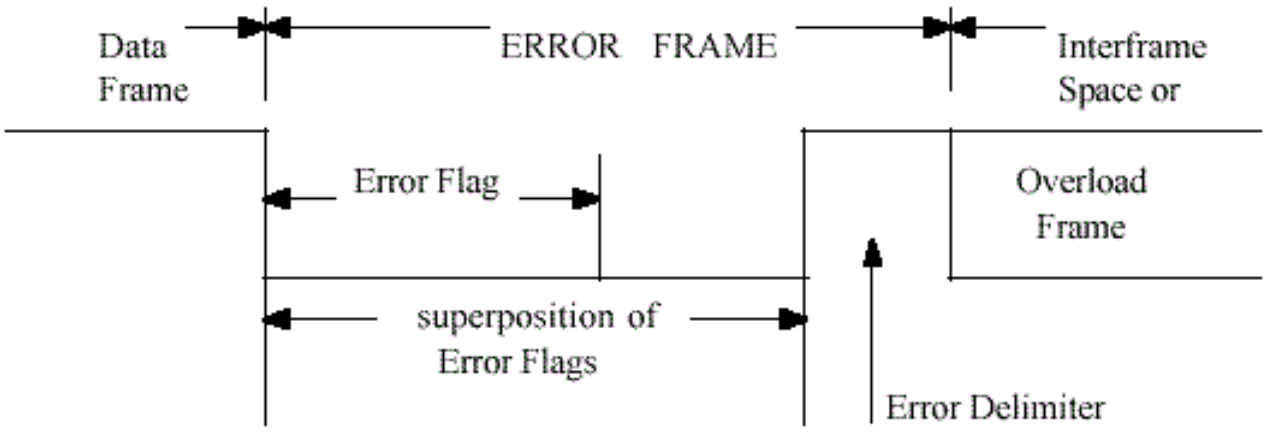
- END OF FRAME - [7 bits]



28.1.3.4 Error Frame

An Error Frame is transmitted by any unit on detecting a bus error. The length varies from 14 to 20 bits.

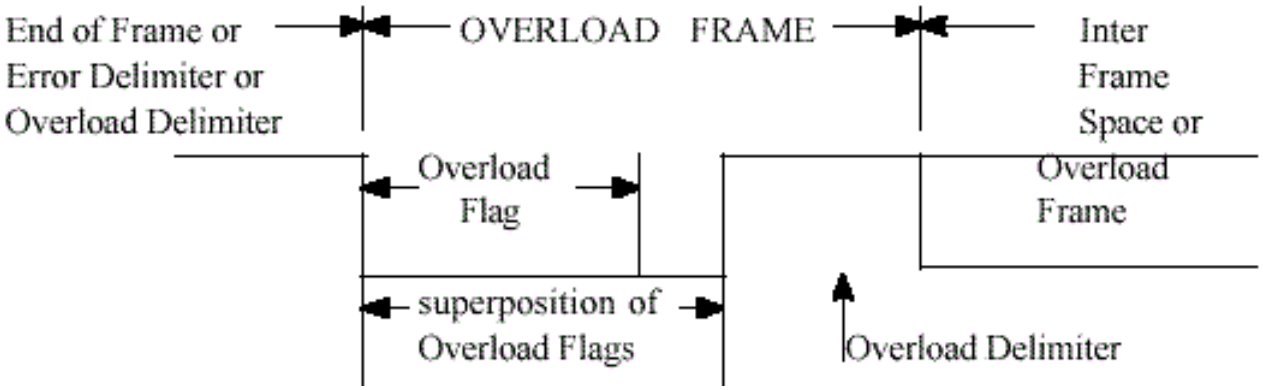
- ERROR FLAG - The superposition of all ERROR FLAG fields from all stations [6 to 12 bits]
- ERROR DELIMITER - 8 recessive bits [8 bits]



28.1.3.5 Overload Frame

An Overload Frame is used to provide for an extra delay between the preceding and the succeeding DATA or REMOTE FRAMEs. The length varies from 14 to 20 bits.

- OVERLOAD FLAG - The superposition of all OVERLOAD FLAG fields from all stations [6 to 12 bits]
- OVERLOAD DELIMITER - 8 recessive bits [8 bits]



28.1.4 CAN Implementation

The FlexCAN module has several different configuration options. On the MAC72xx device, the FlexCAN modules have the following configuration:

- Number of Message Buffers (Mail Boxes): 32 per FlexCAN
- MDIS Reset Value: The reset value of the **MDIS** bit is 1, meaning that the FlexCAN is in a disabled state after reset, and must be explicitly enabled before it can be used.
- Rx Masking: Receive masking is not available on the MAC72xx FlexCANs.
- Reception Queue: Reception queues are not available on the MAC72xx FlexCANs.
- Wakeup: Because there is no STOP mode on the MAC72xx, the wakeup from bus activity feature of the FlexCAN is only valid in DOZE mode. In this case, the FlexCAN can be left running, and any FlexCAN interrupt can be used to wake the system.

28.1.5 CAN External Pins

Table 28-1. CAN External Pins

Signal	Description
RxD_*n*	This input pin provides the serial receive side of the FlexCAN bus.
TxD_*n*	This output pin provides the serial transmit side of the FlexCAN bus.

28.1.6 FlexCAN Bus Aborts

The FlexCAN module supports Peripheral Bus bus aborts, and enforces the following memory map:

Table 28-2. FlexCAN Memory Map

Address Offset	Register	Modes Allowed[1]
$0000	Module Configuration (MCR)	S
$0004	Control Register (CTRL)	S/U
$0008	Free Running Timer (TIMER)	S/U
$000C	Reserved	NONE
$0010	Rx Global Mask (RXGMASK)	S/U
$0014	Rx Buffer 14 Mask (RX14MASK)	S/U
$0018	Rx Buffer 15 Mask (RX15MASK)	S/U
$001C	Error Counter Register (ECR)	S/U
$0020	Error and Status Register (ESR)	S/U
$0024	Interrupt Masks 2 (IMASK2)	S/U
$0028	Interrupt Masks 1 (IMASK1)	S/U
$002C	Interrupt Flags 2 (IFLAG2)	S/U
$0030	Interrupt Flags 1 (IFLAG1)	S/U
$0034 - $005F	Reserved	NONE
$0060 - $007F	Reserved	NONE
$0080 - $017F	Message Buffers MB0 - MB15	S/U
$0180 - $027F	Message Buffers MB16 - MB31	S/U
$0280 - $3FFF	Reserved	NONE

1. **S** Supervisor Mode Access
   U User Mode Access

Supervisor Access: All User Mode accesses to addresses not marked with a ‘U’ are aborted.

28.1.7 CAN Differences from MAC71xx

- Fixed MUCts01374: Message Buffer deactivation in Bus Off holds arbitration
- Removed wakeup glitch filter (and hence wakeup on bus activity)

28.1.8 CAN Application Usage

28.1.8.1 Enabling the CAN

Before the FlexCAN can be used, it must be explicitly enabled by clearing the **MDIS** bit.

28.1.8.2 Message Buffer Initialization

Before being used, all FlexCAN message buffers must be initialized by writing ‘0’ into all locations. Since the FlexCAN Message Buffer Memory does not have ECC, there is no requirement to initialize the memory in a specific manner.

28.2 The Generic FlexCAN2 Module

The FlexCAN module is a communication controller implementing the CAN protocol according to the CAN 2.0B protocol specification.

28.2.1 Block Diagram

A general block diagram is shown in Figure 28-1, which describes the main sub-blocks implemented in the FlexCAN module, including two embedded memories, one for storing Message Buffers (MB) and another one for storing Rx Individual Mask Registers. On the MAC72xx, there are 32 Message Buffers for each FlexCAN module. The functions of the sub-modules are described in subsequent sections.

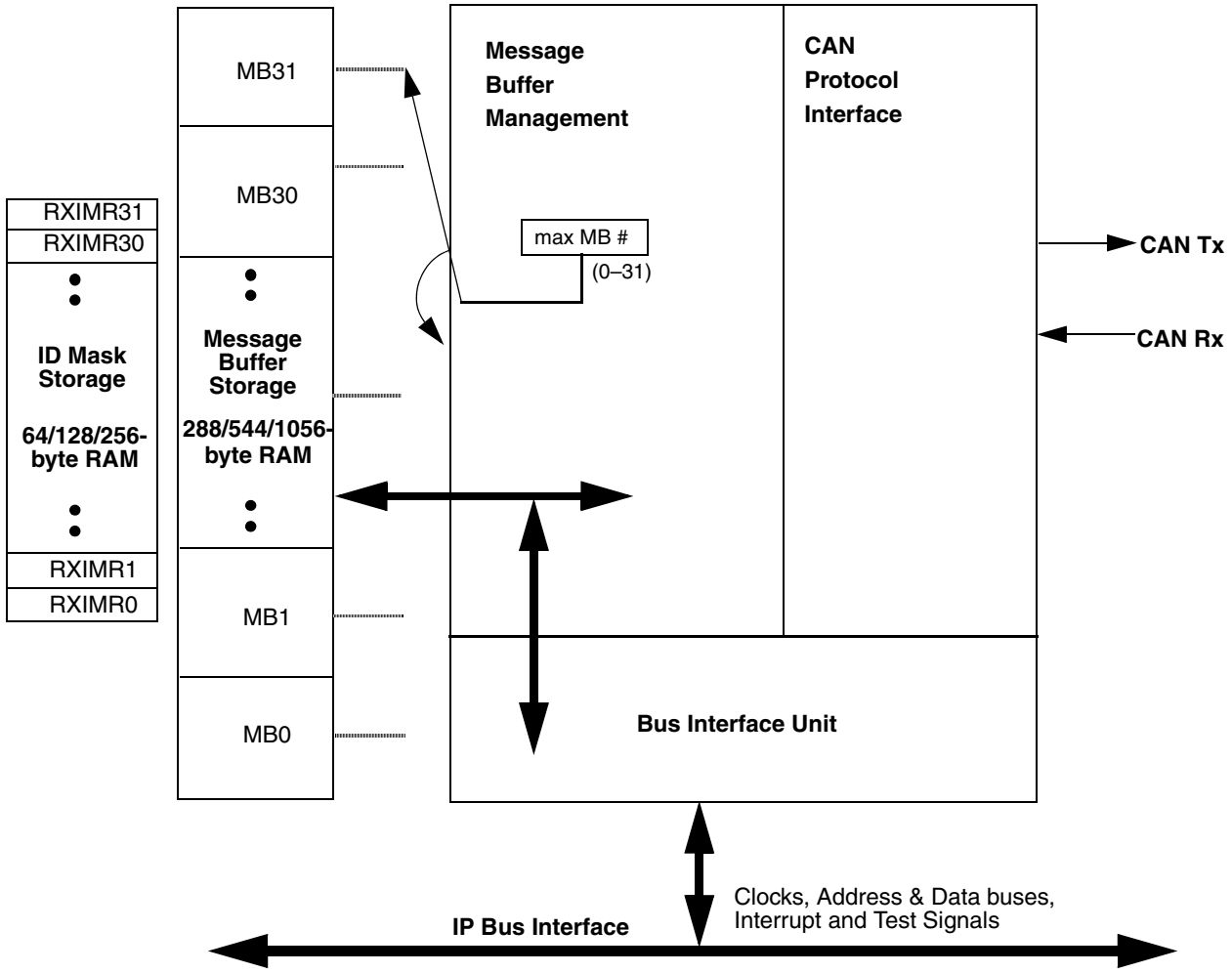


Figure 28-1. FlexCAN Block Diagram

28.2.2 Overview

The CAN protocol was primarily, but not only, designed to be used as a vehicle serial data bus, meeting the specific requirements of this field: real-time processing, reliable operation in the EMI environment of a vehicle, cost-effectiveness and required bandwidth. The FlexCAN module is a full implementation of the CAN protocol specification, Version 2.0 B [Ref. 1], which supports both standard and extended message frames. A flexible number of Message Buffers (16, 32 or 64) is also supported. The Message Buffers are stored in an embedded RAM dedicated to the FlexCAN module. On the MAC72xx, there are 32 Message Buffers for each FlexCAN module.

The CAN Protocol Interface (CPI) sub-module manages the serial communication on the CAN bus, requesting RAM access for receiving and transmitting message frames, validating received messages and performing error handling. The Message Buffer Management (MBM) sub-module handles Message Buffer selection for reception and transmission, taking care of arbitration and ID matching algorithms. The Bus Interface Unit (BIU) sub-module controls the access to and from the internal interface bus, in order to

establish connection to the CPU and to other blocks. Clocks, address and data buses, interrupt outputs and test signals are accessed through the Bus Interface Unit.

28.2.3 Features

The FlexCAN module includes these distinctive features:

- Full Implementation of the CAN protocol specification, Version 2.0B
  - — Standard data and remote frames
  - — Extended data and remote frames
  - — Zero to eight bytes data length
  - — Programmable bit rate up to 1 Mb/sec
  - — Content-related addressing
- Flexible Message Buffers (up to 64) of zero to eight bytes data length
- Each MB configurable as Rx or Tx, all supporting standard and extended messages
- Individual Rx Mask Registers per Message Buffer
- Includes either 1056 bytes (64 MBs), 544 bytes (32 MBs) or 288 bytes (16 MBs) of RAM used for MB storage
- Includes either 256 bytes (64 MBs), 128 bytes (32 MBs) or 64 bytes (16 MBs) of RAM used for individual Rx Mask Registers
- Reception queue possible by setting more than one Rx Message Buffer with the same ID
- Single-bit selectable backwards compatibility with previous FlexCAN version
- Programmable clock source to the CAN Protocol Interface, either bus clock or crystal oscillator
- Unused MB and Rx Mask Register space can be used as general purpose RAM space
- Listen only mode capability
- Programmable loop-back mode supporting self-test operation
- Programmable transmission priority scheme: lowest ID or lowest buffer number
- Time Stamp based on 16-bit free-running timer
- Global network time, synchronized by a specific message
- Maskable interrupts
- Independent of the transmission medium (an external transceiver is assumed)
- Short latency time due to an arbitration scheme for high-priority messages
- Low power modes, with programmable wake up on bus activity

NOTE

The individual Rx Mask per Message Buffer feature may not be available in low cost MCUs. Please consult the specific MCU documentation to find out if this feature is supported.

28.2.4 Modes of Operation

The FlexCAN module has four functional modes: Normal Mode (User and Supervisor), Freeze Mode, Listen-Only Mode and Loop-Back Mode. There are also three low power modes: Disable Mode, Doze Mode and Stop Mode.

28.2.4.1 Normal Mode (User or Supervisor):

In Normal Mode, the module operates receiving and/or transmitting message frames, errors are handled normally and all the CAN Protocol functions are enabled. User and Supervisor Modes differ in the access to some restricted control registers.

28.2.4.2 Freeze Mode:

It is enabled when the FRZ bit in the MCR Register is asserted. If enabled, Freeze Mode is entered when the HALT bit in MCR is set or when Debug Mode is requested at MCU level. In this mode, no transmission or reception of frames is done and synchronicity to the CAN bus is lost. See Section 28.2.7.8.1, “Freeze Mode,” for more information.

28.2.4.3 Listen-Only Mode:

The module enters this mode when the LOM bit in the Control Register is asserted. In this mode, transmission is disabled, all error counters are frozen and the module operates in a CAN Error Passive mode. Only messages acknowledged by another CAN station will be received. If FlexCAN detects a message that has not been acknowledged, it will flag a BIT0 error (without changing the REC), as if it was trying to acknowledge the message.

28.2.4.4 Loop-Back Mode:

The module enters this mode when the LPB bit in the Control Register is asserted. In this mode, FlexCAN performs an internal loop back that can be used for self test operation. The bit stream output of the transmitter is internally fed back to the receiver input. The Rx CAN input pin is ignored and the Tx CAN output goes to the recessive state (logic ‘1’). FlexCAN behaves as it normally does when transmitting and treats its own transmitted message as a message received from a remote node. In this mode, FlexCAN ignores the bit sent during the ACK slot in the CAN frame acknowledge field to ensure proper reception of its own message. Both transmit and receive interrupts are generated.

28.2.4.5 Module Disable Mode:

This low power mode is entered when the MDIS bit in the MCR Register is asserted. When disabled, the module shuts down the clocks to the CAN Protocol Interface and Message Buffer Management sub-modules. Exit from this mode is done by negating the MDIS bit in the MCR Register. See Section 28.2.7.8.2, “Module Disable Mode,” for more information.

28.2.4.6 Doze Mode:

This low power mode is entered when the DOZE bit in MCR is asserted and Doze Mode is requested at MCU level. When in Doze Mode, the module shuts down the clocks to the CAN Protocol Interface and the Message Buffer Management sub-modules. Exit from this mode happens when the DOZE bit in MCR is negated, when the MCU is removed from Doze Mode, or when activity is detected on the CAN bus and the Self Wake Up mechanism is enabled. See Section 28.2.7.8.3, “Doze Mode,” for more information.

28.2.4.7 Stop Mode:

This low power mode is entered when Stop Mode is requested at MCU level. When in Stop Mode, the module puts itself in an inactive state and then informs the CPU that the clocks can be shut down globally. Exit from this mode happens when the Stop Mode request is removed or when activity is detected on the CAN bus and the Self Wake Up mechanism is enabled. See Section 28.2.7.8.4, “Stop Mode,” for more information.

28.2.5 External Signal Descriptions

The FlexCAN module has two I/O signals connected to the external MCU pins. These signals are summarized in Table 28-3 and described in more detail in the next sub-sections.

Table 28-3. FlexCAN Signals

Signal Name[1]	Direction	Description
CAN Rx	Input	CAN Receive Pin
CAN Tx	Output	CAN Transmit Pin

1. The actual MCU pins may have different names. Please consult Chapter 9, “Device Memory Map” for the actual signal names.

28.2.5.1 CAN Rx

This pin is the receive pin from the CAN bus transceiver. Dominant state is represented by logic level ‘0’. Recessive state is represented by logic level ‘1’.

28.2.5.2 CAN Tx

This pin is the transmit pin to the CAN bus transceiver. Dominant state is represented by logic level ‘0’. Recessive state is represented by logic level ‘1’.

28.2.6 Memory Map and Register Definition

This section describes the registers and data structures in the FlexCAN module. The base address of the module depends on the particular memory map of the MCU. The addresses presented here are relative to the base address.

The address space occupied by FlexCAN has 96 bytes for registers starting at the module base address, followed by MB storage space in embedded RAM starting at address $0060, and an extra ID Mask storage space in a separate embedded RAM starting at address $0880.

28.2.6.1 Memory Map

The complete memory map for a FlexCAN module with 64 MBs capability is shown in Table 28-4. Each individual register is identified by its complete name and the corresponding mnemonic. The access type can be Supervisor (S) or Unrestricted (U). Most of the registers can be configured to have either Supervisor or Unrestricted access by programming the SUPV bit in the MCR Register. These registers are identified as S/U in the Access column of Table 28-4.

The IFLAG2 and IMASK2 registers are considered reserved space when FlexCAN is configured with 16 or 32 MBs. The Rx Global Mask (RXGMASK), Rx Buffer 14 Mask (RX14MASK) and the Rx Buffer 15 Mask (RX15MASK) registers are provided for backwards compatibility, and are not used when the BCC bit in MCR is asserted.

The address ranges $0060–$047F and $0880–$097F are occupied by two separate embedded memories. These two ranges are completely occupied by RAM (1056 and 256 bytes, respectively) only when FlexCAN is configured with 64 MBs. When it is configured with 16 MBs, the memory sizes are 288 and 64 bytes, so the address ranges $0180–$047F and $08C0–$097F are considered reserved space. When it is configured with 32 MBs, the memory sizes are 544 and 128 bytes, so the address ranges $0280–$047F and $0900–$097F are considered reserved space. Furthermore, if the BCC bit in MCR is negated, then the whole Rx Individual Mask Registers address range ($0880–$097F) is considered reserved space.

NOTE

The individual Rx Mask per Message Buffer feature may not be available in low cost MCUs. Please consult the specific MCU documentation to find out if this feature is supported. If not supported, the address range $0880-$097F is considered reserved space, independent of the value of the BCC bit.

Table 28-4. Module Memory Map

Address	Use	Access Type	Affected by Hard Reset	Affected by Soft Reset
Base + $0000	Module Configuration (MCR)	S	Yes	Yes
Base + $0004	Control Register (CTRL)	S/U	Yes	No
Base + $0008	Free Running Timer (TIMER)	S/U	Yes	Yes
Base + $000C	Reserved			
Base + $0010	Rx Global Mask (RXGMASK)	S/U	Yes	No
Base + $0014	Rx Buffer 14 Mask (RX14MASK)	S/U	Yes	No
Base + $0018	Rx Buffer 15 Mask (RX15MASK)	S/U	Yes	No
Base + $001C	Error Counter Register (ECR)	S/U	Yes	Yes
Base + $0020	Error and Status Register (ESR)	S/U	Yes	Yes

Table 28-4. Module Memory Map (Continued)

Address	Use	Access Type	Affected by Hard Reset	Affected by Soft Reset
Base + $0024	Interrupt Masks 2 (IMASK2)	S/U	Yes	Yes
Base + $0028	Interrupt Masks 1 (IMASK1)	S/U	Yes	Yes
Base + $002C	Interrupt Flags 2 (IFLAG2)	S/U	Yes	Yes
Base + $0030	Interrupt Flags 1 (IFLAG1)	S/U	Yes	Yes
Base + $0034–$005F	Reserved			
Base + $0060–$007F	Reserved			
Base + $0080–$017F	Message Buffers MB0–MB15	S/U	No	No
Base + $0180–$027F	Message Buffers MB16–MB31	S/U	No	No
Base + $0280–$047F	Reserved			
Base + $0480-087F	Reserved			
Base + $0880-$08BF	Rx Individual Mask Registers RXIMR0-RXIMR15	S/U	No	No
Base + $08C0-$08FF	Rx Individual Mask Registers RXIMR16-RXIMR31	S/U	No	No
Base + $0900-$097F	Reserved			

The FlexCAN module stores CAN messages for transmission and reception using a Message Buffer structure. Each individual MB is formed by 16 bytes mapped on memory as described in Table 28-5. Table 28-5 shows a Standard/Extended Message Buffer (MB0) memory map, using 16 bytes total ($80–$8F space).

Table 28-5. Message Buffer MB0 Memory Mapping

Address Offset	MB Field
$80	Control and Status (C/S)
$84	Identifier Field
$88–$8F	Data Field 0 – Data Field 7 (1 byte each)

28.2.6.2 Message Buffer Structure

The Message Buffer structure used by the FlexCAN module is represented in Figure 28-2. Both Extended and Standard Frames (29-bit Identifier and 11-bit Identifier, respectively) used in the CAN specification (Version 2.0 Part B) are represented.

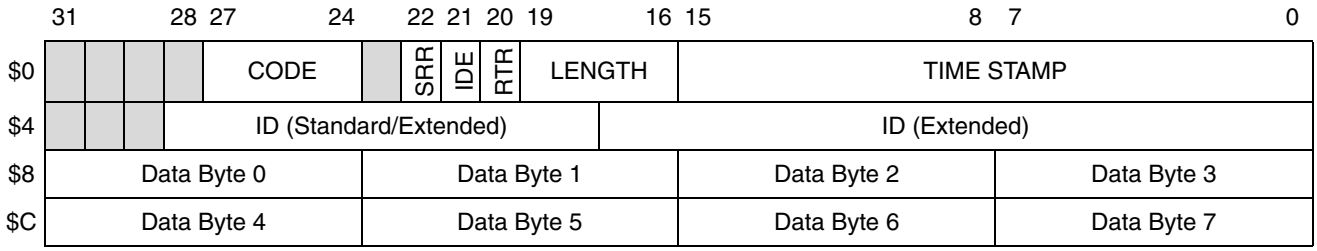


Figure 28-2. Message Buffer Structure

Table 28-6. Message Buffer Field Descriptions

Field	Description
CODE	Message Buffer Code. This 4-bit field can be accessed (read or write) by the CPU and by the Flexcan module itself, as part of the message buffer matching and arbitration process. The encoding is shown in Table 28-7 and Table 28-8. See Section 28.2.7, “Functional Description,” for additional information.
SRR	Substitute Remote Request. Fixed recessive bit, used only in extended format. It must be set to ‘1’ by the user for transmission (Tx Buffers) and will be stored with the value received on the CAN bus for Rx receiving buffers. It can be received as either recessive or dominant. If FlexCAN receives this bit as dominant, then it is interpreted as arbitration loss. 0 Dominant is not a valid value for transmission in Extended Format frames 1 Recessive value is compulsory for transmission in Extended Format frames
IDE	ID Extended Bit. This bit identifies whether the frame format is standard or extended. 0 Frame format is standard 1 Frame format is extended
RTR	Remote Transmission Request. This bit is used for requesting transmissions of a data frame. If FlexCAN transmits this bit as ‘1’ (recessive) and receives it as ‘0’ (dominant), it is interpreted as arbitration loss. If this bit is transmitted as ‘0’ (dominant), then if it is received as ‘1’ (recessive), the FlexCAN module treats it as bit error. If the value received matches the value transmitted, it is considered as a successful bit transmission. 0 Indicates the current MB has a Data Frame to be transmitted 1 Indicates the current MB has a Remote Frame to be transmitted
LENGTH	Length of Data in Bytes. This 4-bit field is the length (in bytes) of the Rx or Tx data, which is located in offset $8 through $F of the MB space (see Figure 28-2). In reception, this field is written by the FlexCAN module, copied from the DLC (Data Length Code) field of the received frame. In transmission, this field is written by the CPU and corresponds to the DLC field value of the frame to be transmitted. When RTR=1, the Frame to be transmitted is a Remote Frame and does not include the data field, regardless of the Length field.
TIME STAMP	Free-Running Counter Time Stamp. This 16-bit field is a copy of the Free-Running Timer, captured for Tx and Rx frames at the time when the beginning of the Identifier field appears on the CAN bus.
ID	Frame Identifier. In Standard Frame format, only the 11 most significant bits (28 to 18) are used for frame identification in both receive and transmit cases. The 18 least significant bits are ignored. In Extended Frame format, all bits are used for frame identification in both receive and transmit cases.
DATA	Data Field. Up to eight bytes can be used for a data frame. For Rx frames, the data is stored as it is received from the CAN bus. For Tx frames, the CPU prepares the data field to be transmitted within the frame.

Table 28-7. Message Buffer Code for Rx buffers

Rx Code BEFORE Rx New Frame	Description	Rx Code AFTER Rx New Frame	Comment
0000	INACTIVE: MB is not active.	–	MB does not participate in the matching process.
0100	EMPTY: MB is active and empty.	0010	MB participates in the matching process. When a frame is received successfully, the code is automatically updated to FULL.
0010	FULL: MB is full.	0010	The act of reading the C/S word followed by unlocking the MB does not make the code return to EMPTY. It remains FULL. If a new frame is written to the MB after the C/S word was read and the MB was unlocked, the code still remains FULL.
		0110	If the MB is FULL and a new frame is overwritten to this MB before the CPU had time to read it, the code is automatically updated to OVERRUN. Refer to Section 28.2.7.5, “Matching Process,” for details about overrun behavior.
0110	OVERRUN: a frame was overwritten into a full buffer.	0010	If the code indicates OVERRUN but the CPU reads the C/S word and then unlocks the MB, when a new frame is written to the MB the code returns to FULL.
		0110	If the code already indicates OVERRUN, and yet another new frame must be written, the MB will be overwritten again, and the code will remain OVERRUN. Refer to Section 28.2.7.5, “Matching Process,” for details about overrun behavior.
0XY1[1]	BUSY: Flexcan is updating the contents of the MB. The CPU must not access the MB.	0010	An EMPTY buffer was written with a new frame (XY was 01).
		0110	A FULL/OVERRUN buffer was overwritten (XY was 11).

1. Note that for Tx MBs (see Table 28-8), the BUSY bit should be ignored upon read.

Table 28-8. Message Buffer Code for Tx buffers

RTR	Initial Tx code	Code after successful transmission	Description
X	1000	–	INACTIVE: MB does not participate in the arbitration process.
0	1100	1000	Transmit data frame unconditionally once. After transmission, the MB automatically returns to the INACTIVE state.
1	1100	0100	Transmit remote frame unconditionally once. After transmission, the MB automatically becomes an Rx MB with the same ID.

Table 28-8. Message Buffer Code for Tx buffers

RTR	Initial Tx code	Code after successful transmission	Description
0	1010	1010	Transmit a data frame whenever a remote request frame with the same ID is received. This MB participates simultaneously in both the matching and arbitration processes. The matching process compares the ID of the incoming remote request frame with the ID of the MB. If a match occurs this MB is allowed to participate in the current arbitration process and the Code field is automatically updated to '1110' to allow the MB to participate in future arbitration runs. When the frame is eventually transmitted successfully, the Code automatically returns to '1010' to restart the process again.
0	1110	1010	This is an intermediate code that is automatically written to the MB by the MBM as a result of match to a remote request frame. The data frame will be transmitted unconditionally once and then the code will automatically return to '1010'. The CPU can also write this code with the same effect.

28.2.6.3 Register Descriptions

The FlexCAN registers are described in this section in ascending address order.

28.2.6.3.1 Module Configuration Register (MCR)

This register defines global system configurations, such as the module operation mode (e.g., low power) and maximum message buffer configuration. Most of the fields in this register can be accessed at any time, except the MAXMB field, which should only be changed while the module is in Freeze Mode.

Address: Base + $0000

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	MDIS	FRZ	0	HALT	NOT_RDY	WAK_MSK	SOFT_RST	FRZ_ACK	SUPV	SLF_WAK	WRN_EN	LPM_ACK	WAK_SRC	DOZE	SRX_DIS	BCC
W																
Reset	–[1]	1	0	1	1	0	0	–[2]	1	0	0	–[3]	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	MAXMB					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

1 Reset value of this bit is different on various platforms. Consult the specific MCU documentation to determine its value.

2 Different on various platforms, but it is always the opposite of the MDIS reset value.

3 Different on various platforms, but it is always the same as the MDIS reset value.

Figure 28-3. Module Configuration Register (MCR)

Table 28-9. MCR Field Descriptions

Field	Description
31 MDIS	Module Disable. This bit controls whether FlexCAN is enabled or not. When disabled, FlexCAN shuts down the clocks to the CAN Protocol Interface and Message Buffer Management sub-modules. This is the only bit in MCR not affected by soft reset. See Section 28.2.7.8.2, “Module Disable Mode,” for more information. 0 Enable the FlexCAN module 1 Disable the FlexCAN module
30 FRZ	Freeze Enable. The FRZ bit specifies the FlexCAN behavior when the HALT bit in the MCR Register is set or when Debug Mode is requested at MCU level. When FRZ is asserted, FlexCAN is enabled to enter Freeze Mode. Negation of this bit field causes FlexCAN to exit from Freeze Mode. 0 Not enabled to enter Freeze Mode 1 Enabled to enter Freeze Mode
29	Reserved, should be cleared.
28 HALT	Halt FlexCAN. Assertion of this bit puts the FlexCAN module into Freeze Mode. The CPU should clear it after initializing the Message Buffers and Control Register. No reception or transmission is performed by FlexCAN before this bit is cleared. While in Freeze Mode, the CPU has write access to the Error Counter Register, that is otherwise read-only. Freeze Mode can not be entered while FlexCAN is in any of the low power modes. See Section 28.2.7.8.1, “Freeze Mode,” for more information. 0 No Freeze Mode request. 1 Enters Freeze Mode if the FRZ bit is asserted.
27 NOT_RDY	FlexCAN Not Ready. This read-only bit indicates that FlexCAN is either in Disable Mode, Doze Mode, Stop Mode or Freeze Mode. It is negated once FlexCAN has exited these modes. 0 FlexCAN module is either in Normal Mode, Listen-Only Mode or Loop-Back Mode 1 FlexCAN module is either in Disable Mode, Doze Mode, Stop Mode or Freeze Mode
26 WAK_MSK	Wake Up Interrupt Mask. This bit enables the Wake Up Interrupt generation. 0 Wake Up Interrupt is disabled 1 Wake Up Interrupt is enabled
25 SOFT_RST	Soft Reset. When this bit is asserted, FlexCAN resets its internal state machines and some of the memory mapped registers. The following registers are reset: MCR (except the MDIS bit), TIMER, ECR, ESR, IMASK1, IMASK2, IFLAG1, IFLAG2. Configuration registers that control the interface to the CAN bus are not affected by soft reset. The following registers are unaffected: • CTRL • RXIMR0–RXIMR63 • RXGMASK, RX14MASK, RX15MASK • all Message Buffers The SOFT_RST bit can be asserted directly by the CPU when it writes to the MCR Register, but it is also asserted when global soft reset is requested at MCU level. Since soft reset is synchronous and has to follow a request/acknowledge procedure across clock domains, it may take some time to fully propagate its effect. The SOFT_RST bit remains asserted while reset is pending, and is automatically negated when reset completes. Therefore, software can poll this bit to know when the soft reset has completed. Soft reset cannot be applied while clocks are shut down in any of the low power modes. The module should be first removed from low power mode, and then soft reset can be applied. 0 No reset request 1 Resets the registers marked as “affected by soft reset” in Table 28-4

Table 28-9. MCR Field Descriptions (Continued)

Field	Description
24 FRZ_ACK	Freeze Mode Acknowledge. This read-only bit indicates that FlexCAN is in Freeze Mode and its prescaler is stopped. The Freeze Mode request cannot be granted until current transmission or reception processes have finished. Therefore the software can poll the FRZ_ACK bit to know when FlexCAN has actually entered Freeze Mode. If Freeze Mode request is negated, then this bit is negated once the FlexCAN prescaler is running again. If Freeze Mode is requested while FlexCAN is in any of the low power modes, then the FRZ_ACK bit will only be set when the low power mode is exited. See Section 28.2.7.8.1, “Freeze Mode,” for more information. 0 FlexCAN not in Freeze Mode, prescaler running 1 FlexCAN in Freeze Mode, prescaler stopped
23 SUPV	Supervisor Mode. This bit configures some of the FlexCAN registers to be either in Supervisor or Unrestricted memory space. The registers affected by this bit are marked as S/U in the Access Type column of Table 28-4. Reset value of this bit is ‘1’, so the affected registers start with Supervisor access restrictions. 0 Affected registers are in Unrestricted memory space 1 Affected registers are in Supervisor memory space. Any access without supervisor permission behaves as though the access was done to an unimplemented register location
22 SLF_WAK	Self Wake Up. This bit enables the Self Wake Up feature when FlexCAN is in Doze Mode or Stop Mode. If this bit had been asserted by the time FlexCAN entered Doze Mode or Stop Mode, then FlexCAN will look for a recessive to dominant transition on the bus during these modes. If a transition from recessive to dominant is detected during Doze Mode, FlexCAN resumes its clocks and, if enabled to do so, generates a Wake Up interrupt to the CPU. If a transition from recessive to dominant is detected during Stop Mode, then FlexCAN generates, if enabled to do so, a Wake Up interrupt to the CPU so that it can resume the clocks globally. This bit can not be written while the module is in Doze Mode or Stop Mode. 0 FlexCAN Self Wake Up feature is disabled 1 FlexCAN Self Wake Up feature is enabled
21 WRN_EN	Warning Interrupt Enable. When asserted, this bit enables the generation of the TWRN_INT and RWRN_INT flags in the Error and Status Register. If WRN_EN is negated, the TWRN_INT and RWRN_INT flags will always be zero, independent of the values of the error counters, and no warning interrupt will ever be generated. 0 TWRN_INT and RWRN_INT bits are zero, independent of the values in the error counters.. 1 TWRN_INT and RWRN_INT bits are set when the respective error counter transition from <96 to ≥ 96.
20 LPM_ACK	Low Power Mode Acknowledge. This read-only bit indicates that FlexCAN is either in Disable Mode, Doze Mode or Stop Mode. Either of these low power modes can not be entered until all current transmission or reception processes have finished, so the CPU can poll the LPM_ACK bit to know when FlexCAN has actually entered low power mode. See Section 28.2.7.8.2, “Module Disable Mode,” Section 28.2.7.8.3, “Doze Mode,” and Section 28.2.7.8.4, “Stop Mode,” for more information. 0 FlexCAN not in any of the low power modes 1 FlexCAN is either in Disable Mode, Doze Mode or Stop mode
19 WAK_SRC	Wake Up Source. This bit defines whether the integrated low-pass filter is applied to protect the Rx CAN input from spurious wake up. See Section 28.2.7.8.3, “Doze Mode,” and Section 28.2.7.8.4, “Stop Mode,” for more information. 0 FlexCAN uses the unfiltered Rx input to detect recessive to dominant edges on the CAN bus. 1 FlexCAN uses the filtered Rx input to detect recessive to dominant edges on the CAN bus **Note**: The integrated low-pass filter may not be available in all MCUs. In case it is not available, the unfiltered input is always used for wake up purposes, and this bit has no effect on the FlexCAN operation.

Table 28-9. MCR Field Descriptions (Continued)

Field	Description
18 DOZE	Doze Mode Enable. This bit defines whether FlexCAN is allowed to enter low power mode when Doze Mode is requested at MCU level. This bit is automatically reset when FlexCAN wakes up from Doze Mode upon detecting activity on the CAN bus (self wake-up enabled). 0 FlexCAN is not enabled to enter low power mode when Doze Mode is requested 1 FlexCAN is enabled to enter low power mode when Doze Mode is requested
17 SRX_DIS	Self Reception Disable. This bit defines whether FlexCAN is allowed to receive frames transmitted by itself. If this bit is asserted, frames transmitted by the module will not be stored in any MB, regardless if the MB is programmed with an ID that matches the transmitted frame, and no interrupt flag or interrupt signal will be generated due to the frame reception. 0 Self reception enabled 1 Self reception disabled
16 BCC	Backwards Compatibility Configuration. This bit is provided to support Backwards Compatibility with previous FlexCAN versions. When this bit is negated, the following configuration is applied: • For MCUs supporting individual Rx ID masking, this feature is disabled. Instead of individual ID masking per MB, FlexCAN uses its previous masking scheme with RXGMASK, RX14MASK and RX15MASK. • The reception queue feature is disabled. Upon receiving a message, if the first MB with a matching ID that is found is still occupied by a previous unread message, FlexCAN will not look for another matching MB. It will override this MB with the new message and set the CODE field to ‘0110’ (overrun). • Generation of the WRN_INT flag and the corresponding interrupt request are disabled. Upon reset this bit is negated, allowing legacy software to work without modification. 0 Individual Rx masking, queue feature and WRN_INT generation are disabled. 1 Individual Rx masking, queue feature and WRN_INT generation are enabled.
15–6	Reserved, should be cleared.
5–0 MAXMB	Maximum Number of Message Buffers. This 6-bit field defines the maximum number of message buffers that will take part in the matching and arbitration processes. The reset value ($0F) is equivalent to 16 MB configuration. This field should be changed only while the module is in Freeze Mode. Maximum MBs in use = MAXMB + 1. **Note**: MAXMB has to be programmed with a value smaller or equal to the number of available Message Buffers, otherwise FlexCAN will not transmit or receive frames.

28.2.6.3.2 Control Register (CTRL)

This register is defined for specific FlexCAN control features related to the CAN bus, such as bit-rate, programmable sampling point within an Rx bit, Loop Back Mode, Listen Only Mode, Bus Off recovery behavior and interrupt enabling (Bus-Off, Error, Warning). It also determines the Division Factor for the clock prescaler. Most of the fields in this register should only be changed while the module is in Disable Mode or in Freeze Mode. Exceptions are the BOFF_MSK, ERR_MSK, WRN_MSK and BOFF_REC bits, that can be accessed at any time.

Address: Base + $0004

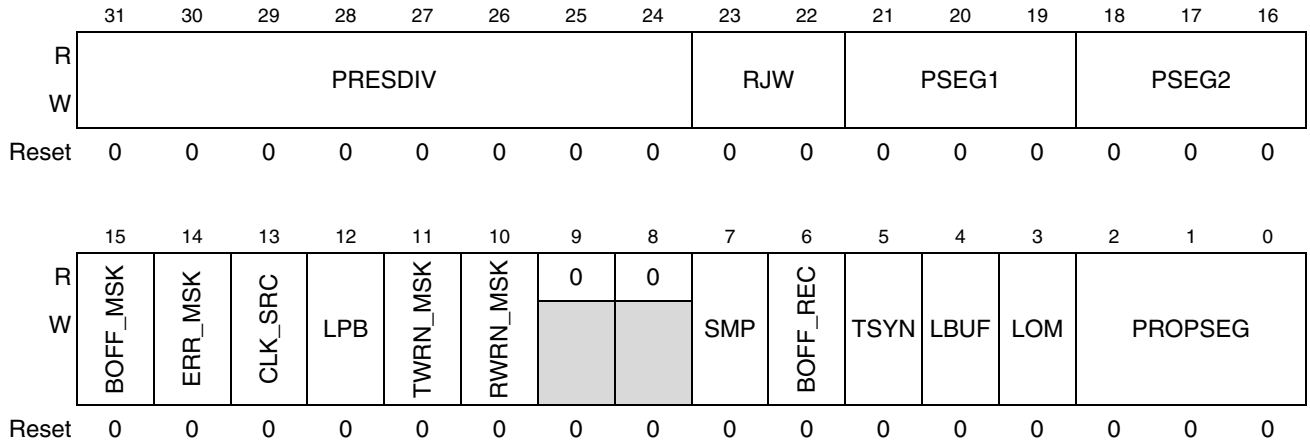


Figure 28-4. Control Register (CTRL)

Table 28-10. CTRL Field Descriptions

Field	Description
31–24 PRESDIV	Prescaler Division Factor. This 8-bit field defines the ratio between the CPI clock frequency and the Serial Clock (Sclock) frequency. The Sclock period defines the time quantum of the CAN protocol. For the reset value, the Sclock frequency is equal to the CPI clock frequency. The Maximum value of this register is $FF, that gives a minimum Sclock frequency equal to the CPI clock frequency divided by 256. For more information refer to Section 28.2.7.7.4, “Protocol Timing.” Sclock frequency = CPI clock frequency / (PRESDIV + 1)
23–22 RJW	Resync Jump Width. This 2-bit field defines the maximum number of time quanta[1] that a bit time can be changed by one re-synchronization. The valid programmable values are 0–3. Resync Jump Width = RJW + 1.
21–19 PSEG1	Phase Segment 1. This 3-bit field defines the length of Phase Buffer Segment 1 in the bit time. The valid programmable values are 0–7. Phase Buffer Segment 1 = (PSEG1 + 1) x Time-Quanta.
18–16 PSEG2	Phase Segment 2. This 3-bit field defines the length of Phase Buffer Segment 2 in the bit time. The valid programmable values are 1–7. Phase Buffer Segment 2 = (PSEG2 + 1) x Time-Quanta.
15 BOFF_MSK	Bus Off Mask. This bit provides a mask for the Bus Off Interrupt. 0 Bus Off interrupt disabled 1 Bus Off interrupt enabled
14 ERR_MSK	Error Mask. This bit provides a mask for the Error Interrupt. 0 Error interrupt disabled 1 Error interrupt enabled

Table 28-10. CTRL Field Descriptions (Continued)

Field	Description
13 CLK_SRC	CAN Engine Clock Source. This bit selects the clock source to the CAN Protocol Interface (CPI) to be either the peripheral clock (driven by the PLL) or the crystal oscillator clock. The selected clock is the one fed to the prescaler to generate the Serial Clock (Sclock). In order to guarantee reliable operation, this bit should only be changed while the module is in Disable Mode. See Section 28.2.7.7.4, "Protocol Timing," for more information. 0 The CAN engine clock source is the oscillator clock1The CAN engine clock source is the bus clock **Note**: This clock selection feature may not be available in all MCUs. A particular MCU may not have a PLL, in which case it would have only the oscillator clock, or it may use only the PLL clock feeding the FlexCAN module. In these cases, this bit has no effect on the module operation.
12 LPB	Loop Back. This bit configures FlexCAN to operate in Loop-Back Mode. In this mode, FlexCAN performs an internal loop back that can be used for self test operation. The bit stream output of the transmitter is fed back internally to the receiver input. The Rx CAN input pin is ignored and the Tx CAN output goes to the recessive state (logic '1'). FlexCAN behaves as it normally does when transmitting, and treats its own transmitted message as a message received from a remote node. In this mode, FlexCAN ignores the bit sent during the ACK slot in the CAN frame acknowledge field, generating an internal acknowledge bit to ensure proper reception of its own message. Both transmit and receive interrupts are generated. 0 Loop Back disabled 1 Loop Back enabled
11 TWRN_MSK	Tx Warning Interrupt Mask. This bit provides a mask for the Tx Warning Interrupt associated with the TWRN_INT flag in the Error and Status Register. This bit has no effect if the WRN_EN bit in MCR is negated, and it is read as zero when WRN_EN is negated. 0 Tx Warning Interrupt disabled 1 Warning Interrupt enabled
10 RWRN_MSK	Rx Warning Interrupt Mask. This bit provides a mask for the Rx Warning Interrupt associated with the RWRN_INT flag in the Error and Status Register. This bit has no effect if the WRN_EN bit in MCR is negated, and it is read as zero when WRN_EN is negated. 0 Rx Warning Interrupt disabled 1 Rx Warning Interrupt enabled
9–8	Reserved, should be cleared.
7 SMP	Sampling Mode. This bit defines the sampling mode of CAN bits at the Rx input. 0 Just one sample is used to determine the bit value 1 Three samples are used to determine the value of the received bit: the regular one (sample point) and 2 preceding samples, a majority rule is used
6 BOFF_REC	Bus Off Recovery Mode. This bit defines how FlexCAN recovers from Bus Off state. If this bit is negated, automatic recovering from Bus Off state occurs according to the CAN Specification 2.0B. If the bit is asserted, automatic recovering from Bus Off is disabled and the module remains in Bus Off state until the bit is negated by the user. If the negation occurs before 128 sequences of 11 recessive bits are detected on the CAN bus, then Bus Off recovery happens as if the BOFF_REC bit had never been asserted. If the negation occurs after 128 sequences of 11 recessive bits occurred, then FlexCAN will re-synchronize to the bus by waiting for 11 recessive bits before joining the bus. After negation, the BOFF_REC bit can be re-asserted again during Bus Off, but it will only be effective the next time the module enters Bus Off. If BOFF_REC was negated when the module entered Bus Off, asserting it during Bus Off will not be effective for the current Bus Off recovery. 0 Automatic recovering from Bus Off state enabled, according to CAN Spec 2.0 part B 1 Automatic recovering from Bus Off state disabled

Table 28-10. CTRL Field Descriptions (Continued)

Field	Description
5 TSYN	Timer Sync Mode. This bit enables a mechanism that resets the free-running timer each time a message is received in Message Buffer 0. This feature provides means to synchronize multiple FlexCAN stations with a special “SYNC” message (i.e., global network time). 0 Timer Sync feature disabled 1 Timer Sync feature enabled
4 LBUF	Lowest Buffer Transmitted First . This bit defines the ordering mechanism for Message Buffer transmission. 0 Buffer with lowest ID is transmitted first 1 Lowest number buffer is transmitted first
3 LOM	Listen-Only Mode. This bit configures FlexCAN to operate in Listen Only Mode. In this mode, transmission is disabled, all error counters are frozen and the module operates in a CAN Error Passive mode [Ref. 1]. Only messages acknowledged by another CAN station will be received. If FlexCAN detects a message that has not been acknowledged, it will flag a BIT0 error (without changing the REC), as if it was trying to acknowledge the message. 0 Listen Only Mode is deactivated 1 FlexCAN module operates in Listen Only Mode
2–0 PROPSEG	Propagation Segment. This 3-bit field defines the length of the Propagation Segment in the bit time. The valid programmable values are 0–7. Propagation Segment Time = (PROPSEG + 1) * Time-Quanta. Time-Quantum = one Sclock period.

1. One time quantum is equal to the Sclock period.

28.2.6.3.3 Free Running Timer (TIMER)

This register represents a 16-bit free running counter that can be read and written by the CPU. The timer starts from $0000 after Reset, counts linearly to $FFFF, and wraps around.

The timer is clocked by the FlexCAN bit-clock (which defines the baud rate on the CAN bus). During a message transmission/reception, it increments by one for each bit that is received or transmitted. When there is no message on the bus, it counts using the previously programmed baud rate. During Freeze Mode, the timer is not incremented.

The timer value is captured at the beginning of the identifier field of any frame on the CAN bus. This captured value is written into the Time Stamp entry in a message buffer after a successful reception or transmission of a message.

Writing to the timer is an indirect operation. The data is first written to an auxiliary register and then an internal request/acknowledge procedure across clock domains is executed. All this is transparent to the user, except for the fact that the data will take some time to be actually written to the register. If desired, software can poll the register to discover when the data was actually written.

Address: Base + $0008

	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																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	TIMER															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-5. Free Running Timer (TIMER)

28.2.6.3.4 Rx Global Mask (RXGMASK)

This register is provided for legacy support and for low cost MCUs that do not have the individual masking per Message Buffer feature. See Section 28.2.6.3.11, “Rx Individual Mask Registers (RXIMR0–RXIMR63).” For MCUs supporting individual masks per MB, setting the BCC bit in MCR causes the RXGMASK Register to have no effect on the module operation. For MCUs not supporting individual masks per MB, this register is always effective.

RXGMASK is used as acceptance mask for all received MBs, excluding MBs 14–15, which have their specific Rx mask registers. The meaning of each mask bit is the following:

- Mask bit = 0: the corresponding incoming ID bit is “don’t care”
- Mask bit = 1: the corresponding ID bit is checked against the incoming ID bit, to see if a match exists

The contents of this register must be programmed while the module is in Freeze Mode, and must not be modified when the module is transmitting or receiving frames.

Address: Base + $0010

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	MI28	MI27	MI26	MI25	MI24	MI23	MI22	MI21	MI20	MI19	MI18	MI17	MI16
W																
Reset	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	MI15	MI14	MI13	MI12	MI11	MI10	MI9	MI8	MI7	MI6	MI5	MI4	MI3	MI2	MI1	MI0
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 28-6. Rx Global Mask Register (RXGMASK)

Table 28-11. RXGMASK Field Descriptions

Field	Description
31–29	Reserved, should be cleared.
28–18 MI28–MI18	Standard ID Mask Bits. These bits are the same mask bits for the Standard and Extended Formats.
17–0 MI17–MI0	Extended ID Mask Bits. These bits are used to mask comparison only in Extended Format.

28.2.6.3.5 Rx 14 Mask (RX14MASK)

This register is provided for legacy support and for low cost MCUs that do not have the individual masking per Message Buffer feature. See Section 28.2.6.3.11, “Rx Individual Mask Registers (RXIMR0–RXIMR63).” For MCUs supporting individual masks per MB, setting the BCC bit in MCR causes the RX14MASK Register to have no effect on the module operation.

RX14MASK is used as acceptance mask for the Identifier in Message Buffer 14. This register has the same structure as the Rx Global Mask Register. It must be programmed while the module is in Freeze Mode, and must not be modified when the module is transmitting or receiving frames.

- Address Offset: $14
- Reset Value: $1FFF_FFFF

28.2.6.3.6 Rx 15 Mask (RX15MASK)

This register is provided for legacy support and for low cost MCUs that do not have the individual masking per Message Buffer feature. See Section 28.2.6.3.11, “Rx Individual Mask Registers (RXIMR0–RXIMR63).” For MCUs supporting individual masks per MB, setting the BCC bit in MCR causes the RX15MASK Register to have no effect on the module operation.

When the BCC bit is negated, RX15MASK is used as acceptance mask for the Identifier in Message Buffer 15. This register has the same structure as the Rx Global Mask Register. It must be programmed while the module is in Freeze Mode, and must not be modified when the module is transmitting or receiving frames.

- Address Offset: $18
- Reset Value: $1FFF_FFFF

28.2.6.3.7 Error Counter Register (ECR)

This register has 2 8-bit fields reflecting the value of two FlexCAN error counters: Transmit Error Counter (Tx_Err_Counter field) and Receive Error Counter (Rx_Err_Counter field). The rules for increasing and decreasing these counters are described in the CAN protocol and are completely implemented in the FlexCAN module. Both counters are read only except in Freeze Mode, where they can be written by the CPU.

Writing to the Error Counter Register while in Freeze Mode is an indirect operation. The data is first written to an auxiliary register and then an internal request/acknowledge procedure across clock domains is executed. All this is transparent to the user, except for the fact that the data will take some time to be

actually written to the register. If desired, software can poll the register to discover when the data was actually written.

FlexCAN responds to any bus state as described in the protocol, e.g. transmit ‘Error Active’ or ‘Error Passive’ flag, delay its transmission start time (‘Error Passive’) and avoid any influence on the bus when in ‘Bus Off’ state. The following are the basic rules for FlexCAN bus state transitions.

- If the value of Tx_Err_Counter or Rx_Err_Counter increases to be greater than or equal to 128, the FLT_CONF field in the Error and Status Register is updated to reflect ‘Error Passive’ state.
- If the FlexCAN state is ‘Error Passive’, and either Tx_Err_Counter or Rx_Err_Counter decrements to a value less than or equal to 127 while the other already satisfies this condition, the FLT_CONF field in the Error and Status Register is updated to reflect ‘Error Active’ state.
- If the value of Tx_Err_Counter increases to be greater than 255, the FLT_CONF field in the Error and Status Register is updated to reflect ‘Bus Off’ state, and an interrupt may be issued. The value of Tx_Err_Counter is then reset to zero.
- If FlexCAN is in ‘Bus Off’ state, then Tx_Err_Counter is cascaded together with another internal counter to count the 128th occurrences of 11 consecutive recessive bits on the bus. Hence, Tx_Err_Counter is reset to zero and counts in a manner where the internal counter counts 11 such bits and then wraps around while incrementing the Tx_Err_Counter. When Tx_Err_Counter reaches the value of 128, the FLT_CONF field in the Error and Status Register is updated to be ‘Error Active’ and both error counters are reset to zero. At any instance of dominant bit following a stream of less than 11 consecutive recessive bits, the internal counter resets itself to zero without affecting the Tx_Err_Counter value.
- If during system start-up, only one node is operating, then its Tx_Err_Counter increases in each message it is trying to transmit, as a result of acknowledge errors (indicated by the ACK_ERR bit in the Error and Status Register). After the transition to ‘Error Passive’ state, the Tx_Err_Counter does not increment anymore by acknowledge errors. Therefore the device never goes to the ‘Bus Off’ state.
- If the Rx_Err_Counter increases to a value greater than 127, it is not incremented further, even if more errors are detected while being a receiver. At the next successful message reception, the counter is set to a value between 119 and 127 to resume to ‘Error Active’ state.

Address: Base + $001C

	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																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Rx_Err_Counter								Tx_Err_Counter							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-7. Error Counter Register (ECR)

28.2.6.3.8 Error and Status Register (ESR)

This register reflects various error conditions, some general status of the device and it is the source of four interrupts to the CPU. The reported error conditions (bits 15–10) are those that occurred since the last time the CPU read this register. The CPU read action clears bits 15–10. Bits 9–4 are status bits.

Most bits in this register are read only, except TWRN_INT, RWRN_INT, BOFF_INT, WAK_INT and ERR_INT, that are interrupt flags that can be cleared by writing ‘1’ to them (writing ‘0’ has no effect). See Section 28.2.7.9, “Interrupts,” for more details.

Address: Base + $0020

	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	TWRN_INT	RWRN_INT
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	BIT1_ERR	BIT0_ERR	ACK_ERR	CRC_ERR	FRM_ERR	STF_ERR	TX_WRN	RX_WRN	IDLE	TXRX	FLT_CONF		0	BOFF_INT	ERR_INT	WAK_INT
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-8. Error and Status Register (ESR)

Table 28-12. ESR Field Descriptions

Field	Description
31–18	Reserved, should be cleared.
17 TWRN_INT	Tx Warning Interrupt Flag. If the WRN_EN bit in MCR is asserted, the TWRN_INT bit is set when the TX_WRN flag transition from ‘0’ to ‘1’, meaning that the Tx error counter reached 96. If the corresponding mask bit in the Control Register (TWRN_MSK) is set, an interrupt is generated to the CPU. This bit is cleared by writing it to ‘1’. Writing ‘0’ has no effect. 0 No such occurrence 1 The Tx error counter transition from < 96 to ≥ 96
16 RWRN_INT	Rx Warning Interrupt Flag. If the WRN_EN bit in MCR is asserted, the RWRN_INT bit is set when the RX_WRN flag transition from ‘0’ to ‘1’, meaning that the Rx error counter reached 96. If the corresponding mask bit in the Control Register (RWRN_MSK) is set, an interrupt is generated to the CPU. This bit is cleared by writing it to ‘1’. Writing ‘0’ has no effect. 0 No such occurrence 1 The Rx error counter transition from < 96 to ≥ 96
15 BIT1_ERR	Bit1 Error. This bit indicates when an inconsistency occurs between the transmitted and the received bit in a message. 0 No such occurrence1At least one bit sent as recessive is received as dominant **Note**: This bit is not set by a transmitter in case of arbitration field or ACK slot, or in case of a node sending a passive error flag that detects dominant bits.

Table 28-12. ESR Field Descriptions (Continued)

Field	Description
14 BIT0_ERR	Bit0 Error. This bit indicates when an inconsistency occurs between the transmitted and the received bit in a message. 0 No such occurrence 1 At least one bit sent as dominant is received as recessive
13 ACK_ERR	Acknowledge Error. This bit indicates that an Acknowledge Error has been detected by the transmitter node, i.e., a dominant bit has not been detected during the ACK SLOT. 0 No such occurrence 1 An ACK error occurred since last read of this register
12 CRC_ERR	Cyclic Redundancy Check Error. This bit indicates that a CRC Error has been detected by the receiver node, i.e., the calculated CRC is different from the received. 0 No such occurrence 1 A CRC error occurred since last read of this register.
11 FRM_ERR	Form Error. This bit indicates that a Form Error has been detected by the receiver node, i.e., a fixed-form bit field contains at least one illegal bit.
10 STF_ERR	Stuffing Error. This bit indicates that a Stuffing Error has been detected. 0 No such occurrence. 1 A Stuffing Error occurred since last read of this register.
9 TX_WRN	TX Error Counter. This bit indicates that repetitive errors are occurring during message transmission. If the BCC bit in MCR is asserted, then when TX_WRN changes from 0 to 1, the WRN_INT flag is set and an interrupt request is generated if allowed by the WRN_MSK bit in CTRL. After the WRN_INT flag is set, no further updates will happen to TX_WRN until the WRN_INT flag is cleared by the CPU (by writing ‘1’ to it). 0 No such occurrence 1 TX_Err_Counter ≥ 96
8 RX_WRN	Rx Error Counter. This bit indicates when repetitive errors are occurring during message reception. If the BCC bit in MCR is asserted, then when RX_WRN changes from 0 to 1, the WRN_INT flag is set and an interrupt request is generated if allowed by the WRN_MSK bit in CTRL. After the WRN_INT flag is set, no further updates will happen to RX_WRN until the WRN_INT flag is cleared by the CPU (by writing ‘1’ to it). 0 No such occurrence 1 Rx_Err_Counter ≥ 96
7 IDLE	CAN bus IDLE state. This bit indicates when CAN bus is in IDLE state. 0 No such occurrence 1 CAN bus is now IDLE
6 TXRX	Current FlexCAN status (transmitting/receiving). This bit indicates if FlexCAN is transmitting or receiving a message when the CAN bus is not in IDLE state. This bit has no meaning when IDLE is asserted. 0 FlexCAN is receiving a message (IDLE=0) 1 FlexCAN is transmitting a message (IDLE=0)
5–4 FLT_CONF	Fault Confinement State. This 2-bit field indicates the Confinement State of the FlexCAN module, as shown in the values below. If the LOM bit in the Control Register is asserted, the FLT_CONF field will indicate “Error Passive”. Since the Control Register is not affected by soft reset, the FLT_CONF field will not be affected by soft reset if the LOM bit is asserted. 00 Error active 01 Error passive 1X Bus off
3	Reserved, should be cleared.

Table 28-12. ESR Field Descriptions (Continued)

Field	Description
2 BOFF_INT	'Bus Off' Interrupt. This bit is set when FlexCAN enters 'Bus Off' state. If the corresponding mask bit in the Control Register (BOFF_MSK) is set, an interrupt is generated to the CPU. This bit is cleared by writing it to '1'. Writing '0' has no effect. 0 No such occurrence 1 FlexCAN module entered 'Bus Off' state
1 ERR_INT	Error Interrupt. This bit indicates that at least one of the Error Bits (bits 15-10) is set. If the corresponding mask bit in the Control Register (ERR_MSK) is set, an interrupt is generated to the CPU. This bit is cleared by writing it to '1'.Writing '0' has no effect. 0 No such occurrence 1 Indicates setting of any Error Bit in the Error and Status Register
0 WAK_INT	Wake-Up Interrupt. When FlexCAN is in Doze Mode or Stop Mode and a recessive to dominant transition is detected on the CAN bus and if the WAK_MSK bit in the MCR Register is set, an interrupt is generated to the CPU. This bit is cleared by writing it to '1'. Writing '0' has no effect. 0 No such occurrence 1 Indicates a recessive to dominant transition received on the CAN bus when the FlexCAN module is in Doze Mode or Stop Mode

28.2.6.3.9 Interrupt Masks 1 Register (IMASK1)

This register allows to enable or disable any number of a range of 32 Message Buffer Interrupts. It contains one interrupt mask bit per buffer, enabling the CPU to determine which buffer generates an interrupt after a successful transmission or reception (i.e., when the corresponding IFLAG1 bit is set).

Address: Base + $0028

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	BUF 31M	BUF 30M	BUF 29M	BUF 28M	BUF 27M	BUF 26M	BUF 25M	BUF 24M	BUF 23M	BUF 22M	BUF 21M	BUF 20M	BUF 19M	BUF 18M	BUF 17M	BUF 16M
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	BUF 15M	BUF 14M	BUF 13M	BUF 12M	BUF 11M	BUF 10M	BUF 9M	BUF 8M	BUF 7M	BUF 6M	BUF 5M	BUF 4M	BUF 3M	BUF 2M	BUF 1M	BUF 0M
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-9. Interrupt Masks 1 Register (IMASK1)

Table 28-13. IMASK1 Field Descriptions

Field	Description
31–0 BUF31M– BUF0M	Buffer MB_i Mask. Each bit enables or disables the respective FlexCAN Message Buffer (MB0 to MB31) Interrupt. 0 The corresponding buffer Interrupt is disabled1The corresponding buffer Interrupt is enabled **Note**: Setting or clearing a bit in the IMASK1 Register can assert or negate an interrupt request, if the corresponding IFLAG1 bit is set.

28.2.6.3.10 Interrupt Flags 1 Register (IFLAG1)

This register defines the flags for 32 Message Buffer interrupts. It contains one interrupt flag bit per buffer. Each successful transmission or reception sets the corresponding IFLAG1 bit. If the corresponding IMASK1 bit is set, an interrupt will be generated. The Interrupt flag must be cleared by writing it to ‘1’. Writing ‘0’ has no effect.

Address: Base + $0030

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	BUF 31I	BUF 30I	BUF 29I	BUF 28I	BUF 27I	BUF 26I	BUF 25I	BUF 24I	BUF 23I	BUF 22I	BUF 21I	BUF 20I	BUF 19I	BUF 18I	BUF 17I	BUF 16I
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	BUF 15I	BUF 14I	BUF 13I	BUF 12I	BUF 11I	BUF 10I	BUF 9I	BUF 8I	BUF 7I	BUF 6I	BUF 5I	BUF 4I	BUF 3I	BUF 2I	BUF 1I	BUF 0I
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-10. Interrupt Flags 1 Register (IFLAG1)

Table 28-14. IFLAG1 Field Descriptions

Field	Description
31–0 BUF31I– BUF0I	Buffer MB_i Interrupt. Each bit flags the respective FlexCAN Message Buffer (MB0 to MB31) interrupt. 0 No such occurrence 1 The corresponding buffer has successfully completed transmission or reception

28.2.6.3.11 Rx Individual Mask Registers (RXIMR0–RXIMR63)

These registers are used as acceptance masks for received frame ID, in both Standard and Extended ID formats. One mask register is provided for each Message Buffer for individual ID masking per Message Buffer. The meaning of each mask bit is the following:

- Mask bit = 0: the corresponding incoming ID bit is “don’t care”
- Mask bit = 1: the corresponding ID bit is checked against the incoming ID bit, to see if a match exists

The Individual Rx Mask Registers are implemented in RAM, so they are not affected by reset and must be explicitly initialized prior to any reception. Furthermore, they can only be accessed by the CPU while the module is in Freeze Mode. Out of Freeze Mode, write accesses are blocked and read accesses will return “all zeros”. Furthermore, if the BCC bit in the MCR Register is negated, any read or write operation to these registers results in access error.

NOTE

The individual Rx Mask per Message Buffer feature may not be available in low cost MCUs. Please consult the specific MCU documentation to find out if this feature is supported. If not supported, the RXGMASK, RX14MASK and RX15MASK registers are available, regardless of the value of the BCC bit.

Address: Base + $0880–$097F

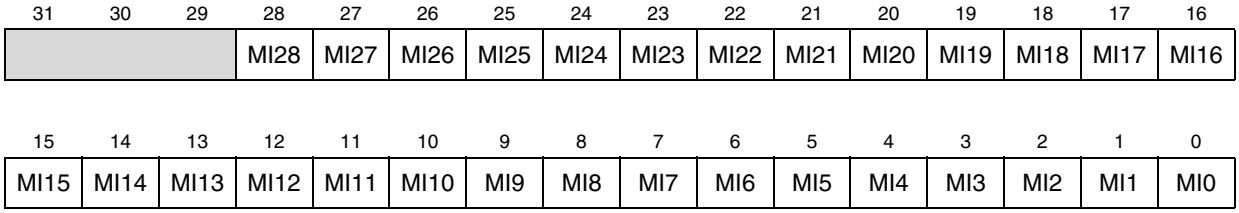


Figure 28-11. Rx Individual Mask Registers (RXIMR0–RXIMR63)

Table 28-15. RXIMR*n* Field Descriptions

Field	Description
31–29	Reserved.
28–18 MI28–MI18	Standard ID Mask Bits. These bits are the same mask bits for the Standard and Extended Formats.
17–0 MI17–MI0	Extended ID Mask Bits. These bits are used to mask comparison only in Extended Format.

28.2.7 Functional Description

28.2.7.1 Overview

The FlexCAN module is a CAN protocol engine with a very flexible mailbox system for transmitting and receiving CAN frames. The mailbox system is composed by a set of up to 64 Message Buffers (MB) that store configuration and control data, time stamp, message ID and data (see Section 28.2.6.2, “Message Buffer Structure”). Any MB can be work as a transmission or reception buffer. An arbitration algorithm decides the prioritization of MBs to be transmitted based on either the message ID or the MB ordering. A matching algorithm makes it possible to store received frames only into MBs that have the same ID programmed on its ID field. A masking scheme makes it possible to match the ID programmed on the MB with a range of IDs on received CAN frames. A reception queue can be implemented by programming the same ID on more than one receiving MB. Data coherency mechanisms are implemented to guarantee data integrity during MB manipulation by the CPU.

Before proceeding with the functional description, an important concept must be explained. A Message Buffer is said to be “active” at a given time if it can participate in the matching and arbitration algorithms that are happening at that time. An Rx MB with a ‘0000’ code is inactive (refer to Table 28-7). Similarly, a Tx MB with a ‘1000’ code is also inactive (refer to Table 28-8). An MB not programmed with either ‘0000’ or ‘1000’ will be temporarily deactivated (will not participate in the current arbitration/matching

run) when the CPU writes to the C/S field of that MB (see Section 28.2.7.6.1, “Message Buffer Deactivation”).

28.2.7.2 Transmit Process

In order to transmit a CAN frame, the CPU should prepare a Message Buffer for transmission by executing the following procedure:

- Write ‘1000’ to the Code field of the Control and Status word to keep the MB inactive
- Write the ID word
- Write the data bytes
- Write the Length, Control and Code fields of the Control and Status word to activate the MB

The first and last steps are mandatory. The first write to the Control and Status word is important in case there was pending reception or transmission. The write operation immediately deactivates the MB, removing it from any currently ongoing arbitration or ID matching processes, giving time for the CPU to program the rest of the MB (see Section 28.2.7.6.1, “Message Buffer Deactivation”). Once the MB is activated in the fourth step, it will participate into the arbitration process and eventually be transmitted according to its priority. At the end of the successful transmission, the value of the Free Running Timer is written into the Time Stamp field, the Code field in the Control and Status word is updated, a status flag is set in the Interrupt Flag Register and an interrupt is generated if allowed by the corresponding Interrupt Mask Register bit. The new Code field after transmission depends on the code that was used to activate the MB in step four (see Table 28-7 and Table 28-8 in Section Section 28.2.6.2, “Message Buffer Structure”).

28.2.7.3 Arbitration Process

The arbitration process is an algorithm executed by the MBM that scans the whole MB memory looking for the highest priority message to be transmitted. All MBs programmed as transmit buffers will be scanned to find the lowest ID[1] or the lowest MB number, depending on the LBUF bit on the Control Register. The arbitration process is triggered in the following events:

- During the CRC field of the CAN frame
- During the error delimiter field of the CAN frame
- During Intermission, if the winner MB defined in a previous arbitration was deactivated, or if there was no MB to transmit, but the CPU wrote to the C/S word of any MB after the previous arbitration finished
- When MBM is in Idle or Bus Off state and the CPU writes to the C/S word of any MB
- Upon leaving Freeze Mode

Once the highest priority MB is selected, it is transferred to a temporary storage space called Serial Message Buffer (SMB), which has the same structure as a normal MB but is not user accessible. This operation is called “move-out”. At the first opportunity window on the CAN bus, the message on the SMB

1. Actually, if LBUF is negated, the arbitration considers not only the ID, but also the RTR and IDE bits placed inside the ID at the same positions they are transmitted in the CAN frame.

is transmitted according to the CAN protocol rules. FlexCAN transmits up to eight data bytes, even if the DLC (Data Length Code) value is bigger.

28.2.7.4 Receive Process

To be able to receive CAN frames, the CPU must prepare one or more Message Buffers for reception by executing the following steps:

- Write ‘0000’ to the Code field of the Control and Status word to keep the MB inactive
- Write the ID word
- Write ‘0100’ to the Code field of the Control and Status word to activate the MB

The first and last steps are mandatory. The first write to the Control and Status word is important in case there was a pending reception or transmission. The write operation immediately deactivates the MB, removing it from any currently ongoing arbitration or matching process, giving time for the CPU to program the rest of the MB (see Section 28.2.7.6.1, “Message Buffer Deactivation”). Once the MB is activated in the third step, it will be able to receive CAN frames that match the programmed ID. At the end of a successful reception, the MB is updated by the MBM as follows:

- The value of the Free Running Timer is written into the Time Stamp field
- The received ID, Data (8 bytes at most) and Length fields are stored
- The Code field in the Control and Status word is updated (see Table 28-7 and Table 28-8 in Section Section 28.2.6.2, “Message Buffer Structure”)
- A status flag is set in the Interrupt Flag Register and an interrupt is generated if allowed by the corresponding Interrupt Mask Register bit

Upon receiving the MB interrupt, the CPU should service the received frame using the following procedure:

- Read the Control and Status word (mandatory – activates an internal lock for this buffer)
- Read the ID field (optional – needed only if a mask was used)
- Read the Data field
- Read the Free Running Timer (optional – releases the internal lock)

Upon reading the Control and Status word, if the BUSY bit is set in the Code field, then the CPU should defer the access to the MB until this bit is negated. Reading the Free Running Timer is not mandatory. If not executed the MB remains locked, unless the CPU reads the C/S word of another MB. Note that only a single MB is locked at a time. The only mandatory CPU read operation is the one on the Control and Status word to assure data coherency (see Section 28.2.7.6, “Data Coherence”).

The CPU should synchronize to frame reception by the status flag bit for the specific MB in one of the IFLAG Registers and not by the Code field of that MB. Polling the Code field does not work because once a frame was received and the CPU services the MB (by reading the C/S word followed by unlocking the MB), the Code field will not return to EMPTY. It will remain FULL, as explained in Table 28-7. If the CPU tries to workaround this behavior by writing to the C/S word to force an EMPTY code after reading the MB, the MB is actually deactivated from any currently ongoing matching process. As a result, a newly received frame matching the ID of that MB may be lost. In summary: *never do polling by reading directly the C/S word of the MBs. Instead, read the IFLAG registers.*

Note that the received ID field is always stored in the matching MB, thus the contents of the ID field in an MB may change if the match was due to masking. Note also that FlexCAN does receive frames transmitted by itself if there exists an Rx matching MB, provided the SRX_DIS bit in the MCR is not asserted. If SRX_DIS is asserted, FlexCAN will not store frames transmitted by itselt in any MB, even if it contains a matching MB, and no interrupt flag or interrupt signal will be generated due to the frame reception.

28.2.7.5 Matching Process

The matching process is an algorithm executed by the MBM that scans the whole MB memory looking for Rx MBs programmed with the same ID as the one received from the CAN bus. Only MBs programmed to receive will participate in the matching process for received frames.

While the ID, DLC and Data fields are retrieved from the CAN bus, they are stored temporarily in the Serial Message Buffer (SMB). The matching process takes place during the CRC field. If a matching ID is found in one of the MBs, the contents of the SMB will be transferred to the matched MB during the 6th bit of the End-Of-Frame field of the CAN protocol. This operation is called “move-in”. If any protocol error (CRC, ACK, etc.) is detected, than the move-in operation does not happen.

An MB with a matching ID is said to be “free to receive” a new frame if the following conditions are satisfied:

- The MB is not locked (see Section 28.2.7.6.2, “Message Buffer Lock Mechanism”)
- The Code field is either EMPTY or else it is FULL or OVERRUN but the CPU has already serviced the MB (read the C/S word and then unlocked the MB)

If the first MB with a matching ID is not “free to receive” the new frame, then the matching algorithm keeps looking for another free MB until it finds one. If it can not find one that is free, then it will overwrite the last matching MB (unless it is locked) and set the Code field to OVERRUN (refer to Table 28-7 and Table 28-8). If the last matching MB is locked, then the new message remains in the SMB, waiting for the MB to be unlocked (see Section 28.2.7.6.2, “Message Buffer Lock Mechanism”).

Suppose, for example, that there are two MBs with the same ID and FlexCAN starts receiving messages with that ID. Let us say that these MBs are the second and the fifth in the array. When the first message arrives, the matching algorithm will find the first match in MB number 2. The code of this MB is EMPTY, so the message is stored there. When the second message arrives, the matching algorithm will find MB number 2 again, but it is not “free to receive”, so it will keep looking and find MB number 5 and store the message there. If yet another message with the same ID arrives, the matching algorithm finds out that there are no matching MBs that are “free to receive”, so it decides to overwrite the last matched MB, which is number 5. In doing so, it sets the Code field of the MB to indicate OVERRUN.

The ability to match the same ID in more than one MB can be exploited to implement a reception queue to allow more time to the CPU for servicing the MBs. By programming more than one MB with the same ID, received messages will be queued into the MBs. The CPU can examine the Time Stamp field of the MBs to determine the order in which the messages arrived.

The matching algorithm described above can be changed to be the same one used in previous versions of the FlexCAN module. When the BCC bit in MCR is negated, the matching algorithm stops at the first MB with a matching ID that it founds, whether this MB is free or not. As a result, the message queueing feature does not work if the BCC bit is negated.

Matching to a range of IDs is possible by using ID Acceptance Masks. FlexCAN supports individual masking per MB. Refer to Section 28.2.6.3.11, “Rx Individual Mask Registers (RXIMR0–RXIMR63).” During the matching algorithm, if a mask bit is asserted, then the corresponding ID bit is compared. If the mask bit is negated, the corresponding ID bit is “don’t care”. Please note that the Individual Mask Registers are implemented in RAM, so they are not initialized out of reset. Also, they can only be programmed if the BCC bit is asserted and while the module is in Freeze Mode.

FlexCAN also supports an alternate masking scheme with only three mask registers (RGXMASK, RX14MASK and RX15MASK) for backwards compatibility. This alternate masking scheme is enabled when the BCC bit in the MCR Register is negated.

NOTE

The individual Rx Mask per Message Buffer feature may not be available in low cost MCUs. Please consult the specific MCU documentation to find out if this feature is supported. If not supported, the RXGMASK, RX14MASK and RX15MASK registers are available, regardless of the value of the BCC bit.

28.2.7.6 Data Coherence

In order to maintain data coherency and FlexCAN proper operation, the CPU must obey the rules described in Section 28.2.7.2, “Transmit Process,” and Section 28.2.7.4, “Receive Process.” Any form of CPU accessing an MB structure within FlexCAN other than those specified may cause FlexCAN to behave in an unpredictable way.

28.2.7.6.1 Message Buffer Deactivation

If the CPU wants to change the function of an active MB, the recommended procedure is to first put the module into Freeze Mode and then change the Code field of that MB. This is a safe procedure because FlexCAN waits for pending CAN bus and MB moving activities to finish before entering Freeze Mode. Nevertheless, a mechanism is provided to maintain data coherence when the CPU writes to the Control and Status word of active MBs out of Freeze Mode.

Any CPU write access to the Control and Status word of an MB causes that MB to be excluded from the transmit or receive processes during the current matching or arbitration round. This mechanism is called MB deactivation. It is temporary, affecting only for the current match/arbitration round.

The purpose of deactivation is data coherency. The match/arbitration process scans the MBs to decide which MB to transmit or receive. If the CPU updates the MB in the middle of a match or arbitration process, the data of that MB may no longer be coherent, therefore deactivation of that MB is done.

Even with the coherence mechanism described above, writing to the Control and Status word of active MBs when not in Freeze Mode may produce undesirable results. Examples are:

- Matching and arbitration are one-pass processes. If MBs are deactivated after they are scanned, no re-evaluation is done to determine a new match/winner. If an Rx MB with a matching ID is deactivated during the matching process after it was scanned, then this MB is marked as invalid to receive the frame, and FlexCAN will keep looking for another matching MB within the ones it has not scanned yet. If it can not find one, then the message will be lost. Suppose, for example, that two

MBs have a matching ID to a received frame, and the user deactivated the first matching MB after FlexCAN has scanned the second. The received frame will be lost even if the second matching MB was “free to receive”.

- If a Tx MB containing the lowest ID is deactivated after FlexCAN has scanned it, then FlexCAN will look for another winner within the MBs that it has not scanned yet. Therefore, it may transmit an MB with ID that may not be the lowest at the time because a lower ID might be present in one of the MBs that it had already scanned before the deactivation.
- There is a point in time until which the deactivation of a Tx MB causes it not to be transmitted (end of move-out). After this point, it is transmitted but no interrupt is issued and the Code field is not updated.

28.2.7.6.2 Message Buffer Lock Mechanism

Besides MB deactivation, FlexCAN has another data coherence mechanism for the receive process. When the CPU reads the Control and Status word of an “active not empty” Rx MB, FlexCAN assumes that the CPU wants to read the whole MB in an atomic operation, and thus it sets an internal lock flag for that MB. The lock is released when the CPU reads the Free Running Timer (global unlock operation), or when it reads the Control and Status word of another MB. The MB locking is done to prevent a new frame to be written into the MB while the CPU is reading it.

NOTE

The locking mechanism only applies to Rx MBs which have a code different than INACTIVE (‘0000’) or EMPTY[1] (‘0100’). Also, Tx MBs can not be locked.

Suppose, for example, that the second and the fifth MBs of the array are programmed with the same ID, and FlexCAN has already received and stored messages into these two MBs. Suppose now that the CPU decides to read MB number 5 and at the same time another message with the same ID is arriving. When the CPU reads the Control and Status word of MB number 5, this MB is locked. The new message arrives and the matching algorithm finds out that there are no “free to receive” MBs, so it decides to override MB number 5. However, this MB is locked, so the new message can not be written there. It will remain in the SMB waiting for the MB to be unlocked, and only then will be written to the MB. If the MB is not unlocked in time and yet another new message with the same ID arrives, then the new message overwrites the one on the SMB and there will be no indication of lost messages either in the Code field of the MB or in the Error and Status Register.

While the message is being moved-in from the SMB to the MB, the BUSY bit on the Code field is asserted. If the CPU reads the Control and Status word and finds out that the BUSY bit is set, it should defer accessing the MB until the BUSY bit is negated.

NOTE

If the BUSY bit is asserted or if the MB is empty, then reading the Control and Status word does not lock the MB.

1. In previous FlexCAN versions, reading the C/S word locked the MB even if it was EMPTY. This behavior will be honoured when the BCC bit is negated.

Deactivation takes precedence over locking. If the CPU deactivates a locked Rx MB, then its lock status is negated and the MB is marked as invalid for the current matching round. Any pending message on the SMB will not be transferred anymore to the MB

28.2.7.7 CAN Protocol Related Features

28.2.7.7.1 Remote Frames

Remote frame is a special kind of frame. The user can program a MB to be a Request Remote Frame by writing the MB as Transmit with the RTR bit set to '1'. After the Remote Request frame is transmitted successfully, the MB becomes a Receive Message Buffer, with the same ID as before.

When a Remote Request frame is received by FlexCAN, its ID is compared to the IDs of the transmit message buffers with the Code field '1010'. If there is a matching ID, then this MB frame will be transmitted. Note that if the matching MB has the RTR bit set, then FlexCAN will transmit a Remote Frame as a response.

A received Remote Request Frame is not stored in a receive buffer. It is only used to trigger a transmission of a frame in response. The mask registers are not used in remote frame matching, and all ID bits (except RTR) of the incoming received frame should match.

In the case that a Remote Request Frame was received and matched an MB, this message buffer immediately enters the internal arbitration process, but is considered as normal Tx MB, with no higher priority. The data length of this frame is independent of the DLC field in the remote frame that initiated its transmission.

28.2.7.7.2 Overload Frames

FlexCAN does transmit overload frames due to detection of following conditions on CAN bus:

- Detection of a dominant bit in the first/second bit of Intermission
- Detection of a dominant bit at the 7th bit (last) of End of Frame field (Rx frames)
- Detection of a dominant bit at the 8th bit (last) of Error Frame Delimiter or Overload Frame Delimiter

28.2.7.7.3 Time Stamp

The value of the Free Running Timer is sampled at the beginning of the Identifier field on the CAN bus, and is stored at the end of "move-in" in the TIME STAMP field, providing network behavior with respect to time.

Note that the Free Running Timer can be reset upon a specific frame reception, enabling network time synchronization. Refer to TSYN description in Section 28.2.6.3.2, "Control Register (CTRL)."

28.2.7.7.4 Protocol Timing

Figure 28-12 shows the structure of the clock generation circuitry that feeds the CAN Protocol Interface (CPI) sub-module. The clock source bit (CLK_SRC) in the CTRL Register defines whether the internal clock is connected to the output of a crystal oscillator (Oscillator Clock) or to the Peripheral Clock

(generally from a PLL). In order to guarantee reliable operation, the clock source should be selected while the module is in Disable Mode (bit MDIS set in the Module Configuration Register).

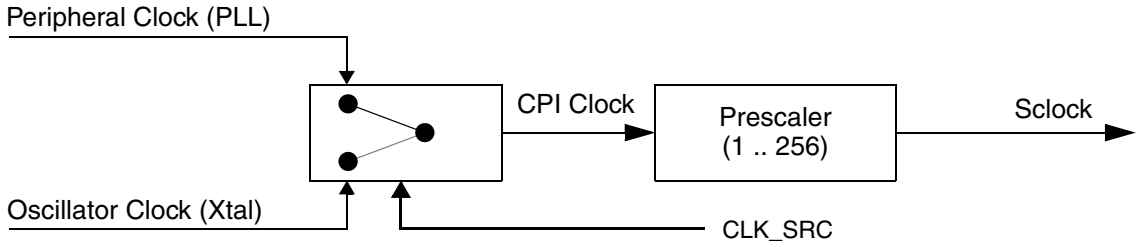


Figure 28-12. CAN Engine Clocking Scheme

The crystal oscillator clock should be selected whenever a tight tolerance (up to 0.1%) is required in the CAN bus timing. The crystal oscillator clock has better jitter performance than PLL generated clocks.

NOTE

This clock selection feature may not be available in all MCUs. A particular MCU may not have a PLL, in which case it would have only the oscillator clock, or it may use only the PLL clock feeding the FlexCAN module. In these cases, the CLK_SRC bit in the CTRL Register has no effect on the module operation.

The FlexCAN module supports a variety of means to setup bit timing parameters that are required by the CAN protocol. The Control Register has various fields used to control bit timing parameters: PRESDIV, PROPSEG, PSEG1, PSEG2 and RJW. See Section 28.2.6.3.2, “Control Register (CTRL).”

The PRESDIV field controls a prescaler that generates the Serial Clock (Sclock), whose period defines the ‘time quantum’ used to compose the CAN waveform. A time quantum is the atomic unit of time handled by the CAN engine.

$$f_{Tq} = \frac{f_{CANCLK}}{\text{Prescaler value}} \qquad \textit{Eqn. 28-1}$$

A bit time is subdivided into three segments[1] (reference Figure 28-13 and Table 28-16):

- SYNC_SEG: This segment has a fixed length of one time quantum. Signal edges are expected to happen within this section
- Time Segment 1: This segment includes the Propagation Segment and the Phase Segment 1 of the CAN standard. It can be programmed by setting the PROPSEG and the PSEG1 fields of the CTRL Register so that their sum (plus 2) is in the range of 4 to 16 time quanta
- Time Segment 2: This segment represents the Phase Segment 2 of the CAN standard. It can be programmed by setting the PSEG2 field of the CTRL Register (plus 1) to be 2 to 8 time quanta long

$$\text{Bit Rate} = \frac{f_{Tq}}{\text{number of Time Quanta}} \qquad \textit{Eqn. 28-2}$$

1. For further explanation of the underlying concepts please refer to ISO/DIS 11519–1, Section 10.3. Reference also the Bosch CAN 2.0A/B protocol specification dated September 1991 for bit timing.

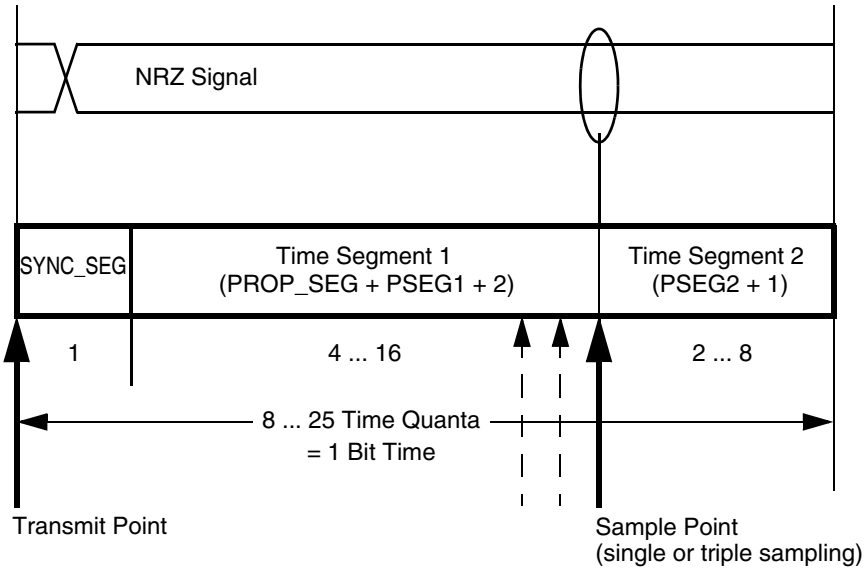


Figure 28-13. Segments within the Bit Time

Table 28-16. Time Segment Syntax

Syntax	Description
SYNC_SEG	System expects transitions to occur on the bus during this period.
Transmit Point	A node in transmit mode transfers a new value to the CAN bus at this point.
Sample Point	A node samples the bus at this point. If the three samples per bit option is selected, then this point marks the position of the third sample.

Figure 28-17 gives an overview of the CAN compliant segment settings and the related parameter values.

Table 28-17. CAN Standard Compliant Bit Time Segment Settings

Time Segment 1	Time Segment 2	Re-synchronization Jump Width
5 .. 10	2	1 .. 2
4 .. 11	3	1 .. 3
5 .. 12	4	1 .. 4
6 .. 13	5	1 .. 4
7 .. 14	6	1 .. 4
8 .. 15	7	1 .. 4
9 .. 16	8	1 .. 4

NOTE

It is the user's responsibility to ensure the bit time settings are in compliance with the CAN standard. For bit time calculations, use an IPT (Information Processing Time) of 2, which is the value implemented in the FlexCAN module.

28.2.7.7.5 Arbitration and Matching Timing

During normal transmission or reception of frames, the arbitration, matching, move-in and move-out processes are executed during certain time windows inside the CAN frame, as shown in Figure 28-14.

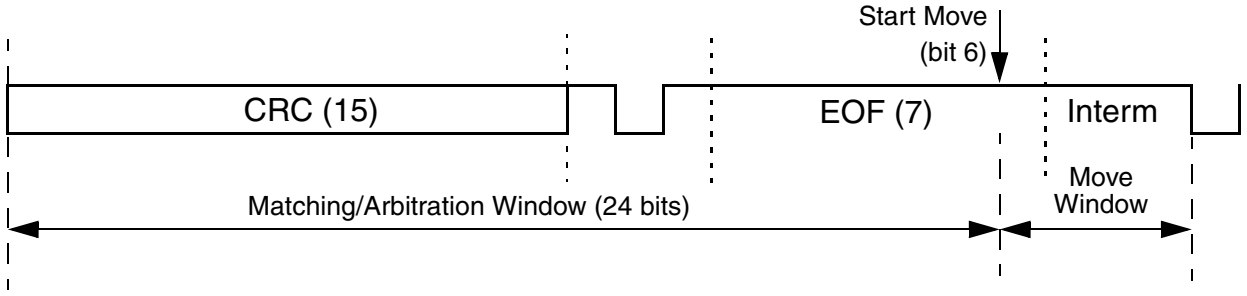


Figure 28-14. Arbitration, Match and Move Time Windows

When doing matching and arbitration, FlexCAN needs to scan the whole Message Buffer memory during the available time slot. In order to have sufficient time to do that, the following requirements must be observed:

- A valid CAN bit timing must be programmed, as indicated in Table 28-17
- The peripheral clock frequency can not be smaller than the oscillator clock frequency, i.e. the PLL can not be programmed to divide down the oscillator clock
- There must be a minimum ratio between the peripheral clock frequency and the CAN bit rate, as specified in Table 28-18

Table 28-18. Minimum Ratio Between Peripheral Clock Frequency and CAN Bit Rate

Number of Message Buffers	Minimum Ratio
16	8
32	8
64	16

A direct consequence of the first requirement is that the minimum number of time quanta per CAN bit must be 8, so the oscillator clock frequency should be at least 8 times the CAN bit rate. The minimum frequency ratio specified in Table 28-18 can be achieved by choosing a high enough peripheral clock frequency when compared to the oscillator clock frequency, or by adjusting one or more of the bit timing parameters (PRESDIV, PROPSEG, PSEG1, PSEG2). As an example, taking the case of 64 MBs, if the oscillator and peripheral clock frequencies are equal and the CAN bit timing is programmed to have 8 time quanta per bit, then the prescaler factor (PRESDIV + 1) should be at least 2. For prescaler factor equal to one and

CAN bit timing with 8 time quanta per bit, the ratio between peripheral and oscillator clock frequencies should be at least 2.

28.2.7.8 Modes of Operation Details

28.2.7.8.1 Freeze Mode

This mode is entered by asserting the HALT bit in the MCR Register or when the MCU is put into Debug Mode. In both cases it is also necessary that the FRZ bit is asserted in the MCR Register and the module is not in any of the low power modes (Disable, Doze, Stop). When Freeze Mode is requested during transmission or reception, FlexCAN does the following:

- Waits to be in either Intermission, Passive Error, Bus Off or Idle state
- Waits for all internal activities like arbitration, matching, move-in and move-out to finish
- Ignores the Rx input pin and drives the Tx pin as recessive
- Stops the prescaler, thus halting all CAN protocol activities
- Grants write access to the Error Counters Register, which is read-only in other modes
- Sets the NOT_RDY and FRZ_ACK bits in MCR

After requesting Freeze Mode, the user must wait for the FRZ_ACK bit to be asserted in MCR before executing any other action, otherwise FlexCAN may operate in an unpredictable way. In Freeze mode, all memory mapped registers are accessible.

Exiting Freeze Mode is done in one of the following ways:

- CPU negates the FRZ bit in the MCR Register
- The MCU is removed from Debug Mode and/or the HALT bit is negated

Once out of Freeze Mode, FlexCAN tries to re-synchronize to the CAN bus by waiting for 11 consecutive recessive bits.

28.2.7.8.2 Module Disable Mode

This low power mode is entered when the MDIS bit in the MCR Register is asserted. If the module is disabled during Freeze Mode, it shuts down the clocks to the CPI and MBM sub-modules, sets the LPM_ACK bit and negates the FRZ_ACK bit. If the module is disabled during transmission or reception, FlexCAN does the following:

- Waits to be in either Idle or Bus Off state, or else waits for the third bit of Intermission and then checks it to be recessive
- Waits for all internal activities like arbitration, matching, move-in and move-out to finish
- Ignores its Rx input pin and drives its Tx pin as recessive
- Shuts down the clocks to the CPI and MBM sub-modules
- Sets the NOT_RDY and LPM_ACK bits in MCR

The Bus Interface Unit continues to operate, enabling the CPU to access memory mapped registers, except the Free Running Timer, the Error Counter Register and the Message Buffers, which cannot be accessed

when the module is in Disable Mode. Exiting from this mode is done by negating the MDIS bit, which will resume the clocks and negate the LPM_ACK bit.

28.2.7.8.3 Doze Mode

This is a system low power mode in which the CPU bus is kept alive and a global Doze Mode request is sent to all peripherals asking them to enter low power mode. When Doze Mode is globally requested, the DOZE bit in MCR needs to have been asserted previously for Doze Mode to be triggered. If Doze Mode is triggered during Freeze Mode, FlexCAN shuts down the clocks to the CPI and MBM sub-modules, sets the LPM_ACK bit and negates the FRZ_ACK bit. If Doze Mode is triggered during transmission or reception, FlexCAN does the following:

- Waits to be in either Idle or Bus Off state, or else waits for the third bit of Intermission and checks it to be recessive
- Waits for all internal activities like arbitration, matching, move-in and move-out to finish
- Ignores its Rx input pin and drives its Tx pin as recessive
- Shuts down the clocks to the CPI and MBM sub-modules
- Sets the NOT_RDY and LPM_ACK bits in MCR

The Bus Interface Unit continues to operate, enabling the CPU to access memory mapped registers, except the Free Running Timer, the Error Counter Register and the Message Buffers, which can not be accessed in Doze Mode.

Exiting Doze Mode is done in one of the following ways:

- CPU removing the Doze Mode request
- CPU negating the DOZE bit of the MCR Register
- Self Wake mechanism

In the Self Wake mechanism, if the SLF_WAK bit in MCR Register was set at the time FlexCAN entered Doze Mode, then upon detection of a recessive to dominant transition on the CAN bus, FlexCAN negates the DOZE bit and resumes its clocks. It also sets the WAK_INT bit in the ESR Register and, if enabled by the WAK_MSK bit in MCR, generates a Wake Up interrupt to the CPU. FlexCAN will then wait for 11 consecutive recessive bits to synchronize to the CAN bus. As a consequence, it will not receive the frame that woke it up. Table 28-19 details the effect of SLF_WAK and WAK_MSK upon wake-up from Doze Mode.

Table 28-19. Wake-up from Doze Mode

SLF_WAK	WAK_MSK	FlexCAN Clocks Enabled	Wake-up Interrupt Generated
0	0	No	No
0	1	No	No
1	0	Yes	No
1	1	Yes	Yes

The sensitivity to CAN bus activity can be modified by applying a low-pass filter function to the Rx CAN input line while in Doze Mode. See the WAK_SRC bit in Section 28.2.6.3.1, “Module Configuration Register (MCR).” This feature can be used to protect FlexCAN from waking up due to short glitches on the CAN bus lines. Such glitches can result from electromagnetic interference within noisy environments.

NOTE

Not all MCUs are equipped with the low-pass filter. Consult the specific MCU documentation to determine if the low-pass filter is available, and to determine its electrical parameters.

28.2.7.8.4 Stop Mode

This is a system low power mode in which all MCU clocks are stopped for maximum power savings. If FlexCAN receives the global Stop Mode request during Freeze Mode, it sets the LPM_ACK bit, negates the FRZ_ACK bit and then sends a Stop Acknowledge signal to the CPU, in order to shut down the clocks globally. If Stop Mode is requested during transmission or reception, FlexCAN does the following:

- Waits to be in either Idle or Bus Off state, or else waits for the third bit of Intermission and checks it to be recessive
- Waits for all internal activities like arbitration, matching, move-in and move-out to finish
- Ignores its Rx input pin and drives its Tx pin as recessive
- Sets the NOT_RDY and LPM_ACK bits in MCR
- Sends a Stop Acknowledge signal to the CPU, so that it can shut down the clocks globally

Exiting Stop Mode is done in one of the following ways:

- CPU resuming the clocks and removing the Stop Mode request
- CPU resuming the clocks and Stop Mode request as a result of the Self Wake mechanism

In the Self Wake mechanism, if the SLF_WAK bit in MCR Register was set at the time FlexCAN entered Stop Mode, then upon detection of a recessive to dominant transition on the CAN bus, FlexCAN sets the WAK_INT bit in the ESR Register and, if enabled by the WAK_MSK bit in MCR, generates a Wake Up interrupt to the CPU. Upon receiving the interrupt, the CPU should resume the clocks and remove the Stop Mode request. FlexCAN will then wait for 11 consecutive recessive bits to synchronize to the CAN bus. As a consequence, it will not receive the frame that woke it up. Table 28-20 details the effect of SLF_WAK and WAK_MSK upon wake-up from Stop Mode. Note that wake-up from Stop Mode only works when both bits are asserted.

Table 28-20. Wake-up from Stop Mode

SLF_WAK	WAK_MSK	MCU Clocks Enabled	Wake-up Interrupt Generated
0	0	No	No
0	1	No	No
1	0	No	No
1	1	Yes	Yes

The sensitivity to CAN bus activity can be modified by applying a low-pass filter function to the Rx CAN input line while in Stop Mode. See the WAK_SRC bit in Section 28.2.6.3.1, “Module Configuration Register (MCR).” This feature can be used to protect FlexCAN from waking up due to short glitches on the CAN bus lines. Such glitches can result from electromagnetic interference within noisy environments.

NOTE

Not all MCUs are equipped with the low-pass filter. Consult the specific MCU documentation to determine if the low-pass filter is available and to determine its electrical parameters.

28.2.7.9 Interrupts

The module can generate up to 70 interrupt sources (64 interrupts due to message buffers and 6 interrupts due to Ored interrupts from MBs, Bus Off, Error, Tx Warning, Rx Warning, and Wake Up). The number of actual sources depends on the configured number of Message Buffers.

Each one of the message buffers can be an interrupt source, if its corresponding IMASK bit is set. There is no distinction between Tx and Rx interrupts for a particular buffer, under the assumption that the buffer is initialized for either transmission or reception. Each of the buffers has assigned a flag bit in the IFLAG Registers. The bit is set when the corresponding buffer completes a successful transmission/reception and is cleared when the CPU writes it to ‘1’ (unless another interrupt is generated at the same time).

NOTE

It must be guaranteed that the CPU only clears the bit causing the current interrupt. For this reason, bit manipulation instructions (BSET) must not be used to clear interrupt flags. These instructions may cause accidental clearing of interrupt flags which are set after entering the current interrupt service routine.

A combined interrupt for all MBs is also generated by an Or of all the interrupt sources from MBs. This interrupt gets generated when any of the MBs generates an interrupt. In this case the CPU must read the IFLAG Registers to determine which MB caused the interrupt.

The other 4 interrupt sources (Bus Off, Error, Warning and Wake Up) generate interrupts like the MB ones, and can be read from the Error and Status Register. The Bus Off , Error and Warning interrupt mask bits are located in the Control Register, and the Wake-Up interrupt mask bit is located in the MCR.

28.2.7.10 Bus Interface

The CPU access to FlexCAN registers are subject to the following rules:

- Read and write access to supervisor registers in User Mode results in access error.
- Read and write access to unimplemented or reserved address space also results in access error. Any access to unimplemented MB or Rx Individual Mask Register locations results in access error. Any access to the Rx Individual Mask Register space when the BCC bit in MCR is negated results in access error.
- If MAXMB is programmed with a value smaller than the available number of MBs, then the unused memory space can be used as general purpose RAM space. Note that the Rx Individual

Mask Registers can only be accessed in Freeze Mode, and this is still true for unused space within this memory. Note also that reserved words within RAM cannot be used. As an example, suppose FlexCAN is configured with 64 MBs and MAXMB is programmed with zero. The maximum number of MBs in this case becomes one. The MB memory starts at $0060, but the space from $0060 to $007F is reserved (for SMB usage), and the space from $0080 to $008F is used by the one MB. This leaves us with the available space from $0090 to $047F. The available memory in the Mask Registers space would be from $0884 to $097F.

NOTE

Unused MB space must not be used as general purpose RAM while FlexCAN is transmitting and receiving CAN frames.

28.2.8 Initialization/Application Information

This section provide instructions for initializing the FlexCAN module.

28.2.8.1 FlexCAN Initialization Sequence

The FlexCAN module may be reset in three ways:

- MCU level hard reset, which resets all memory mapped registers asynchronously
- MCU level soft reset, which resets some of the memory mapped registers synchronously (refer to Table 28-4 to see what registers are affected by soft reset)
- SOFT_RST bit in MCR, which has the same effect as the MCU level soft reset

Soft reset is synchronous and has to follow an internal request/acknowledge procedure across clock domains. Therefore, it may take some time to fully propagate its effects. The SOFT_RST bit remains asserted while soft reset is pending, so software can poll this bit to know when the reset has completed. Also, soft reset can not be applied while clocks are shut down in any of the low power modes. The low power mode should be exited and the clocks resumed before applying soft reset.

The clock source (CLK_SRC bit) should be selected while the module is in Disable Mode. After the clock source is selected and the module is enabled (MDIS bit negated), FlexCAN automatically goes to Freeze Mode. In Freeze Mode, FlexCAN is un-synchronized to the CAN bus, the HALT and FRZ bits in MCR Register are set, the internal state machines are disabled and the FRZ_ACK and NOT_RDY bits in the MCR Register are set. The Tx pin is in recessive state and FlexCAN does not initiate any transmission or reception of CAN frames. Note that the Message Buffers and the Rx Individual Mask Registers are not affected by reset, so they are not automatically initialized.

- For any configuration change/initialization it is required that FlexCAN is put into Freeze Mode (see Section 28.2.7.8.1, “Freeze Mode”). The following is a generic initialization sequence applicable to the FlexCAN module:Initialize the Module Configuration Register
  - Enable the individual filtering per MB and reception queue features by setting the BCC bit
  - Enable the warning interrupts by setting the WRN_EN bit
  - If required, disable frame self reception by setting the SRX_DIS bit
- Initialize the Control Register
  - Determine the bit timing parameters: PROPSEG, PSEG1, PSEG2, RJW

  - Determine the bit rate by programming the PRESDIV field
  - Determine the internal arbitration mode (LBUF bit)
- Initialize the Message Buffers
  - The Control and Status word of all Message Buffers must be initialized
  - Other entries in each Message Buffer should be initialized as required
- Initialize the Rx Individual Mask Registers
- Set required interrupt mask bits in the IMASK Registers (for all MB interrupts), in CTRL Register (for Bus Off and Error interrupts) and in MCR Register for Wake-Up interrupt
- Negate the HALT bit in MCR

Starting with the last event, FlexCAN attempts to synchronize to the CAN bus.

28.2.8.2 FlexCAN Addressing

The user can program the maximum number of MBs that will take part in the matching and arbitration processes using the MAXMB field in the MCR Register. On the MAC72xx, with its 32 Message Buffer configuration, MAXMB can be any number between 0–31.

Chapter 29
Inter-Integrated Circuit Bus Controller Module (I2C_DMA)

29.1 Introduction

The Inter-Integrated Circuit (I^2C™ or IIC) bus is a two-wire bidirectional serial bus that provides a simple and efficient method of data exchange between devices. It minimizes the number of external connections to devices and does not require an external address decoder.

This bus is suitable for applications requiring occasional communications over a short distance between a number of devices. It also provides flexibility, allowing additional devices to be connected to the bus for further expansion and system development.

The interface is designed to operate up to 100kbps with maximum bus loading and timing. The device is capable of operating at higher baud rates, up to a maximum of module clock/20, with reduced bus loading. The maximum communication length and the number of devices that can be connected are limited by a maximum bus capacitance of 400pF.

The I^2C module can be independently disabled by writing to the IBDIS bit in the module's control register (IBCR). Disabling the module will turn off the clock to the module, although the module's registers remain available to be accessed by the core across the peripheral bus. The IBDIS bit is intended to be used when the module is not required in the application. Following a Reset operation the IBDIS bit is set, causing the device to be disabled.

The I^2C master on the MAC72xx provides a fast mode (400 kbits/s) interface for two scenarios:

- Provide a bus interface to external EEPROM. In this scenario, the throughput of the interface will be very low, typically only to load and store parameters and data at reset, or other special conditions.
- Provide a communications channel to off-chip peripherals, such as audio chips or other radio-related peripherals. In this scenario, the throughput will be much higher, but the exact rate is unknown.

29.1.1 Block Diagram

The block diagram of the I^2C module is shown in Figure 29-1

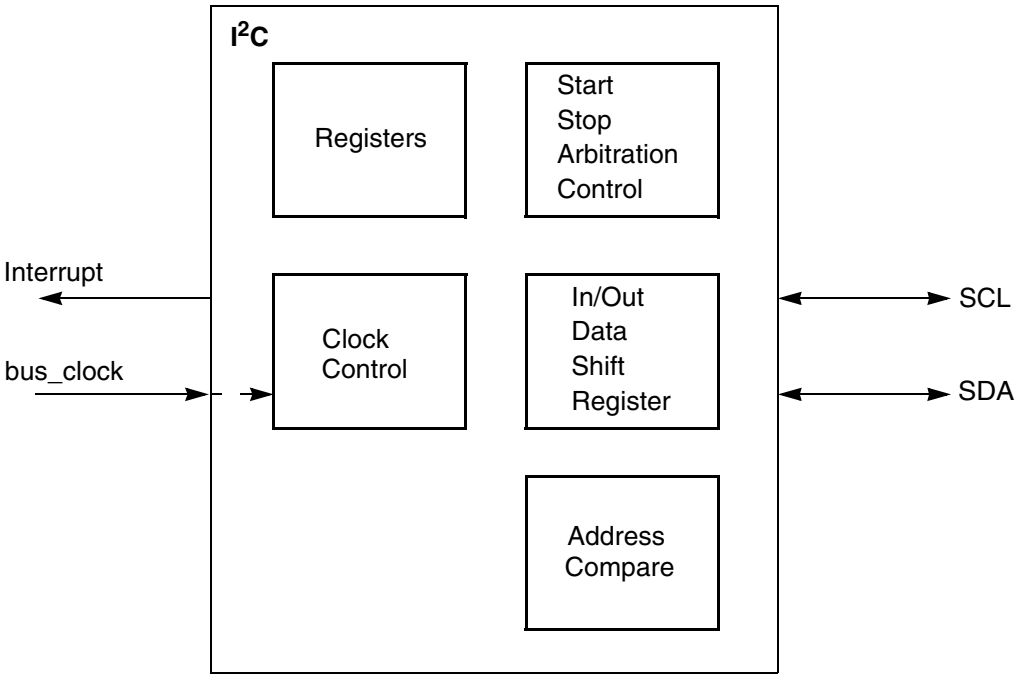


Figure 29-1. I²C Block Diagram

29.1.2 DMA Interface

A simple DMA interface is implemented so that the I²C can request data transfers with minimal support from the CPU. DMA mode is enabled by setting bit 1 in the Control Register.

The DMA interface is only valid when the I²C module is configured for Master Mode.

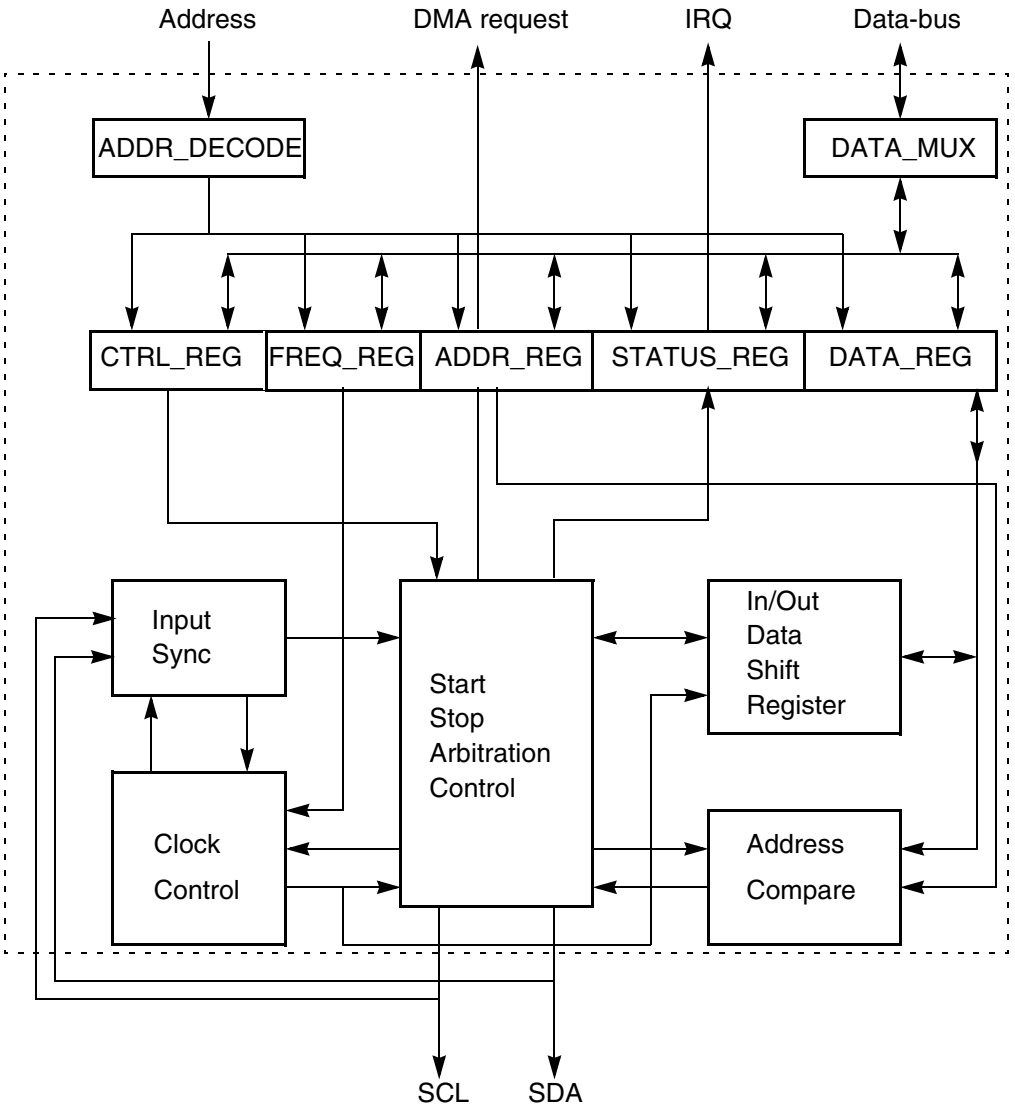


Figure 29-2. I²C Module DMA Interface Block Diagram

At least 3 bytes of data per frame must be transferred from/to the slave when using DMA mode, although in practice it will only be worthwhile using the DMA mode when there is a large number of data bytes to transfer per frame.

Two internal signals, TX request and RX request, are used to signal to a DMA controller when the I²C module requires data to be written or read from the data register.

Further details of the DMA interface can be found in the Initialisation/Application section of this document.

29.1.3 Features

The I²C module has the following key features:

- Compatibility with I²C Bus standard.

- Memory Map: 32-bit peripheral with 8 register bytes, byte/halfword/word addressable
- Two wire bi-directional serial bus for on board communications.
- Multi-master operation.
- Software-programmable for one of 256 different serial clock frequencies.
- Software-selectable acknowledge bit.
- Interrupt-driven byte-by-byte data transfer.
- Arbitration-lost interrupt with automatic mode switching from master to slave.
- Calling address identification interrupt.
- Start and stop signal generation/detection.
- Repeated START signal generation.
- Acknowledge bit generation/detection
- Bus-busy detection.
- Two DMA request lines to receive and transmit data

Features currently not supported:

- No support for general call address
- Not compliant to ten-bit addressing

29.1.4 Modes of Operation

The I^2C module operates in all MAC7200 modes.

- Run Mode

  This is the basic mode of operation.
- Doze Mode

  This is a low-power mode that allows the system to turn off the clock depending on the state of an internal bit. The I^2C block will allow this state to be entered only when there are no active transfers on the bus. Note that a transfer is defined as any active data between valid I^2C Start and I^2C Stop conditions.

29.2 I^2C Module Implementation

The I^2C on the MAC72xx is identical to the I^2C (IIC) on the MAC71xx family of devices.

29.3 External Signal Description

The Inter-Integrated Circuit (I^2C) module has two external pins, SCL and SDA. Refer to Table 4-1 and Section 4.3, “Detailed Signal Descriptions” for detailed descriptions of these signals.

29.4 I^2C Module Differences from MAC71xx

- Fixed MUCts01649 (Write on bus abort still writes register)

29.4.1 Enabling the I²C Module

Before the I²C module can be used, it must be explicitly enabled by clearing the IBDIS/MDIS bit in the ICBR register in the I²C module. If Open Drain functionality is required on the I²C bus, this functionality must be enabled in the Port Integration Module (PIM).

NOTE

Note that, unlike in other peripherals, clearing the IBDIS bit causes a reset of all other I²C registers.

29.5 Memory Map/Register Definition

This section provides a detailed description of all memory-mapped registers in the I²C module.

29.5.1 Module Memory Map

The memory map for the I²C module is given below in Table 29-1. The total address for each register is the sum of the base address for the I²C module and the address offset for each register.

NOTE

Please see Chapter 9, “Device Memory Map” for base address of this module.

Table 29-1. Module Memory Map

Address	Register	Size	Access	Mode[1]
Base + 0x00	I²C Bus Address Register (IBAD)	8	R/W	A
Base + 0x01	I²C Bus Frequency Divider Register (IBFD)	8	R/W	A
Base + 0x02	I²C Bus Control Register (IBCR)	8	R/W	A
Base + 0x03	I²C Bus Status Register (IBSR)	8	R/W	A
Base + 0x04	I²C Bus Data I/O Register (IBDR)	8	R/W	A
Base + 0x05	I²C Bus Interrupt Config Register (IBIC)	8	R/W	A
Base + 0x06	Unused	8	R	A
Base + 0x07	Unused	8	R	A
Base + 0x08 – Base + 0x3FFF	Reserved		See Note [2]	

1. **U** = User Mode, **S** = Supervisor Mode, **A** = All (No restrictions)
2. If enabled at the SoC level, reads or writes to these registers will cause bus aborts. Refer to the System Services Module documentation for more details.

All registers are accessible via 8-bit, 16-bit or 32-bit accesses. However, 16-bit accesses must be aligned to 16-bit boundaries, and 32-bit accesses must be aligned to 32-bit boundaries. As an example, the IBDF register for the frequency devider is accessible by a 16-bit READ/WRITE to address Base + 0x000, but performing a 16-bit access to Base + 0x001 is illegal.

29.5.2 Register Descriptions

This section consists of register descriptions in address order. Each description includes a standard register diagram with an associated figure number. Details of register bit and field function follow the register diagrams, in bit order.

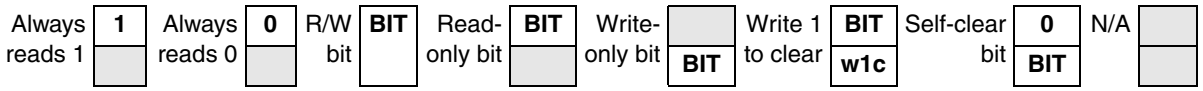


Figure 29-3. Key to Register Fields

29.5.2.1 I²C Address Register

This register contains the address the I²C Bus will respond to when addressed as a slave; note that it is not the address sent on the bus during the address transfer.

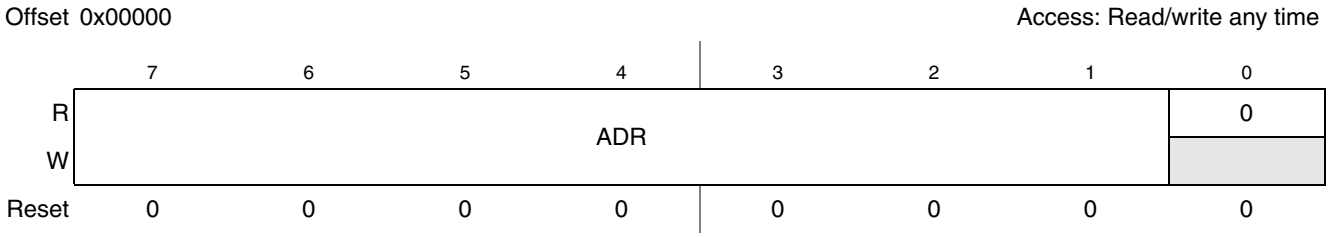


Figure 29-4. I²C Bus Address Register (IBAD)

Table 29-2. IBAD Field Descriptions

Field	Description
7–1 ADR	Slave Address. Specific slave address to be used by the I²C Bus module. Note: The default mode of I²C Bus is slave mode for an address match on the bus.
0	Reserved, should be cleared; will always read 0.

29.5.2.2 I²C Frequency Divider Register

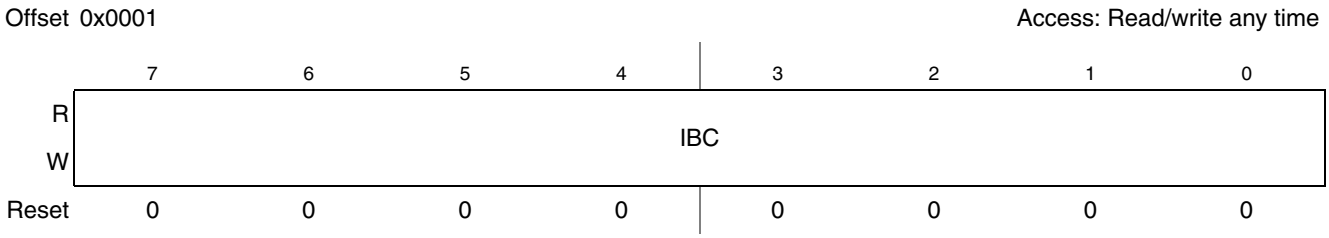


Figure 29-5. I²C Bus Frequency Divider Register (IBFD)

Table 29-3. IBFD Field Descriptions

Field	Description
7–0 IBC	I-Bus Clock Rate. This field is used to prescale the clock for bit rate selection. The bit clock generator is implemented as a prescale divider. The IBC bits are decoded to give the Tap and Prescale values as follows: 7–6 select the prescaled shift register (see Table 29-4) 5–3 select the prescaler divider (see Table 29-5) 2–0 select the shift register tap point (see Table 29-6)

Table 29-4. I-Bus Multiplier Factor

IBC7–6	MUL
00	01
01	02
10	04
11	RESERVED

Table 29-5. I-Bus Prescaler Divider Values

IBC5–3	scl2start (clocks)	scl2stop (clocks)	scl2tap (clocks)	tap2tap (clocks)
000	2	7	4	1
001	2	7	4	2
010	2	9	6	4
011	6	9	6	8
100	14	17	14	16
101	30	33	30	32
110	62	65	62	64
111	126	129	126	128

Table 29-6. I-Bus Tap and Prescale Values

IBC2-0	SCL Tap (clocks)	SDA Tap (clocks)
000	5	1
001	6	1
010	7	2
011	8	2
100	9	3
101	10	3

Table 29-6. I-Bus Tap and Prescale Values

IBC2-0	SCL Tap (clocks)	SDA Tap (clocks)
110	12	4
111	15	4

The number of clocks from the falling edge of SCL to the first tap (Tap[1]) is defined by the values shown in the scl2tap column of Table 29-5. All subsequent tap points are separated by 2^{IBC5-3} as shown in the tap2tap column in Table 29-5. The SCL Tap is used to generate the SCL period and the SDA Tap is used to determine the delay from the falling edge of SCL to the change of state of SDA i.e. the SDA hold time.

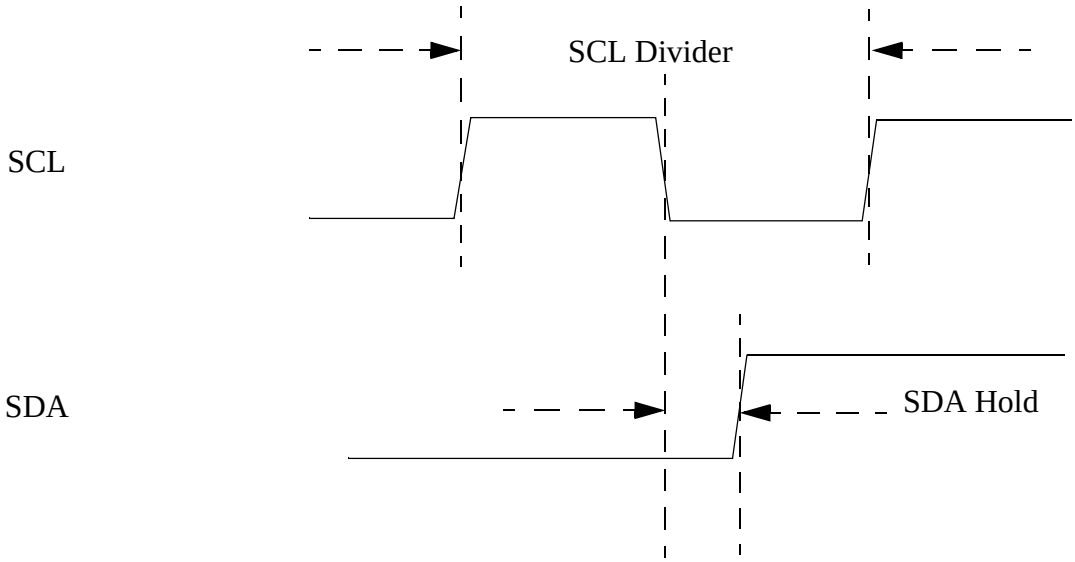


Figure 29-6. SDA Hold Time

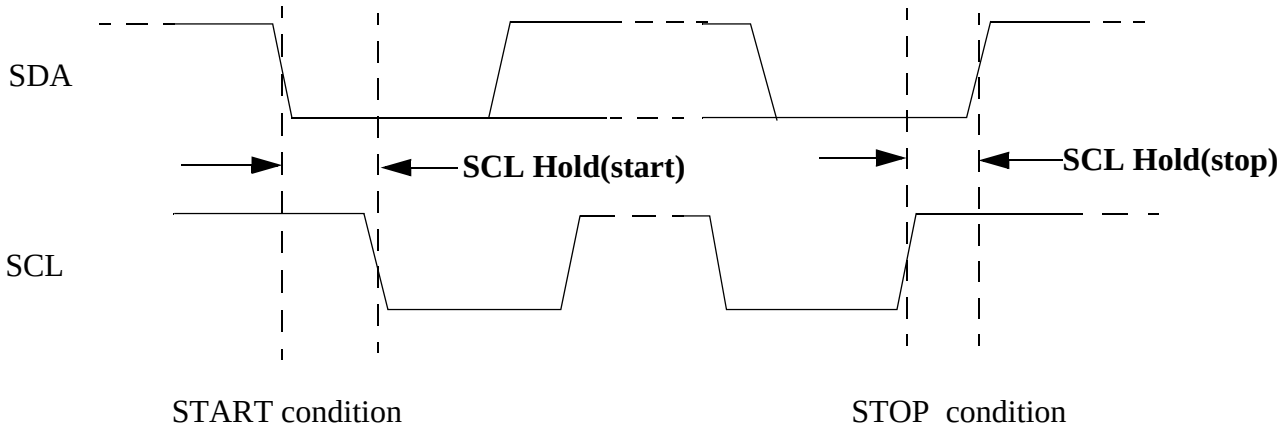


Figure 29-7. SCL Divider and SDA Hold

The equation used to generate the divider values from the IBFD bits is:

$$\text{SCL Divider} = \text{MUL} \times \{2 \times (\text{scl2tap} + [(\text{SCL\_Tap} - 1) \times \text{tap2tap}] + 2)\} \qquad \textit{Eqn. 29-1}$$

The SDA hold delay is equal to the CPU clock period multiplied by the SDA Hold value shown in Table 3-4. The equation used to generate the SDA Hold value from the IBFD bits is:

$$\text{SDA Hold} = \text{MUL} \times \{\text{scl2tap} + [(\text{SDA\_Tap} - 1) \times \text{tap2tap}] + 3\} \quad \textit{Eqn. 29-2}$$

The equation for SCL Hold values to generate the start and stop conditions from the IBFD bits is:

$$\text{SCL Hold(start)} = \text{MUL} \times [\text{scl2start} + (\text{SCL\_Tap} - 1) \times \text{tap2tap}] \quad \textit{Eqn. 29-3}$$

$$\text{SCL Hold(stop)} = \text{MUL} \times [\text{scl2stop} + (\text{SCL\_Tap} - 1) \times \text{tap2tap}] \quad \textit{Eqn. 29-4}$$

Table 29-7. I²C Divider and Hold Values

	IBC7—0 (hex)	SCL Divider (clocks)	SDA Hold (clocks)	SCL Hold (start)	SCL Hold (stop)
MUL = 1	00	20	7	6	11
	01	22	7	7	12
	02	24	8	8	13
	03	26	8	9	14
	04	28	9	10	15
	05	30	9	11	16
	06	34	10	13	18
	07	40	10	16	21
	08	28	7	10	15
	09	32	7	12	17
	0A	36	9	14	19
	0B	40	9	16	21
	0C	44	11	18	23
	0D	48	11	20	25
	0E	56	13	24	29
	0F	68	13	30	35
	10	48	9	18	25
	11	56	9	22	29
	12	64	13	26	33
	13	72	13	30	37
	14	80	17	34	41
	15	88	17	38	45
	16	104	21	46	53
	17	128	21	58	65
	18	80	9	38	41
	19	96	9	46	49
	1A	112	17	54	57
	1B	128	17	62	65
	1C	144	25	70	73
	1D	160	25	78	81
	1E	192	33	94	97
	1F	240	33	118	121
	20	160	17	78	81
	21	192	17	94	97
	22	224	33	110	113
	23	256	33	126	129
	24	288	49	142	145
	25	320	49	158	161
	26	384	65	190	193
	27	480	65	238	241
	28	320	33	158	161
	29	384	33	190	193
	2A	448	65	222	225
	2B	512	65	254	257
	2C	576	97	286	289
	2D	640	97	318	321
	2E	768	129	382	385
	2F	960	129	478	481
	30	640	65	318	321
	31	768	65	382	385
	32	896	129	446	449
	33	1024	129	510	513
	34	1152	193	574	577
	35	1280	193	638	641
	36	1536	257	766	769
	37	1920	257	958	961
	38	1280	129	638	641
	39	1536	129	766	769
	3A	1792	257	894	897
	3B	2048	257	1022	1025
	3C	2304	385	1150	1153
	3D	2560	385	1278	1281
	3E	3072	513	1534	1537
	3F	3840	513	1918	1921

Table 29-7. I²C Divider and Hold Values (Continued)

	IBC7–0 (hex)	SCL Divider (clocks)	SDA Hold (clocks)	SCL Hold (start)	SCL Hold (stop)
MUL = 2	40	40	14	12	22
	41	44	14	14	24
	42	48	16	16	26
	43	52	16	18	28
	44	56	18	20	30
	45	60	18	22	32
	46	68	20	26	36
	47	80	20	32	42
	48	56	14	20	30
	49	64	14	24	34
	4A	72	18	28	38
	4B	80	18	32	42
	4C	88	22	36	46
	4D	96	22	40	50
	4E	112	26	48	58
	4F	136	26	60	70
	50	96	18	36	50
	51	112	18	44	58
	52	128	26	52	66
	53	144	26	60	74
	54	160	34	68	82
	55	176	34	76	90
	56	208	42	92	106
	57	256	42	116	130
	58	160	18	76	82
	59	192	18	92	98
	5A	224	34	108	114
	5B	256	34	124	130
	5C	288	50	140	146
	5D	320	50	156	162
	5E	384	66	188	194
	5F	480	66	236	242
	60	320	28	156	162
	61	384	28	188	194
	62	448	32	220	226
	63	512	32	252	258
	64	576	36	284	290
	65	640	36	316	322
	66	768	40	380	386
	67	960	40	476	482
	68	640	28	316	322
	69	768	28	380	386
	6A	896	36	444	450
	6B	1024	36	508	514
	6C	1152	44	572	578
	6D	1280	44	636	642
	6E	1536	52	764	770
	6F	1920	52	956	962
	70	1280	36	636	642
	71	1536	36	764	770
	72	1792	52	892	898
	73	2048	52	1020	1026
	74	2304	68	1148	1154
	75	2560	68	1276	1282
	76	3072	84	1532	1538
	77	3840	84	1916	1922
	78	2560	36	1276	1282
	79	3072	36	1532	1538
	7A	3584	68	1788	1794
	7B	4096	68	2044	2050
	7C	4608	100	2300	2306
	7D	5120	100	2556	2562
	7E	6144	132	3068	3074
	7F	7680	132	3836	3842

Table 29-7. I²C Divider and Hold Values (Continued)

	IBC7–0 (hex)	SCL Divider (clocks)	SDA Hold (clocks)	SCL Hold (start)	SCL Hold (stop)
MUL = 4	80	80	28	24	44
	81	88	28	28	48
	82	96	32	32	52
	83	104	32	36	56
	84	112	36	40	60
	85	120	36	44	64
	86	136	40	52	72
	87	160	40	64	84
	88	112	28	40	60
	89	128	28	48	68
	8A	144	36	56	76
	8B	160	36	64	84
	8C	176	44	72	92
	8D	192	44	80	100
	8E	224	52	96	116
	8F	272	52	120	140
	90	192	36	72	100
	91	224	36	88	116
	92	256	52	104	132
	93	288	52	120	148
	94	320	68	136	164
	95	352	68	152	180
	96	416	84	184	212
	97	512	84	232	260
	98	320	36	152	164
	99	384	36	184	196
	9A	448	68	216	228
	9B	512	68	248	260
	9C	576	100	280	292
	9D	640	100	312	324
	9E	768	132	376	388
	9F	960	132	472	484
	A0	640	68	312	324
	A1	768	68	376	388
	A2	896	132	440	452
	A3	1024	132	504	516
	A4	1152	196	568	580
	A5	1280	196	632	644
	A6	1536	260	760	772
	A7	1920	260	952	964
	A8	1280	132	632	644
	A9	1536	132	760	772
	AA	1792	260	888	900
	AB	2048	260	1016	1028
	AC	2304	388	1144	1156
	AD	2560	388	1272	1284
	AE	3072	516	1528	1540
	AF	3840	516	1912	1924
	30	2560	260	1272	1284
	B1	3072	260	1528	1540
	B2	3584	516	1784	1796
	B3	4096	516	2040	2052
	B4	4608	772	2296	2308
	B5	5120	772	2552	2564
	B6	6144	1028	3064	3076
	B7	7680	1028	3832	3844
	B8	5120	516	2552	2564
	B9	6144	516	3064	3076
	BA	7168	1028	3576	3588
	BB	8192	1028	4088	4100
	BC	9216	1540	4600	4612
	BD	10240	1540	5112	5124
	BE	12288	2052	6136	6148
	BF	15360	2052	7672	7684

29.5.2.3 I²C Control Register

Offset 0x0002 Access: Read/write any time

	7	6	5	4	3	2	1	0
R	MDIS	IBIE	MS/$\overline{SL}$	Tx/$\overline{Rx}$	NOACK	0	DMAEN	IBDOZE
W						RSTA		
Reset	1	0	0	0	0	0	0	0

Figure 29-8. I²C Bus Control Register (IBCR)

Table 29-8. IBCR Field Descriptions

Field	Description
7 MDIS	Module disable. This bit controls the software reset of the entire I²C Bus module. 1 The module is reset and disabled. This is the power-on reset situation. When high, the interface is held in reset, but registers can still be accessed 0 The I²C Bus module is enabled. This bit must be cleared before any other IBCR bits have any effect Note: If the I²C Bus mdule is enabled in the middle of a byte transfer, the interface behaves as follows: slave mode ignores the current transfer on the bus and starts operating whenever a subsequent start condition is detected. Master mode will not be aware that the bus is busy, hence if a start cycle is initiated then the current bus cycle may become corrupt. This would ultimately result in either the current bus master or the I²C Bus module losing arbitration, after which, bus operation would return to normal.
6 IBIE	I-Bus Interrupt Enable. 1 Interrupts from the I²C Bus module are enabled. An I²C Bus interrupt occurs provided the IBIF bit in the status register is also set. 0 Interrupts from the I²C Bus module are disabled. Note that this does not clear any currently pending interrupt condition
5 MS/$\overline{SL}$	Master/Slave mode select. Upon reset, this bit is cleared. When this bit is changed from 0 to 1, a START signal is generated on the bus and the master mode is selected. When this bit is changed from 1 to 0, a STOP signal is generated and the operation mode changes from master to slave. A STOP signal should be generated only if the IBIF flag is set. MS/$\overline{SL}$ is cleared without generating a STOP signal when the master loses arbitration. 1 Master Mode 0 Slave Mode
4 Tx/$\overline{Rx}$	Transmit/Receive mode select. This bit selects the direction of master and slave transfers. When addressed as a slave this bit should be set by software according to the SRW bit in the status register. In master mode this bit should be set according to the type of transfer required. Therefore, for address cycles, this bit will always be high. 1 Transmit 0 Receive
3 NOACK	Data Acknowledge disable. This bit specifies the value driven onto SDA during data acknowledge cycles for both master and slave receivers. The I²C module will always acknowledge address matches, provided it is enabled, regardless of the value of NOACK. Note that values written to this bit are only used when the I²C Bus is a receiver, not a transmitter. 1 No acknowledge signal response is sent (i.e., acknowledge bit = 1) 0 An acknowledge signal will be sent out to the bus at the 9th clock bit after receiving one byte of data
2 RSTA	Repeat Start. Writing a 1 to this bit will generate a repeated START condition on the bus, provided it is the current bus master. This bit will always be read as a low. Attempting a repeated start at the wrong time, if the bus is owned by another master, will result in loss of arbitration. 1 Generate repeat start cycle 0 No effect

Table 29-8. IBCR Field Descriptions (Continued)

Field	Description
1 DMAEN	DMA Enable. When this bit is set, the DMA TX and RX lines will be asserted when the I²C module requires data to be read or written to the data register. No Transfer Done interrupts will be generated when this bit is set, however an interrupt will be generated if the loss of arbitration or addressed as slave conditions occur. The DMA mode is only valid when the I²C module is configured as a Master and the DMA transfer still requires CPU intervention at the start and the end of each frame of data. See the DMA Application Information section for more details. 1 Enable the DMA TX/RX request signals 0 Disable the DMA TX/RX request signals
0 IBSDOZE	I-Bus Interface Stop in DOZE mode. 1 Halt I²C Bus module clock generation (if DOZE mode signal asserted) 0 I²C Bus module clock operates normally Note: If the IBSDOZE mode is SET, the I²C module will enter DOZE mode when the DOZE signal is asserted, if there are no current transactions on the bus. The I²C module would then signal to the system that the clock can be shut down. If it were the case that the IBDOZE bit was cleared when the DOZE signal was asserted, the I²C Bus module clock would remain alive, and any current transactions would continue as normal.

29.5.2.4 I²C Status Register

Offset 0x0003 — Access: Read-only any time[1]

	7	6	5	4	3	2	1	0
R	TCF	IAAS	IBB	IBAL	0	SRW	IBIF	RXAK
W								
Reset	1	0	0	0	0	0	0	0

Figure 29-9. I²C Bus Status Register(IBSR)

1. With the exception of IBIF and IBAL, which are software clearable.

Table 29-9. IBSR Field Descriptions

Field	Description
7 TCF	Transfer complete. While one byte of data is being transferred, this bit is cleared. It is set by the falling edge of the 9th clock of a byte transfer. Note that this bit is only valid during or immediately following a transfer to the I²C module or from the I²C module. 1 Transfer complete 0 Transfer in progress
6 IAAS	Addressed as a slave. When its own specific address (I-Bus Address Register) is matched with the calling address, this bit is set. The CPU is interrupted provided the IBIE is set. Then the CPU needs to check the SRW bit and set its Tx/$\overline{Rx}$ mode accordingly. Writing to the I-Bus Control Register clears this bit. 1 Addressed as a slave 0 Not addressed
5 IBB	Bus busy. This bit indicates the status of the bus. When a START signal is detected, the IBB is set. If a STOP signal is detected, IBB is cleared and the bus enters idle state. 1 Bus is busy 0 Bus is Idle

Table 29-9. IBSR Field Descriptions (Continued)

Field	Description
4 IBAL	Arbitration Lost. The arbitration lost bit (IBAL) is set by hardware when the arbitration procedure is lost. Arbitration is lost in the following circumstances: • SDA is sampled low when the master drives a high during an address or data transmit cycle. • SDA is sampled low when the master drives a high during the acknowledge bit of a data receive cycle. • A start cycle is attempted when the bus is busy. • A repeated start cycle is requested in slave mode. • A stop condition is detected when the master did not request it. This bit must be cleared by software, by writing a one to it. A write of zero has no effect.
3	Reserved for future use. A read will return 0; should be written as 0.
2 SRW	Slave Read/Write. When IAAS is set, this bit indicates the value of the R/W command bit of the calling address sent from the master. This bit is only valid when the I-Bus is in slave mode, a complete address transfer has occurred with an address match and no other transfers have been initiated. By programming this bit, the CPU can select slave transmit/receive mode according to the command of the master. 1 Slave transmit, master reading from slave 0 Slave receive, master writing to slave
1 IBIF	I-Bus Interrupt Flag. The IBIF bit is set when one of the following conditions occurs: • Arbitration lost (IBAL bit set) • Byte transfer complete (TCF bit set and DMAEN bit not set) • Addressed as slave (IAAS bit set) • NoAck from Slave (MS & Tx bits set) • I^2C Bus going idle (IBB high-low transition and enabled by BIIE) A processor interrupt request will be caused if the IBIE bit is set. This bit must be cleared by software, by writing a one to it. A write of zero has no effect on this bit. In DMA mode (DMAEN set) a byte transfer complete condition will not trigger the setting of IBIF. All other conditions still apply.
0 RXAK	Received Acknowledge. This is the value of SDA during the acknowledge bit of a bus cycle. If the received acknowledge bit (RXAK) is low, it indicates an acknowledge signal has been received after the completion of 8 bits data transmission on the bus. If RXAK is high, it means no acknowledge signal is detected at the 9th clock. 1 No acknowledge received 0 Acknowledge received

29.5.2.5 I^2C Data I/O Register

Offset 0x0004 Access: Read/write any time

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

Figure 29-10. I^2C Bus Data I/O Register (IBDR)

In master transmit mode, when data is written to IBDR, a data transfer is initiated. The most significant bit is sent first. In master receive mode, reading this register initiates next byte data receiving. In slave mode, the same functions are available after an address match has occurred. Note that the Tx/$\overline{\text{Rx}}$ bit in the IBCR must correctly reflect the desired direction of transfer in master and slave modes for the transmission to begin. For instance, if the I^2C is configured for master transmit but a master receive is desired, then reading the IBDR will not initiate the receive.

Reading the IBDR will return the last byte received while the I^{2}C is configured in either master receive or slave receive modes. The IBDR does not reflect every byte that is transmitted on the I^{2}C bus, nor can software verify that a byte has been written to the IBDR correctly by reading it back.

In master transmit mode, the first byte of data written to IBDR following assertion of MS/$\overline{SL}$ is used for the address transfer and should comprise the calling address (in position D7–D1) concatenated with the required R/$\overline{W}$ bit (in position D0).

29.5.2.6 I^{2}C Interrupt Config Register

Offset 0x0005 Access: Read/write any time

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

Figure 29-11. I^{2}C Bus Interrupt Config Register (IBIC)

Table 29-10. IBIC Field Descriptions

Field	Description
7 BIIE	Bus Idle Interrupt Enable bit. This config bit can be used to enable the generation of an interrupt once the I^{2}C bus becomes idle. Once this bit is set, an IBB high-low transition will set the IBIF bit. This feature can be used to signal to the CPU the completion of a STOP on the I^{2}C bus. 1 Bus Idle Interrupts enabled 0 Bus Idle Interrupts disabled
6–0	Reserved for future use. A read will return 0; should be written as 0.

29.6 Functional Description

29.6.1 General

This section provides a complete functional description of the Inter-Integrated Circuit (I^{2}C).

29.6.2 I-Bus Protocol

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

Normally, a standard communication is composed of four parts: START signal, slave address transmission, data transfer and STOP signal. They are described briefly in the following sections and illustrated in Figure 29-12.

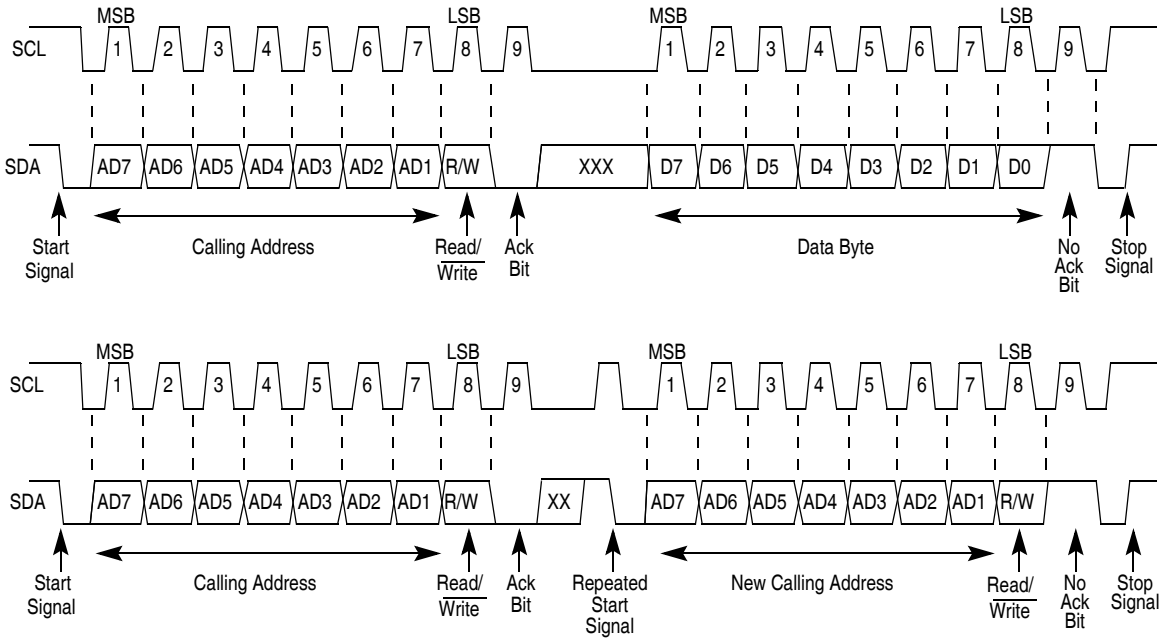


Figure 29-12. I²C Bus Transmission Signals

29.6.2.1 START Signal

When the bus is free, i.e. no master device is engaging the bus (both SCL and SDA lines are at logical high), a master may initiate communication by sending a START signal. As shown in Figure 29-12, a START signal is defined as a high-to-low transition of SDA while SCL is high. This signal denotes the beginning of a new data transfer (each data transfer may contain several bytes of data) and brings all slaves out of their idle states.

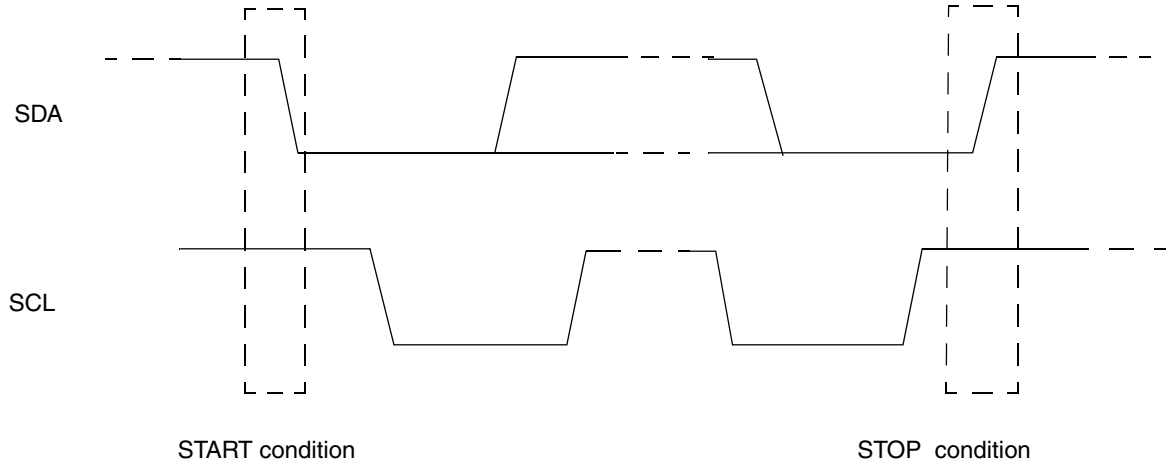


Figure 29-13. Start and Stop conditions

29.6.2.2 Slave Address Transmission

The first byte of data transfer immediately after the START signal is the slave address transmitted by the master. This is a seven-bit calling address followed by a R/W bit. The R/W bit tells the slave the desired direction of data transfer.

1 = Read transfer - the slave transmits data to the master

0 = Write transfer - the master transmits data to the slave

Only the slave with a calling address that matches the one transmitted by the master will respond by sending back an acknowledge bit. This is done by pulling the SDA low at the 9th clock (see Figure 29-12).

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

29.6.2.3 Data Transfer

Once successful slave addressing is achieved, the data transfer can proceed byte-by-byte in a direction specified by the R/W bit sent by the calling master.

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

Each data byte is 8 bits long. Data may be changed only while SCL is low and must be held stable while SCL is high as shown in Figure 29-12. There is one clock pulse on SCL for each data bit, the MSB being transferred first. Each data byte must be followed by an acknowledge bit, which is signalled from the receiving device by pulling the SDA low at the ninth clock. Therefore, one complete data byte transfer needs nine clock pulses.

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 signal to abort the data transfer or a start signal (repeated start) to commence a new calling.

If the master receiver does not acknowledge the slave transmitter after a byte transmission, it means 'end of data' to the slave, so the slave releases the SDA line for the master to generate a STOP or START signal.

29.6.2.4 STOP Signal

The master can terminate the communication by generating a STOP signal to free the bus. However, the master may generate a START signal followed by a calling command without generating a STOP signal first. This is called repeated START. A STOP signal is defined as a low-to-high transition of SDA while SCL is at logical “1” (see Figure 29-12).

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

29.6.2.5 Repeated START Signal

As shown in Figure 29-12, a repeated START signal is a START signal generated without first generating a STOP signal to terminate the communication. This is used by the master to communicate with another slave or with the same slave in different mode (transmit/receive mode) without releasing the bus.

29.6.2.6 Arbitration Procedure

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

29.6.2.7 Clock Synchronization

Since wire-AND logic is performed on the SCL line, a high-to-low transition on the SCL line affects all the devices connected on the bus. The devices start counting their low period and once a device's clock has gone low, it holds the SCL line low until the clock high state is reached. However, the change of low to high in this device clock may not change the state of the SCL line if another device clock is still within its low period. Therefore, synchronized clock SCL is held low by the device with the longest low period. Devices with shorter low periods enter a high wait state during this time (see Figure 29-14). When all devices concerned have counted off their low period, the synchronized clock SCL line is released and pulled high. There is then no difference between the device clocks and the state of the SCL line and all the devices start counting their high periods. The first device to complete its high period pulls the SCL line low again.

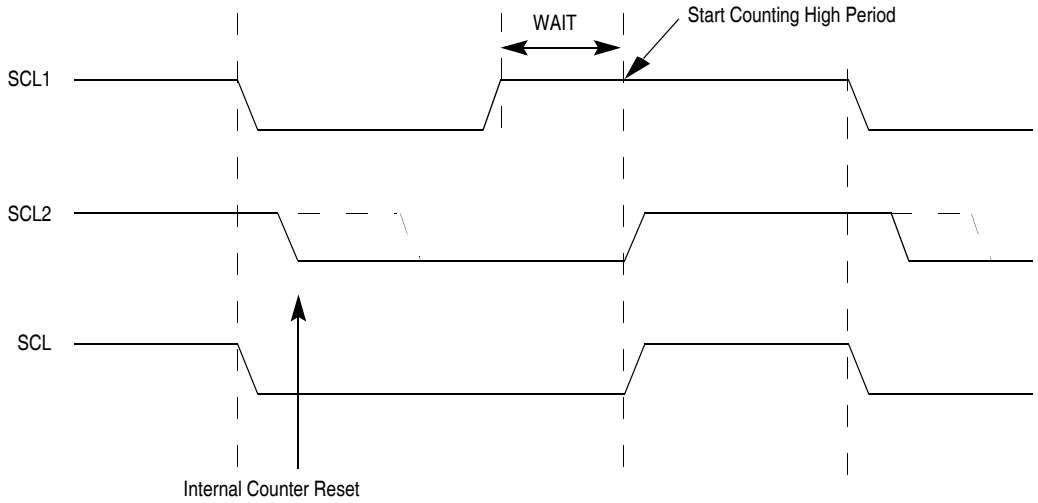


Figure 29-14. I²C Bus Clock Synchronization

29.6.2.8 Handshaking

The clock synchronization mechanism can be used as a handshake in data transfer. Slave devices may hold the SCL low after completion of one byte transfer (9 bits). In such cases, it halts the bus clock and forces the master clock into wait state until the slave releases the SCL line.

29.6.2.9 Clock Stretching

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

29.6.3 Interrupts

29.6.3.1 General

The I2C_DMA uses only one interrupt vector.

Table 29-11. Interrupt Summary

Interrupt	Offset	Vector	Priority	Source	Description
I^2C Interrupt	—	—	—	IBAL, TCF, IAAS, IBB bits in IBSR register	When any of IBAL, TCF or IAAS bits is set an interrupt may be caused based on Arbitration lost, Transfer Complete or Address Detect conditions. If enabled by BIIE, the deassertion of IBB can also cause an interrupt, indicating that the bus is idle.

29.6.3.2 Interrupt Description

There are five types of internal interrupts in the I^2C. The interrupt service routine can determine the interrupt type by reading the Status Register.

I^2C Interrupt can be generated on

- Arbitration Lost condition (IBAL bit set)
- Byte Transfer condition (TCF bit set and DMAEN bit not set)
- Address Detect condition (IAAS bit set)
- No Acknowledge from slave received when expected
- Bus Going Idle (IBB bit not set)

The I^2C interrupt is enabled by the IBIE bit in the I^2C Control Register. It must be cleared by writing ‘1’ to the IBIF bit in the interrupt service routine. The Bus Going Idle interrupt needs to be additionaly enabled by the BIIE bit in the IBIC register.

29.7 Initialization/Application Information

29.7.1 I²C Programming Examples

29.7.1.1 Initialization Sequence

Reset will put the I²C Bus Control Register to its default state. Before the interface can be used to transfer serial data, an initialization procedure must be carried out, as follows:

1. Update the Frequency Divider Register (IBFD) and select the required division ratio to obtain SCL frequency from system clock.
2. Update the I²C Bus Address Register (IBAD) to define its slave address.
3. Clear the IBDIS bit of the I²C Bus Control Register (IBCR) to enable the I²C interface system.
4. Modify the bits of the I²C Bus Control Register (IBCR) to select Master/Slave mode, Transmit/Receive mode and interrupt enable or not. Optionally also modify the bits of the I²C Bus Interrupt Config Register (IBIC) to further refine the interrupt behaviour.

29.7.1.2 Generation of START

After completion of the initialization procedure, serial data can be transmitted by selecting the 'master transmitter' mode. If the device is connected to a multi-master bus system, the state of the I²C Bus Busy bit (IBB) must be tested to check whether the serial bus is free.

If the bus is free (IBB=0), the start condition and the first byte (the slave address) can be sent. The data written to the data register comprises the slave calling address and the LSB, which is set to indicate the direction of transfer required from the slave.

The bus free time (i.e., the time between a STOP condition and the following START condition) is built into the hardware that generates the START cycle. Depending on the relative frequencies of the system clock and the SCL period, it may be necessary to wait until the I²C is busy after writing the calling address to the IBDR before proceeding with the following instructions. This is illustrated in the following example.

An example of the sequence of events which generates the START signal and transmits the first byte of data (slave address) is shown below:

```
while (bit 5, IBSR ==1)// wait in loop for IBB flag to clear
bit4 and bit 5, IBCR = 1// set transmit and master mode, i.e. generate start condition
IBDR = calling_address// send the calling address to the data register
while (bit 5, IBSR ==0)// wait in loop for IBB flag to be set
```

29.7.1.3 Post-Transfer Software Response

Transmission or reception of a byte will set the data transferring bit (TCF) to 1, which indicates one byte communication is finished. The I²C Bus interrupt bit (IBIF) is set also; an interrupt will be generated if the interrupt function is enabled during initialization by setting the IBIE bit. The IBIF (interrupt flag) can be cleared by writing 1 (in the interrupt service routine, if interrupts are used).

The TCF bit will be cleared to indicate data transfer in progress by reading the IBDR data register in receive mode or writing the IBDR in transmit mode. The TCF bit should not be used as a data transfer

complete flag as the flag timing is dependent on a number of factors including the I^2C bus frequency. This bit may not conclusively provide an indication of a transfer complete situation. It is recommended that transfer complete situations are detected using the IBIF flag

Software may service the I^2C I/O in the main program by monitoring the IBIF bit if the interrupt function is disabled. Note that polling should monitor the IBIF bit rather than the TCF bit since their operation is different when arbitration is lost.

Note that when an interrupt occurs at the end of the address cycle, the master will always be in transmit mode, i.e. the address is transmitted. If master receive mode is required, indicated by R/W bit in IBDR, then the Tx/Rx bit should be toggled at this stage.

During slave mode address cycles (IAAS=1) the SRW bit in the status register is read to determine the direction of the subsequent transfer and the Tx/Rx bit is programmed accordingly. For slave mode data cycles (IAAS=0) the SRW bit is not valid. The Tx/Rx bit in the control register should be read to determine the direction of the current transfer.

The following is an example software sequence for 'master transmitter' in the interrupt routine.

```
clear bit 1, IBSR          // Clear the IBIF flag
if (bit 5, IBCR ==0)
    slave_mode()           // run slave mode routine
if (bit 4, IBCR ==0))
    receive_mode()         // run receive_mode routine
if (bit 0, IBSR == 1)      // if NO ACK
     end();                // end transmission
else
IBDR = data_to_transmit    // transmit next byte of data
```

29.7.1.4 Generation of STOP

A data transfer ends with a STOP signal generated by the 'master' device. A master transmitter can simply generate a STOP signal after all the data has been transmitted. The following is an example showing how a stop condition is generated by a master transmitter.

```
if (tx_count == 0) or      // check to see if all data bytes have been transmitted
    (bit 0, IBSR == 1) {   // or if no ACK generated
         clear bit 5, IBCR // generate stop condition
        }
else {
IBDR = data_to_transmit    // write byte of data to DATA register
          tx_count --      // decrement counter
      }                    // return from interrupt
```

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 can be done by setting the transmit acknowledge bit (TXAK) before reading the 2nd last byte of data. Before reading the last byte of data, a STOP signal must first be generated. The following is an example showing how a STOP signal is generated by a master receiver.

```
rx_count --                // decrease the rx counter
if (rx_count ==1)          // 2nd last byte to be read ?
   bit 3, IBCR = 1         // disable ACK
if (rx_count == 0)         // last byte to be read ?
  bit 1, IBCR = 0          // generate stop signal
else
```

```
data_received = IBDR        // read RX data and store
```

29.7.1.5 Generation of Repeated START

At the end of data transfer, if the master still wants to communicate on the bus, it can generate another START signal followed by another slave address without first generating a STOP signal. A program example is as shown.

```
bit 2, IBCR = 1             // generate another start ( restart)
IBDR == calling_address     // transmit the calling address
```

29.7.1.6 Slave Mode

In the slave interrupt service routine, the module addressed as slave bit (IAAS) should be tested to check if a calling of its own address has just been received. If IAAS is set, software should set the transmit/receive mode select bit (Tx/Rx bit of IBCR) according to the R/W command bit (SRW). Writing to the IBCR clears IAAS automatically. Note that the only time IAAS is read as set is from the interrupt at the end of the address cycle where an address match occurred. Interrupts resulting from subsequent data transfers will have IAAS cleared. A data transfer may now be initiated by writing information to IBDR for slave transmits or dummy reading from IBDR in slave receive mode. The slave will drive SCL low in-between byte transfers SCL is released when the IBDR is accessed in the required mode.

In slave transmitter routine, the received acknowledge bit (RXAK) must be tested before transmitting the next byte of data. Setting RXAK means an 'end of data' signal from the master receiver, after which it must be switched from transmitter mode to receiver mode by software. A dummy read then releases the SCL line so that the master can generate a STOP signal.

29.7.1.7 Arbitration Lost

If several masters try to engage the bus simultaneously, only one master wins and the others lose arbitration. The devices that lost arbitration are immediately switched to slave receive mode by the hardware. Their data output to the SDA line is stopped, but SCL is still generated until the end of the byte during which arbitration was lost. An interrupt occurs at the falling edge of the ninth clock of this transfer with IBAL=1 and MS/SL=0. If one master attempts to start transmission, while the bus is being engaged by another master, the hardware will inhibit the transmission, switch the MS/SL bit from 1 to 0 without generating a STOP condition, generate an interrupt to CPU and set the IBAL to indicate that the attempt to engage the bus is failed. When considering these cases, the slave service routine should test the IBAL first and the software should clear the IBAL bit if it is set.

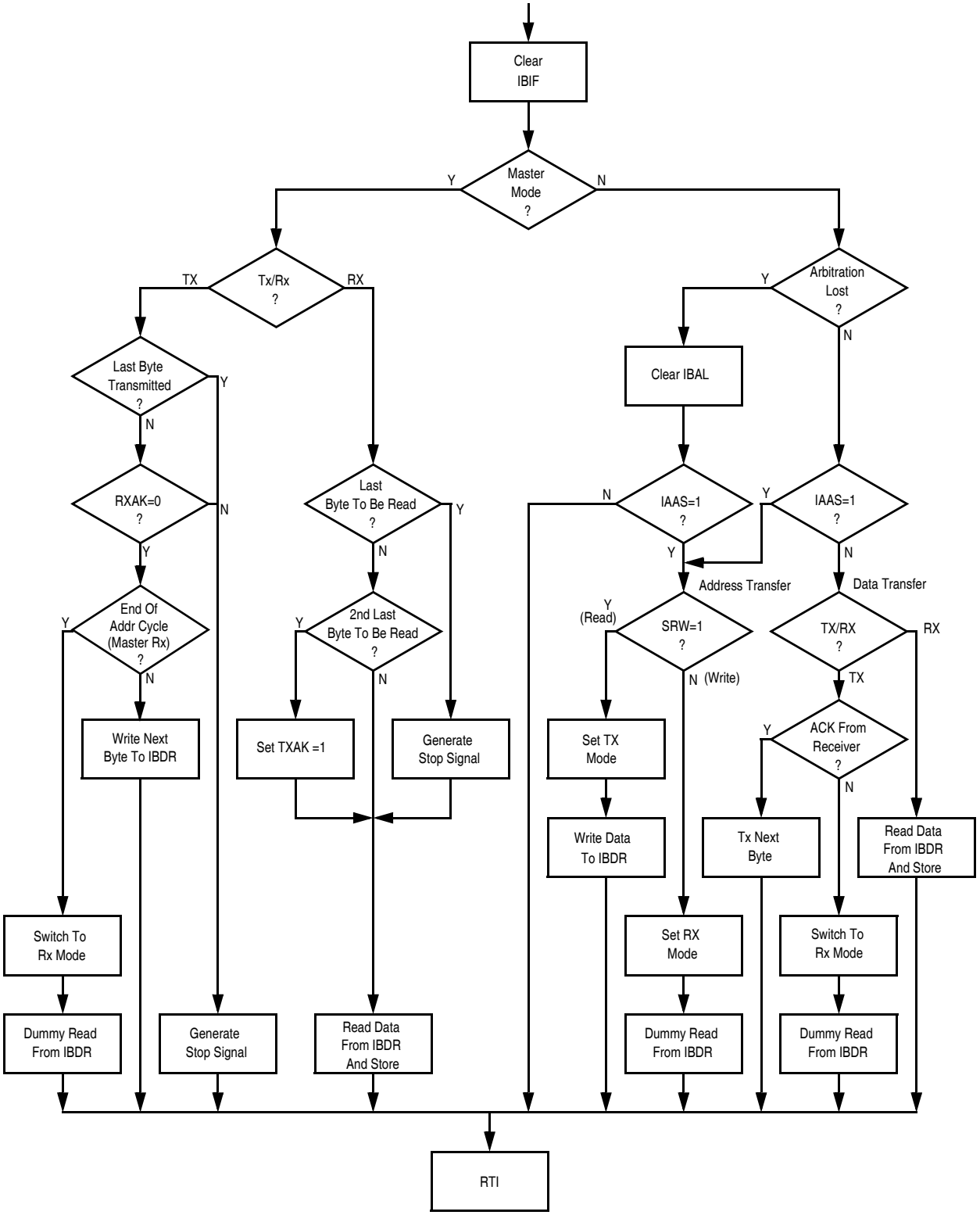


Figure 29-15. Flow-Chart of Typical I^{2}C Interrupt Routine

29.7.2 DMA Application Information

The DMA interface on the I^2C is not completely autonomous and requires intervention from the CPU to start and to terminate the frame transfer. DMA mode is only valid for Master transmit and Master receive modes. Software must ensure that the DMA enable bit in the control register is not set when the I^2C module is configured in master mode.

The DMA controller must only transfer one byte of data per Tx/Rx request. This is because there is no FIFO on the I^2C block.

The CPU should also keep the I^2C interrupt enabled during a DMA transfer to detect the arbitration lost condition and take action to recover from this situation. The DMAEN bit in the IBCR register works as a disable for the transfer complete interrupt. This means that during normal transfers (no errors) there will always be either an interrupt or a request to the DMA controller, dependent on the setting of the DMAEN bit. All error conditions will trigger an interrupt and require CPU intervention. The address match condition will not occur in DMA mode as the I^2C should never be configured for slave operation.

The following sections detail how to set up a DMA transfer and what intervention is required from the CPU. It is assumed that the system DMA controller is capable of generating an interrupt after a certain number of DMA transfers have taken place.

29.7.2.1 DMA Mode, Master Transmit

The following flow diagram details exactly the operation for using a DMA controller to transmit “n” data bytes to a slave. The first byte (the slave calling address) is always transmitted by the CPU. All subsequent data bytes (apart from the last data byte) can be transferred by the DMA controller. The last data byte must be transferred by the CPU.

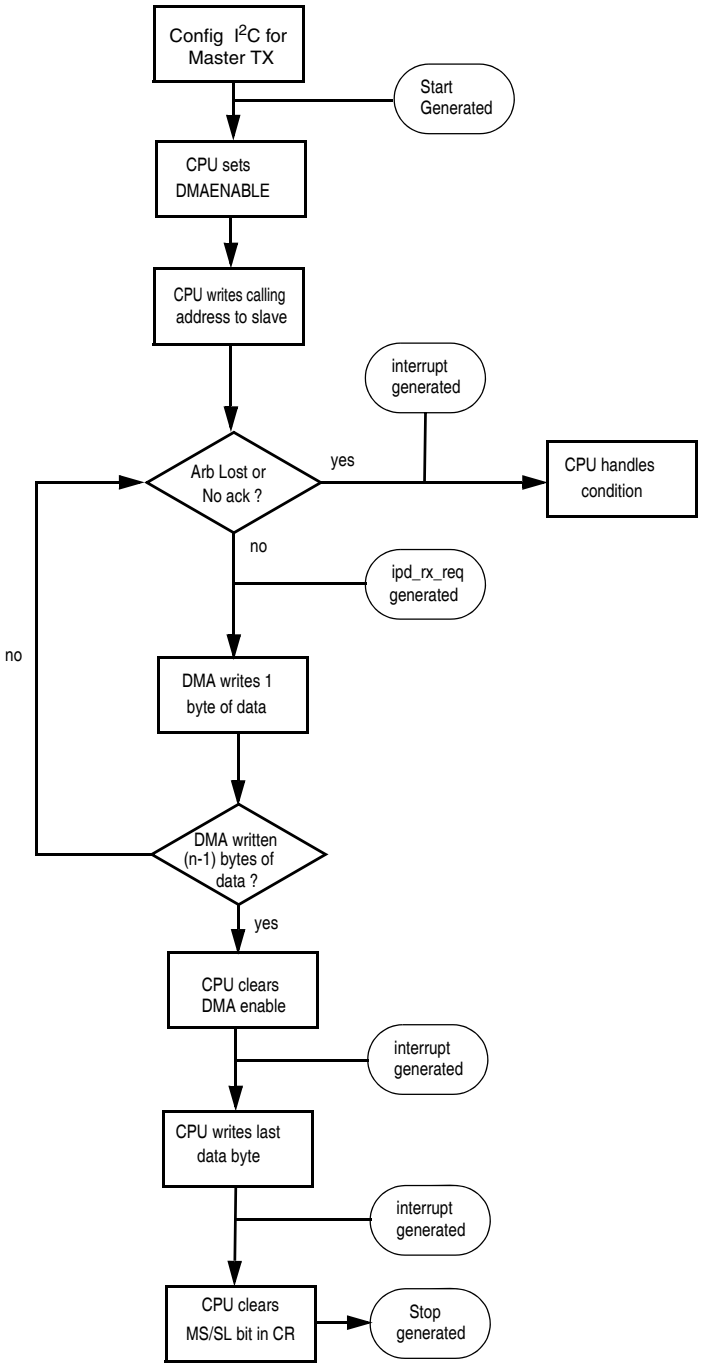


Figure 29-16. Flow-Chart of DMA Mode Master Transmit

29.7.2.2 DMA Mode, Master RX

The following flow diagram details the exact operation for using a DMA controller to receive “n” data bytes from a slave. The first byte (the slave calling address) is always transmitted by the CPU. All subsequent data bytes (apart from the two last data bytes) can be read by the DMA controller. The last two data bytes must be transferred by the CPU.

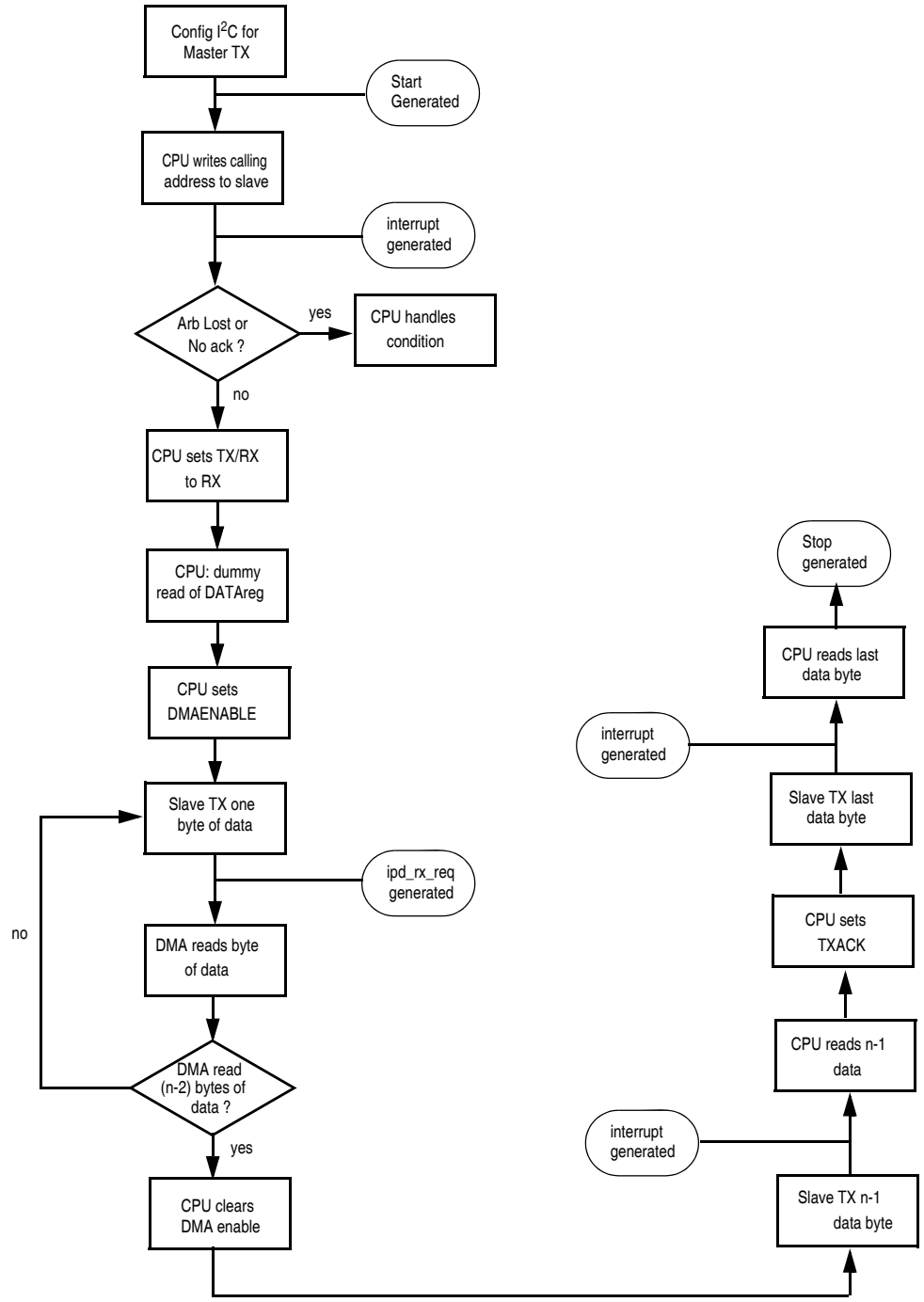


Figure 29-17. Flow-Chart of DMA Mode Master Receive

29.7.2.3 Exiting DMA Mode, System Requirement Considerations

As described above, the final transfers of both TX and RX transfers need to be handled via interrupt by the CPU. To change from DMA to interrupt driven transfers in the I^2C module, you have to disable the DMAEN bit in the IBCR register. The trigger to exit the DMA mode is that the programmed DMA Transfer Control Descriptor (TCD) has completed all its transfers to/from the I^2C module.

After the last DMA write (TX mode) to the I^2C the module will immediately start the next I^2C-bus transfer. The same is true for RX mode. After the DMA read from the IBDR register the module initiates the next I^2C-bus transfer. This results in 2 possible scenarios in the DMA mode exiting scheme.

1. Fast reaction

   The DMAEN bit is cleared before the next I^2C-bus transfer completes. In this case the module will raise an interrupt request to the CPU which can be serviced normally.

2. Slow reaction

   The DMAEN bit is cleared after the next I^2C-bus transfer has already completed. In this case the module will not raise an interrupt request to the CPU. Instead the TCF bit can be read to determine that the transfer completed and the module is ready for further transfer.

What is fast/slow reaction?:

The reaction time T_R for the system to disable DMAEN after the last DMA controller access to the I^2C is the time required for one byte transfer over the I^2C. For 'fast reaction' the disabling has to occur before the 9th bit of the data transfer which is the ACK bit. So the time available is 8 times the SCL period.

$$T_R = 8 \times T_{SCL} \qquad \textit{Eqn. 29-5}$$

In fast mode, with 400 kbit/s, T_{SCL} is 2.5 us, so T_R is 20us.

Depending on the system and DMA controller there are different possibilities for the deassertion of DMAEN. Three options are:

1. CPU intervention via Interrupt

   The DMA controller is programmed to signal an interrupt to the CPU which is then responsible for the deassertion of DMAEN. This scheme should be supported by most systems but it can result in a slow reaction time if other, higher priority interrupts interfer. Therfore the interrupt handling routine can become complicated as it has to check which of the two cases happened (check TCF bit) and act accordingly. In case of slow reaction you can force an interrupt for the I^2C in the interrupt controller to have the further transfer handled by the normal I^2C interrupt routine. Please note that the use of nested interrupts can still cause potential issues in this scenario, if you happen to stall the DMA interrupt between the deassertion and the DMAEN bit and the checking of the TCF bit.

2. DMA channel linking (if supported)

   The Transfer Control Descriptor in the DMA controller that performs the data transfer is linked to another channel that does a write to the I^2C IBCR register to disable the DMAEN bit. This is probably the fastest system solution, but it uses 2 DMA channels. Please note that you have to make sure on system level that no higher priority DMA requests occur between the two linked TCDs as those could again create a scenario of slow reaction.

3. DMA scatter/gather process (if supported)

   The Transfer Control Descriptor in the DMA controller that performs the data transfer has the scatter/gather feature activated. This feature will initiate a reload of another TCD from system RAM after the completion of the first TCD. The new TCD will have its start bit already set and immediately start the required write to the I^2C IBCR register to disable the DMAEN bit. This TCD also has scatter/gather activated and is programmed to reload the initial TCD upon completion,

bringing the system back into a 'ready-for-I^2C-transfer' state. The advantage over the two other solutions is that this neigher requires CPU intervention nor a 2nd DMA channel. This comes at the cost of 64 bytes RAM (2 TCDs), some system bus transfer overhead and a little increase in application code complexity. Please note that you have to make sure on system level that no higher priority DMA requests occur during the scatter/gather process, as those could again create a scenario of slow reaction.

Example latencies for a 32 MHz system with a full speed 32bit AHB bus and an I^2C connected via half speed IPI bus:

- Accessing the I^2C from the DMA controller via IPI bus typically requires 4 cycles (consecutive accesses to the I^2C could be faster):

$$4 \times T_{IPI} = 4 / 16\ MHz = 250\ ns \qquad \textit{Eqn. 29-6}$$

- Reloading a new TCD (8 x 32 bit) via AHB to the DMA controller (scatter/gather process):

$$8 \times T_{AHB} = 8 / 32\ MHz = 250\ ns \qquad \textit{Eqn. 29-7}$$

Conclusion:

With the DMA scatter/gather process the required IBCR access can be done in 0.5 us, leaving a large margin of 19.5 us for additional system delays. In this way the slow reaction case can be prevented. The system user needs to decide which usage model suits his overall requirements best.

Chapter 30
Deserial Serial Peripheral Interface (DSPI)

30.1 Introduction to the DSPI on MAC7200

The De-serialization Serial Peripheral Interface (DSPI) block provides a synchronous serial bus for communication between an MCU and an external peripheral device. On the MAC72xx , the DSPIs operate only in serial peripheral interface (SPI) configuration, where the DSPI operates as a basic SPI or as a queued SPI through the use of internal FIFOs.

30.2 DSPI Features

- Full Duplex, Synchronous Transfers.
- Master or Slave Operation.
- Programmable Master Bit Rates.
- Programmable Clock Polarity and Phase.
- End-of-Transmission Interrupt Flag.
- Programmable transfer Baud rate.
- Programmable data frames from 4-bits to 16-bits.
- Up to 6 chip select lines enable 64 external devices to be selected using external muxing from a single DSPI.
- 8 Clock and Transfer Attributes registers.
- Chip Select Strobe available as alternate function on one of the Chip Select pins for de-glitching.
- Two dedicated DMA request lines on each peripheral for receive and transmit data.
- FIFO for buffering up to 4 transfers on the transmit and receive side.
- Queueing operation possible through use of the DMA controllers channels.
- General purpose I/O functionality on pins when not used for SPI

On the MAC72xx, the de-serialization feature of the DSPI is not available.

30.3 DSPI Protocol

The SPI bus is a 4-wire serial communications interface used for medium speed (less than 100 MB/s). The interface consists of the following 4 pins:

- **SCK** (Serial Data Clock): Data is shifted/latched on the rising or falling edge of SCK, depending on value of the CPOL control bit. This signal is bussed (i.e., one signal for all Slaves).
- **MOSI** (Master Output/Slave Input): Data is transmitted out of this pin if the chip is a Master and into this pin if the chip is a Slave. This signal is bused (i.e., 1 signal for all Slaves).

- **MISO** (Master Input/Slave Output): Data is received into this pin if the chip is a Master and transmitted out of this pin if the chip is a Slave. This signal is bussed (i.e., 1 signal for all Slaves).
- **CS** (Chip Select - active low): Point to point connection between a Master and a Slave that is used to select a Slave for a particular transfer. On the slave side, it is sometimes referred to as a Slave Select.

A typical SPI bus consists of a single Master and one or more slaves. Selection between Master and Slave may be dynamic (i.e., software selectable), resulting in a (possibly) dynamic network topography. The Master on the bus is always responsible for driving the clock (SCK), with the possibility of inserting wait states by holding the clock low. Data transfers over a SPI bus are always done in 8-bit chunks (bytes), with a single transfer consisting of 1 or more bytes. Data on the MOSI and MISO lines is always driven and sampled with respect to SCK, with one of a defined set of 4 clock phase and polarities.

One of these combinations is selected (either as a static configuration or on a transfer by transfer basis) using the two control bits CPOL and CPHA. These combinations are shown in Table 30-1.

Table 30-1. CPOL and CPHA Control Bits

CPOL	CPHA	Transfer Characteristics
o	o	SCK rising edge is used. SCK transitions in the middle of the bit timing.
0	1	SCK rising edge is used. SCK transitions at the beginning of the bit timing.
1	0	SCK falling edge is used. SCK transitions in the middle of the bit timing.
1	1	SCK falling edge is used. SCK transitions at the beginning of the bit timing.

The SPI protocol itself does not specify the content of the data, only the physical transmission of the bits. Thus, there are no defined addressing or data packet schemes, leaving the software to implement the upper layers of the SPI protocol stack.

30.4 DSPI Implementation

The DSPI module has several different configuration options. On the MAC72xx device, the DSPI modules have the following configuration:

- MDIS Reset Value: The reset value of the **MDIS** bit is 1, meaning that the DSPI is in a disabled state after reset, and must be explicitly enabled before it can be used.
- Number of CTAR Registers: The number of Clock and Transfer Attributes Registers (CTAR) on the MAC72xx is configured to 6 per DSPI.
- Transmit FIFO Size: The size of the transmit FIFOs on all DSPIs on the MAC72xx is set to a depth of 4 per DSPI.
- Receive FIFO Size: The size of the receive FIFOs on all DSPIs on the MAC72xx is set to a depth of 4 per DSPI.

- Module Base Address: The address offset between the beginning of the DSPI module and the first DSPI register (DSPI_MCR), referred to as **DSPI_BASE**, is set to $0000 for all DSPIs on the MAC72xx.
- The MAC7200 family provides the DSPI with a range of Chip Select lines depending on the device. A maximum of six peripheral chip select lines can be provided from each of the on-board DSPIs, offering selection of up to 64 external devices and transfer configurations. One of the Chip Selects (CS[5]) can be used to provide a Chip Select Strobe in order to eliminate decoding glitches generated when the Chip Selects change. This allows glitch free selection of up to 32 external devices.

30.5 DSPI External Pins

Table 30-2. DSPI External Pins

SCK	Serial Clock
MOSI	Master Out/Slave In
MISO	Master In/Slave Out
$\overline{\text{CS0}}$/$\overline{\text{SS}}$	Chip Select 0/Slave Select
$\overline{\text{CS}}$[4:1]	Chip Select
$\overline{\text{CS5}}$/$\overline{\text{PCSS}}$	Chip Select/Chip Select Strobe
$\overline{\text{CS}}$[7:6]	Chip Select

Note that all chip select pins may not be available for all DSPIs on all device/package combinations.

30.6 DSPI Bus Aborts

The DSPI module supports Peripheral Bus bus aborts, and enforces the following memory map:

Table 30-3. DSPI Bus Aborts

Abort	Allowed
	$0000-$00cf
$00d0-$3fff	

Supervisor Access: Unused

30.7 DSPI Differences from MAC71xx

- Added CS6/CS7 (8 chip-selects total). Not all chip selects are bonded out on all device and/or packages.
- Number of CTARs (6) will not change
- Added Double Baud Rate functionality
  - Added DBR bit to each CTAR register
- Fixed MUCts01474: Limitations of continuous chip select mode

- Fixed MUCts01650: Write on bus abort still writes register
- DSPI runs directly from system (fast) clock (not programmable) instead of from peripheral (slow) clock
- Continuous chip select definition updated in the Block Guide (v0.26)

30.8 DSPI Application Usage

30.8.1 Enabling the DSPI

Before the DSPI can be used, it must be explicitly enabled by clearing the **MDIS** bit.

30.8.2 Baud Rate Calculation

$$\text{BaudRate} = \frac{\text{fdspi}}{\text{PBR}} \times \frac{\text{DBR} + 1}{\text{BR}} \qquad \textit{Eqn. 30-1}$$

Given the above Baud Rate equation, and the limitations of the f_{sys}, PBR, DBR and BR settings, the maximum baud rate is as follows:

Table 30-4. DSPI Maximum Baud Rate Parameters

Parameter	MAC71x1	MAC72xx
f_{sys}	< 50MHz	< 70MHz
f_{dspi}	< 25MHz	< 70MHz
PBR	2...7	2...7
DBR	DBR bit not available	0,1
BR	2...32768	2...32768
Maximum Baud Rate (DBR=0)	$f_{dspi}/4$ = 6.25 Mb/s	$f_{dspi}/4$ = 17.5 Mb/s
Maximum Baud Rate (DBR=1)	DBR bit not available	$f_{dspi}/2$ = 35 Mb/s
Min. system clock frequency for 8 Mb/s (DBR=0)	64 MHz	32 MHz
Min. system clock frequency for 8 Mb/s (DBR=1)	DBR bit not available	16 MHz

NOTE

The above Baud Rate calculations apply to Master Mode only.

On the MAC71x1, $f_{DSPI} = f_{SYS}/2$. On the MAC72xx $f_{DSPI} = f_{SYS}$

Table 30-5. DSPI SCK Duty Cycle Calculation

DBR	CPHA	PBR	SCK Duty Cycle
0	any	any	50/50
1	0	00	50/50

Table 30-5. DSPI SCK Duty Cycle Calculation

DBR	CPHA	PBR	SCK Duty Cycle
1	0	01	33/66
1	0	10	40/60
1	0	11	43/57
1	1	00	50/50
1	1	01	66/33
1	1	10	60/40
1	1	11	57/43

30.9 DSPI Module

30.9.1 Block Diagram

Figure 30-1 is a block diagram of the Deserial Serial Peripheral Interface (DSPI) block on the MAC72xx.

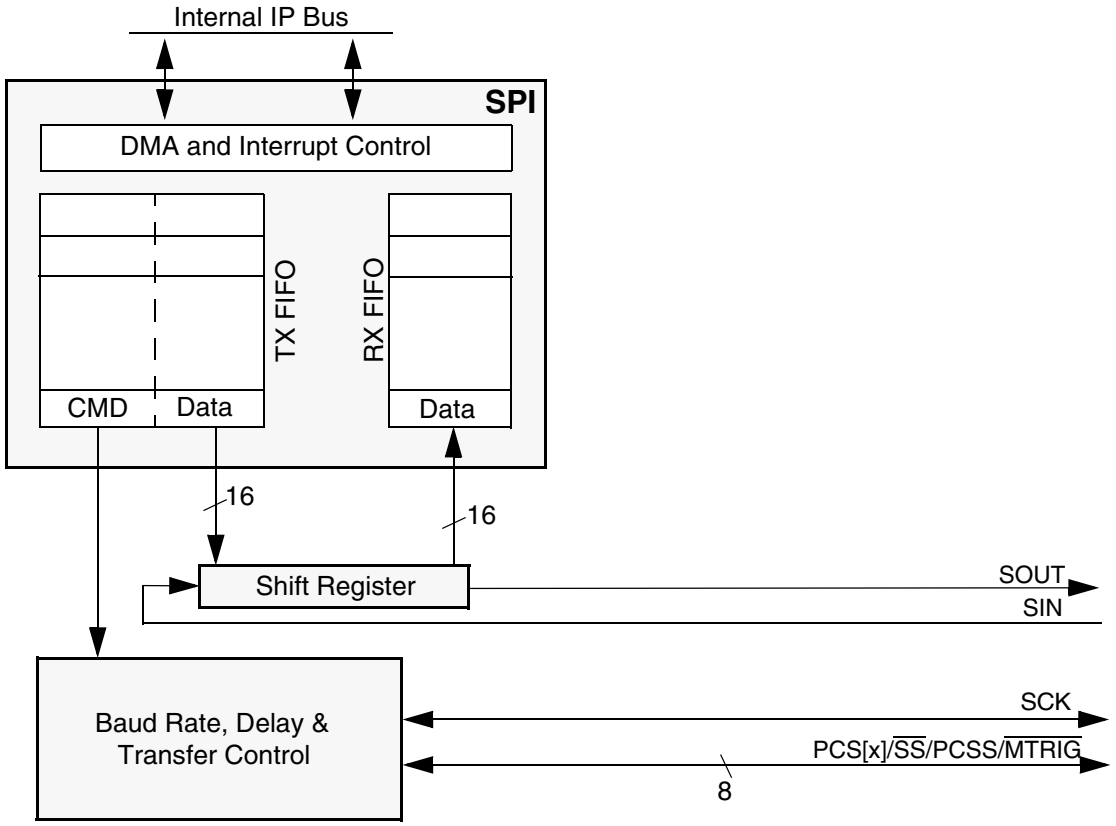


Figure 30-1. DSPI Block Diagram

30.9.2 Overview

The Deserial Serial Peripheral Interface (DSPI) block provides a synchronous serial bus for communication between an MCU and an external peripheral device. The DSPI supports pin count reduction through serialization and deserialization of parallel signals transmitted over the SPI serial link. The DSPIs implemented on MAC72xx have one configuration:

- Serial Peripheral interface (SPI) Configuration where the DSPI operates as a basic SPI or as a queued SPI through the use of internal FIFOs.

For queued operations the SPI queues reside in system RAM which is external to the DSPI. Data transfers between the queues and the DSPI FIFOs are accomplished through the use of a DMA controller or through host software. Figure 30-2 shows a DSPI with external queues in system RAM.

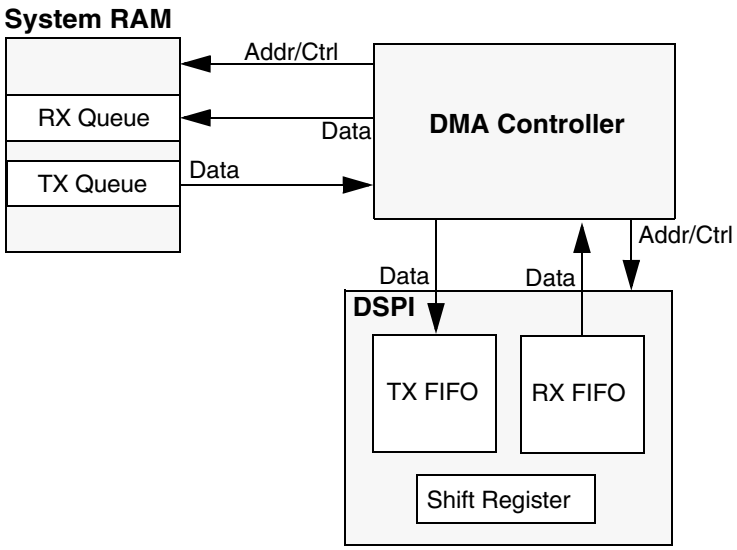


Figure 30-2. DSPI with Queues and DMA

30.9.3 DSPI Configuration

The DSPI block used on MAC72xx has one distinct serial transmission configuration: SPI.

30.9.3.1 SPI Configuration

The SPI Configuration allows the DSPI to send and receive serial data. This configuration allows the DSPI to operate as a basic SPI block with the FIFOs providing support for external queue operation. Data to be transmitted and data received reside in separate FIFOs. The FIFOs can be popped and pushed by host software or by a DMA controller.

30.9.4 Modes of Operation

The DSPI has five modes of operation that can be grouped into two categories: block-specific modes, i.e., Master, Slave, and Module Disable Modes; and MCU-specific modes, i.e., External Stop and Debug Modes.

The block-specific modes are entered by host software writing to a register. The MCU-specific modes are controlled by signals external to the DSPI. The MCU-specific modes are modes that the entire MCU may enter, in parallel to the DSPI being in one of its block-specific modes.

30.9.4.1 Master Mode

Master Mode allows the DSPI to initiate and control serial communication. In this mode, the SCK signal and the PCS[*x*] signals are controlled by the DSPI and configured as outputs.

30.9.4.2 Slave Mode

The Slave Mode allows the DSPI to communicate with SPI bus masters. In this mode the DSPI responds to externally controlled serial transfers. The DSPI cannot control serial transfers in Slave Mode. In this mode, the SCK signal and the PCS[0]/$\overline{SS}$ signal are configured as inputs and provided by a bus master.

30.9.4.3 Module Disable Mode

The Module Disable Mode is used for MCU power management. The clock to the non-memory mapped logic in the DSPI can be stopped while in the Module Disable Mode. Logic external to the DSPI is needed to fully implement the Module Disable Mode.

30.9.4.4 External Stop Mode

The External Stop Mode is used for MCU power management. When a request is made to enter External Stop Mode, the DSPI block acknowledges the request and completes the transfer in progress. When the DSPI reaches the frame boundary it signals that the system clocks to the DSPI block may be shut off.

30.9.4.5 Debug Mode

The Debug Mode is used for system development and debugging. If the SoC enters Debug Mode while the FRZ bit in the DSPI_MCR is set, the DSPI stops all serial transfers. If the SoC enters Debug Mode while the FRZ bit is negated, the DSPI behavior is unaffected and remains dictated by the block-specific mode and configuration of the DSPI.

30.10 External Signal Description

30.10.1 Overview

Table 30-6 lists the signals that may connect off chip depending on SoC implementation.

Table 30-6. Signal Properties

Name	I/O Type	Function	
		Master Mode	Slave Mode
PCS[0]/$\overline{SS}$	Output / Input	Peripheral Chip Select 0	Slave Select
PCS[1] - PCS[3]	Output	Peripheral Chip Select 1 - 3	Unused

Table 30-6. Signal Properties

Name	I/O Type	Function	
		Master Mode	Slave Mode
PCS[4]/$\overline{\text{MTRIG}}$	Output	Peripheral Chip Select 4	Master Trigger
PCS[5]/$\overline{\text{PCSS}}$	Output	Peripheral Chip Select 5 / Peripheral Chip Select Strobe	Unused
PCS[6] - PCS[7]	Output	Peripheral Chip Select 6- 7	Unused
SIN	Input	Serial Data In	Serial Data In
SOUT	Output	Serial Data Out	Serial Data Out
SCK	Output / Input	Serial Clock (output)	Serial Clock (input)

30.10.2 Detailed Signal Description

30.10.2.1 PCS[0]/$\overline{\text{SS}}$ — Peripheral Chip Select/Slave Select

In Master Mode, the PCS[0] signal is a Peripheral Chip Select output that selects which slave device the current transmission is intended for.

In Slave Mode, the $\overline{\text{SS}}$ signal is a Slave Select input signal that allows a SPI master to select the DSPI as the target for transmission.

30.10.2.2 PCS[1] - PCS[3] — Peripheral Chip Selects 1 - 3

PCS[1] - PCS[3] are Peripheral Chip Select output signals in Master Mode. In Slave Mode these signals are not used.

30.10.2.3 PCS[4]/$\overline{\text{MTRIG}}$ — Peripheral Chip Select 4/Master Trigger

In Master Mode, PCS[4] is a Peripheral Chip Select output signal.

In Slave Mode, $\overline{\text{MTRIG}}$ is an output trigger signal that indicates that a change in data to be serialized has occurred. The $\overline{\text{MTRIG}}$ pulse is four system clock cycles in duration. If the DSPI is in Slave Mode and the MTO is disabled, the PCS[4]/$\overline{\text{MTRIG}}$ signal is unused.

30.10.2.4 PCS[5]/$\overline{\text{PCSS}}$ — Peripheral Chip Select 5/Peripheral Chip Select Strobe

PCS[5] is a Peripheral Chip Select output signal. When the DSPI is in Master Mode and PCSSE bit in the DSPI_MCR is negated, this signal is used to select which slave device the current transfer is intended for.

$\overline{\text{PCSS}}$ provides a strobe signal that can be used with an external demultiplexer for deglitching of the PCS signals. When the DSPI is in Master Mode and the PCSSE bit in the DSPI_MCR is set, the $\overline{\text{PCSS}}$ provides the appropriate timing for the decoding of the PCS[0] - PCS[4] and PCS[6] - PCS[7] signals which prevents glitches from occurring.

This signal is not used in Slave Mode.

30.10.2.5 PCS[6] - PCS[7] — Peripheral Chip Selects 6- 7

PCS[6] - PCS[7] are Peripheral Chip Select output signals in Master Mode. In Slave Mode these signals are not used.

30.10.2.6 SIN — Serial Input

SIN is a serial data input signal.

30.10.2.7 SOUT — Serial Output

SOUT is a serial data output signal.

30.10.2.8 SCK — Serial Clock

SCK is a serial communication clock signal. In Master Mode, the DSPI generates the SCK. In Slave Mode, SCK is an input from an external bus master.

30.11 Memory Map and Register Definition

30.11.1 Memory Map

Table 30-7 shows the DSPI memory map.

Table 30-7. DSPI Memory Map

Address	Register Name
DSPI_BASE	DSPI Module Configuration Register (DSPI_MCR)
DSPI_BASE+0x4	Reserved
DSPI_BASE+0x8	DSPI Transfer Count Register (DSPI_TCR)
DSPI_BASE+0xC– DSPI_BASE+0x28	DSPI Clock and Transfer Attributes Register 0 (DSPI_CTAR0) - DSPI Clock and Transfer Attributes Register 7 (DSPI_CTAR7)[1]
DSPI_BASE+0x2C	DSPI Status Register (DSPI_SR)
DSPI_BASE+0x30	DSPI DMA/Interrupt Request Select and Enable Register (DSPI_RSER)
FIFO Registers	
DSPI_BASE+0x34	DSPI Push TX FIFO Register (DSPI_PUSHR)
DSPI_BASE+0x38	DSPI Pop RX FIFO Register (DSPI_POPR)
DSPI_BASE+0x3C - DSPI_BASE+0x48	DSPI Transmit FIFO Register 0 (DSPI_TXFR0) - DSPI Transmit FIFO Register 3 (DSPI_TXFR3)[2]
DSPI_BASE+0x4C - DSPI_BASE+0x58	DSPI Receive FIFO Register 0 (DSPI_RXFR0) - DSPI Receive FIFO Register 3 (DSPI_RXFR3)[2]

1. The number of CTAR registers is parameterized in RTL.
2. FIFO Depths are parameterized in RTL.

30.11.2 Register Descriptions

30.11.2.1 DSPI Module Configuration Register (DSPI_MCR)

The DSPI_MCR contains bits which configure various attributes associated with DSPI operation. The HALT and MDIS bits can be changed at any time but will only take effect on the next frame boundary. Only the HALT and MDIS bits in the DSPI_MCR may be changed while the DSPI is in the Running state.

Address: DSPI_BASE Access:

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	MSTR	CONT_SCKE	DCONF		FRZ	MTFE	PCSSE	ROOE	PCSIS7	PCSIS6	PCSIS5	PCSIS4	PCSIS3	PCSIS2	PCSIS1	PCSIS0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	DOZE	MDIS	DIS_TXF	DIS_RXF	CLR_TXF	CLR_RXF	SMPL_PT		0	0	0	0	0	0	0	HALT
W																
Reset	0	0[1]	0	0	0	0	0	0	0	0	0	0	0	0	0	1

[1] MDIS reset value is paramaterized in RTL.

Figure 30-3. DSPI Module Configuration Register (DSPI_MCR)

Table 30-8. DSPI_MCR Field Descriptions

Field	Description
31 MSTR	Master/Slave Mode Select. The MSTR bit configures the DSPI for either Master Mode or Slave Mode. 0 DSPI is in Slave Mode 1 DSPI is in Master Mode
30 CONT_SCKE	Continuous SCK Enable. The CONT_SCKE bit enables the Serial Communication Clock (SCK) to run continuously. See Section 30.12.6, “Continuous Serial Communications Clock,” for details. 0 Continuous SCK disabled 1 Continuous SCK enabled
29–28 DCONF[1:0]	DSPI Configuration. The DCONF field selects the configuration of the DSPI. As only the SPI configuration is implemented on the MAC72xx, the DCONF bits must always be set to 00.. **DCONF** / **DSPI Configuration** 00 / SPI 01 / Reserved 10 / Reserved 11 / Reserved

Table 30-8. DSPI_MCR Field Descriptions (Continued)

Field	Description
27 FRZ	Freeze. The FRZ bit enables the DSPI transfers to be stopped on the next frame boundary when the SoC enters Debug Mode. 0 Do not halt serial transfers1Halt serial transfers
26 MTFE	Modified Timing Format Enable. The MTFE bit enables a modified transfer format to be used. See Section 30.12.5.4, "Modified SPI Transfer Format (MTFE = 1, CPHA = 1)," for more information. 0 Modified SPI transfer format disabled 1 Modified SPI transfer format enabled
25 PCSSE	Peripheral Chip Select Strobe Enable. The PCSSE bit enables the PCS[5]/$\overline{\text{PCSS}}$ to operate as an PCS Strobe output signal. See Section 30.12.4.5, "Peripheral Chip Select Strobe Enable (PCSS)," for more information. 0 PCS[5]/$\overline{\text{PCSS}}$ is used as the Peripheral Chip Select[5] signal 1 PCS[5]/$\overline{\text{PCSS}}$ is used as an active-low PCS Strobe signal
24 ROOE	Receive FIFO Overflow Overwrite Enable. The ROOE bit enables an RX FIFO overflow condition to either ignore the incoming serial data or to overwrite existing data. If the RX FIFO is full and new data is received, the data from the transfer that generated the overflow is either ignored or shifted in to the shift register. If the ROOE bit is asserted, the incoming data is shifted in to the shift register. If the ROOE bit is negated, the incoming data is ignored. See Section 30.12.7.6, "Receive FIFO Overflow Interrupt Request," for more information. 0 Incoming data is ignored 1 Incoming data is shifted in to the shift register
23–16 PCSIS*x*	Peripheral Chip Select Inactive State. The PCSIS bit determines the inactive state of the PCS*x* signal. 0 The inactive state of PCS*x* is low 1 The inactive state of PCS*x* is high
15 DOZE	Doze Enable. The DOZE bit provides support for externally controlled Doze Mode power-saving mechanism. See Section 30.12.8, "Power Saving Features," for details.
14 MDIS	Module Disable. The MDIS bit allows the clock to be stopped to the non-memory mapped logic in the DSPI effectively putting the DSPI in a software controlled power-saving state. See Section 30.12.8, "Power Saving Features," for more information. The reset value of the MDIS bit is parameterized, with a default reset value of '0'. 0 Enable DSPI clocks. 1 Allow external logic to disable DSPI clocks.
13 DIS_TXF	Disable Transmit FIFO. The DIS_TXF bit provides a mechanism to disable the TX FIFO. When the TX FIFO is disabled, the transmit part of the DSPI operates as a simplified double-buffered SPI. See Section 30.12.3.3, "FIFO Disable Operation," for details. 0 TX FIFO is enabled 1 TX FIFO is disabled
12 DIS_RXF	Disable Receive FIFO. The DIS_RXF bit provides a mechanism to disable the RX FIFO. When the RX FIFO is disabled, the receive part of the DSPI operates as a simplified double-buffered SPI. See Section 30.12.3.3, "FIFO Disable Operation," for details. 0 RX FIFO is enabled 1 RX FIFO is disabled
11 CLR_TXF	Clear TX FIFO. CLR_TXF is used to flush the TX FIFO. Writing a '1' to CLR_TXF clears the TX FIFO Counter. The CLR_TXF bit is always read as zero. 0 Do not clear the TX FIFO Counter 1 Clear the TX FIFO Counter

Table 30-8. DSPI_MCR Field Descriptions (Continued)

Field	Description
10 CLR_RXF	Clear RX FIFO. CLR_RXF is used to flush the RX FIFO. Writing a '1' to CLR_RXF clears the RX Counter. The CLR_RXF bit is always read as zero. 0 Do not clear the RX FIFO Counter 1 Clear the RX FIFO Counter
9–8 SMPL_PT	SMPL_PT — Sample Point. SMPL_PT allows the host software to select when the DSPI Master samples SIN in Modified Transfer Format. Figure 30-18 shows where the Master can sample the SIN pin. The table below lists the various delayed sample points.
7–1	Reserved, should be cleared.
0 HALT	Halt. The HALT bit provides a mechanism by software to start and stop DSPI transfers. See Section 30.12.2, "Start and Stop of DSPI Transfers," for details on the operation of this bit. 0 Start transfers 1 Stop transfers

SMPL_PT	Number of system clock cycles between odd-numbered edge of SCK and sampling of SIN.
00	0
01	1
10	2
11	Reserved

30.11.2.2 DSPI Transfer Count Register (DSPI_TCR)

The DSPI_TCR contains a counter that indicates the number of SPI transfers made. The transfer counter is intended to assist in queue management. The user must not write to the DSPI_TCR while the DSPI is in the Running state.

Address: DSPI_BASE + 0x8 Access:

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	SPI_TCNT															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-4. DSPI Transfer Count Register (DSPI_TCR)

Table 30-9. DSPI_TCR Field Descriptions

Field	Description
31–16 SPI_TCNT[15:0]	SPI Transfer Counter. SPI_TCNT is used to keep track of the number of SPI transfers made. The SPI_TCNT field counts the number of SPI transfers the DSPI makes. The SPI_TCNT field is incremented every time the last bit of a SPI frame is transmitted. A value written to SPI_TCNT presets the counter to that value. SPI_TCNT is reset to zero at the beginning of the frame when the CTCNT field is set in the executing SPI command. The Transfer Counter 'wraps around' i.e. incrementing the counter past 65535 resets the counter to zero.
15–0	Reserved, should be cleared.

30.11.2.3 DSPI Clock and Transfer Attributes Registers 0–7 (DSPI_CTAR0–DSPI_CTAR7)

The DSPI_CTAR registers are used to define different transfer attribute configurations. SPI transfers select which one of the DSPI_CTARs to get their transfer attributes from. The number of CTAR registers is parameterized in the RTL and can be from two to eight registers. The user must not write to the DSPI_CTAR registers while the DSPI is in the Running state.

In Master Mode, the DSPI_CTAR0 - DSPI_CTAR7 registers define combinations of transfer attributes such as frame size, clock phase and polarity, data bit ordering, baud rate, and various delays. In Slave Mode, a subset of the bitfields in the DSPI_CTAR0 and DSPI_CTAR1 registers are used to set the slave transfer attributes. See the individual bit descriptions for details on which bits are used in Slave Modes.

When the DSPI is configured as an SPI Master, the CTAS field in the command portion of the TX FIFO entry selects which of the DSPI_CTAR register is used. When the DSPI is configured as an SPI bus Slave, the DSPI_CTAR0 register is used.

Address: DSPI_BASE + 0xC–DSPI_BASE + 0x28 Access:

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	DBR	FMSZ				CPOL	CPHA	LSBFE	PCSSCK		PASC		PDT		PBR	
Reset	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	CSSCK				ASC				DT				BR			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-5. DSPI Clock and Transfer Attributes Register 0–7 (DSPI_CTAR0–DSPI_CTAR7)

Table 30-10. DSPI_CTAR*n* Field Descriptions

Field	Descriptions
31 DBR	Double Baud Rate. The DBR bit doubles the effective baud rate of the Serial Communications Clock (SCK). This field is only used in Master Mode. It effectively halves the Baud Rate division ratio supporting faster frequencies and odd division ratios for the Serial Communications Clock (SCK). When the DBR bit is set, the duty cycle of the Serial Communications Clock (SCK) depends on the value in the Baud Rate Prescaler and the Clock Phase bit as listed in Table 30-11. See the BR[3:0] field description for details on how to compute the baud rate. If the overall baud rate is divide by two or divide by three of the system clock then neither the Continuous SCK Enable or the Modified Timing Format Enable bits should be set. 0 The baud rate is computed normally with a 50/50 duty cycle 1 The baud rate is doubled with the duty cycle depending on the Baud Rate Prescaler
30–27 FMSZ[3:0]	Frame Size. The FMSZ field selects the number of bits transferred per frame. The FMSZ field is used in Master Mode and Slave Mode. Table 30-12 lists the frame sizes..
26 CPOL	Clock Polarity. The CPOL bit selects the inactive state of the Serial Communications Clock (SCK). This bit is used in both Master and Slave Mode. For successful communication between serial devices, the devices must have identical clock polarities. When the Continuous Selection Format is selected, switching between clock polarities without stopping the DSPI can cause errors in the transfer due to the peripheral device interpreting the switch of clock polarity as a valid clock edge. 0 The inactive state value of SCK is low 1 The inactive state value of SCK is high
25 CPHA	Clock Phase. The CPHA bit selects which edge of SCK causes data to change and which edge causes data to be captured. This bit is used in both Master and Slave Mode. For successful communication between serial devices, the devices must have identical clock phase settings. 0 Data is captured on the leading edge of SCK and changed on the following edge 1 Data is changed on the leading edge of SCK and captured on the following edge
24 LSBFE	LSB First. The LSBFE bit selects if the LSB or MSB of the frame is transferred first. This bit is only used in Master Mode. 0 Data is transferred MSB first 1 Data is transferred LSB first
23–22 PCSSCK[1:0]	PCS to SCK Delay Prescaler. The PCSSCK field selects the prescaler value for the delay between assertion of PCS and the first edge of the SCK. This field is only used in Master Mode. The table below lists the prescaler values. See the CSSCK[3:0] field description for details on how to compute the PCS to SCK Delay.

PCSSCK	PCS to SCK Delay Prescaler Value
00	1
01	3
10	5
11	7

Table 30-10. DSPI_CTAR*n* Field Descriptions (Continued)

<table>
<tr><th>Field</th><th>Descriptions</th></tr>
<tr><td>21–20
PASC[1:0]</td><td>After SCK Delay Prescaler. The PASC field selects the prescaler value for the delay between the last edge of SCK and the negation of PCS. This field is only used in Master Mode. The table below lists the prescaler values. See the ASC[3:0] field description for details on how to compute the After SCK Delay.
<table>
<tr><th>PASC</th><th>After SCK Delay Prescaler Value</th></tr>
<tr><td>00</td><td>1</td></tr>
<tr><td>01</td><td>3</td></tr>
<tr><td>10</td><td>5</td></tr>
<tr><td>11</td><td>7</td></tr>
</table></td></tr>
<tr><td>19–18
PDT[1:0]</td><td>Delay after Transfer Prescaler. The PDT field selects the prescaler value for the delay between the negation of the PCS signal at the end of a frame and the assertion of PCS at the beginning of the next frame. The PDT field is only used in Master Mode. The table below lists the prescaler values. See the DT[3:0] field description for details on how to compute the Delay after Transfer.
<table>
<tr><th>PDT</th><th>Delay after Transfer Prescaler Value</th></tr>
<tr><td>00</td><td>1</td></tr>
<tr><td>01</td><td>3</td></tr>
<tr><td>10</td><td>5</td></tr>
<tr><td>11</td><td>7</td></tr>
</table></td></tr>
<tr><td>17–16
PBR[1:0]</td><td>Baud Rate Prescal. The PBR field selects the prescaler value for the baud rate. This field is only used in Master Mode. The Baud Rate is the frequency of the Serial Communications Clock (SCK). The system clock is divided by the prescaler value before the baud rate selection takes place. The Baud Rate Prescaler values are listed in the table below. See the BR[3:0] field description for details on how to compute the baud rate.
<table>
<tr><th>PBR</th><th>Baud Rate Prescaler Value</th></tr>
<tr><td>00</td><td>2</td></tr>
<tr><td>01</td><td>3</td></tr>
<tr><td>10</td><td>5</td></tr>
<tr><td>11</td><td>7</td></tr>
</table></td></tr>
<tr><td>15–12
CSSCK[3:0]</td><td>PCS to SCK Delay Scaler. The CSSCK field selects the scaler value for the PCS to SCK delay. This field is only used in Master Mode. The PCS to SCK Delay is the delay between the assertion of PCS and the first edge of the SCK. Table 30-13 list the scaler values.The PCS to SCK Delay is a multiple of the system clock period and it is computed according to the following equation:
$$t_{CSC} = \frac{1}{f_{SYS}} \times PCSSCK \times CSSCK$$ Eqn. 30-2
See Section 30.12.4.2, “PCS to SCK Delay (t_{CSC}),” for more details.</td></tr>
</table>

Table 30-10. DSPI_CTAR*n* Field Descriptions (Continued)

Field	Descriptions
11–8 ASC[3:0]	After SCK Delay Scaler. The ASC field selects the scaler value for the After SCK Delay. This field is only used in Master Mode. The After SCK Delay is the delay between the last edge of SCK and the negation of PCS. Table 30-14 list the scaler values.The After SCK Delay is a multiple of the system clock period, and it is computed according to the following equation: $t_{ASC} = \frac{1}{f_{SYS}} \times PASC \times ASC$ ***Eqn. 30-3*** See Section 30.12.4.3, "After SCK Delay (t_{ASC})," for more details.
7–4 DT[3:0]	Delay after Transfer Scaler. The DT field selects the Delay after Transfer Scaler. This field is only used in Master Mode. The Delay after Transfer is the time between the negation of the PCS signal at the end of a frame and the assertion of PCS at the beginning of the next frame. Table 30-15 lists the scaler values.. In the Continuous Serial Communications Clock operation the DT value is fixed to one TSCK. The Delay after Transfer is a multiple of the system clock period and it is computed according to the following equation: $t_{DT} = \frac{1}{f_{SYS}} \times PDT \times DT$ ***Eqn. 30-4*** See Section 30.12.4.4, "Delay after Transfer (tDT)," for more details.
3–0 BR[3:0]	Baud Rate Scaler. The BR field selects the scaler value for the baud rate. This field is only used in Master Mode. The pre-scaled system clock is divided by the Baud Rate Scaler to generate the frequency of the SCK. Table 30-16 lists the Baud Rate Scaler values.The baud rate is computed according to the following equation: $\text{SCK baud rate} = \frac{f_{SYS}}{PBR} \times \frac{1 + DBR}{BR}$ ***Eqn. 30-5*** See Section 30.12.4.1, "Baud Rate Generator," for more details.

Table 30-11. DSPI SCK Duty Cycle

DBR	CPHA	PBR	SCK Duty Cycle
0	any	any	50/50
1	0	00	50/50
1	0	01	33/66
1	0	10	40/60
1	0	11	43/57
1	1	00	50/50
1	1	01	66/33
1	1	10	60/40
1	1	11	57/43

Table 30-12. DSPI Transfer Frame Size

FMSZ	Framesize	FMSZ	Framesize
0000	Reserved	1000	9
0001	Reserved	1001	10
0010	Reserved	1010	11
0011	4	1011	12
0100	5	1100	13
0101	6	1101	14
0110	7	1110	15
0111	8	1111	16

Table 30-13. DSPI PCS to SCK Delay Scaler

CSSCK	PCS to SCK Delay Scaler Value	CSSCK	PCS to SCK Delay Scaler Value
0000	2	1000	512
0001	4	1001	1024
0010	8	1010	2048
0011	16	1011	4096
0100	32	1100	8192
0101	64	1101	16384
0110	128	1110	32768
0111	256	1111	65536

Table 30-14. DSPI After SCK Delay Scaler

ASC	After SCK Delay Scaler Value	ASC	After SCK Delay Scaler Value
0000	2	1000	512
0001	4	1001	1024
0010	8	1010	2048
0011	16	1011	4096
0100	32	1100	8192
0101	64	1101	16384
0110	128	1110	32768
0111	256	1111	65536

Table 30-15. DSPI Delay after Transfer Scaler

DT	Delay after Transfer Scaler Value	DT	Delay after Transfer Scaler Value
0000	2	1000	512
0001	4	1001	1024
0010	8	1010	2048
0011	16	1011	4096
0100	32	1100	8192
0101	64	1101	16384
0110	128	1110	32768
0111	256	1111	65536

Table 30-16. DSPI Baud Rate Scaler

BR	Baud Rate Scaler Value	BR	Baud Rate Scaler Value
0000	2	1000	256
0001	4	1001	512
0010	6	1010	1024
0011	8	1011	2048
0100	16	1100	4096
0101	32	1101	8192
0110	64	1110	16384
0111	128	1111	32768

30.11.2.4 DSPI Status Register (DSPI_SR)

The DSPI_SR contains status and flag bits. The bits reflect the status of the DSPI and indicate the occurrence of events that can generate interrupt or DMA requests. Software can clear flag bits in the DSPI_SR by writing a ‘1’ to it. Writing a ‘0’ to a flag bit has no effect.

Address: DSPI_BASE + 0x2C Access:

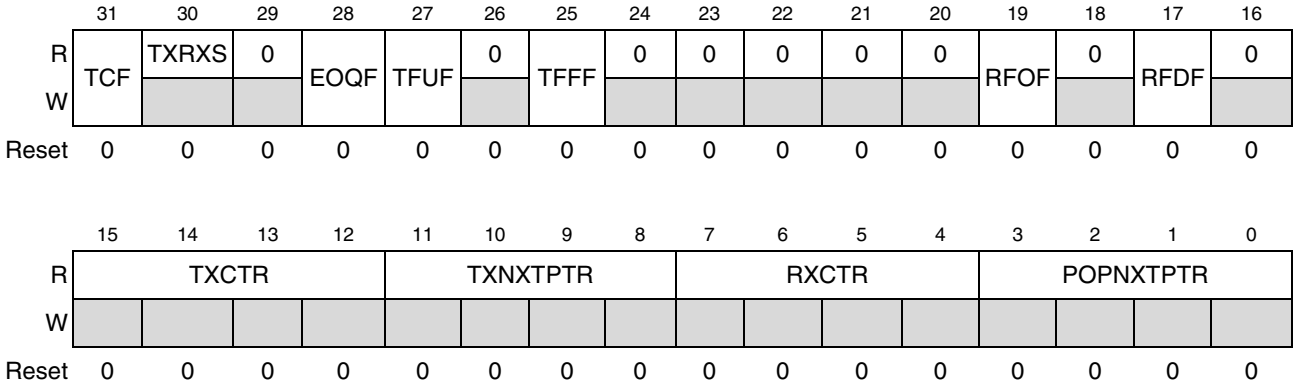


Figure 30-6. DSPI Status Register (DSPI_SR)

Table 30-17. DSPI_SR Field Descriptions

Field	Description
31 TCF	Transfer Complete Flag. The TCF bit indicates that all bits in a frame have been shifted out. The TCF bit is set at the end of the frame transfer. The TCF bit remains set until cleared by software. 0 Transfer not complete 1 Transfer complete
30 TXRXS	TX & RX Status. The TXRXS bit reflects the status of the DSPI. See Section 30.12.2, “Start and Stop of DSPI Transfers,” for information on how what causes this bit to be negated or asserted. 0 TX and RX operations are disabled (DSPI is in STOPPED state) 1 TX and RX operations are enabled (DSPI is in RUNNING state)
29	Reserved, should be cleared.
28 EOQF	End of Queue Flag. The EOQF bit indicates that transmission in progress is the last entry in a queue. The EOQF bit is set when TX FIFO entry has the EOQ bit set in the command halfword and the end of the transfer is reached. The EOQF bit remains set until cleared by software. When the EOQF bit is set, the TXRXS bit is automatically cleared. 0 EOQ is not set in the executing command 1 EOQ bit is set in the executing SPI command
27 TFUF	Transmit FIFO Underflow Flag. The TFUF bit indicates that an underflow condition in the TX FIFO has occurred. The transmit underflow condition is detected only for DSPI blocks operating in slave mode and SPI configuration. The TFUF bit is set when the TX FIFO of a DSPI operating in SPI slave mode is empty, and a transfer is initiated by an external SPI master. The TFUF bit remains set until cleared by software. 0 TX FIFO underflow has not occurred 1 TX FIFO underflow has occurred
26	Reserved, should be cleared.
25 TFFF	Transmit FIFO Fill Flag. The TFFF bit provides a method for the DSPI to request more entries to be added to the TX FIFO. The TFFF bit is set while the TX FIFO is not full. The TFFF bit can be cleared by host software or an acknowledgement from the DMA controller when the TX FIFO is full. 0 TX FIFO is full 1 TX FIFO is not full
24–20	Reserved, should be cleared.

Table 30-17. DSPI_SR Field Descriptions (Continued)

Field	Description
19 RFOF	Receive FIFO Overflow Flag. The RFOF bit indicates that an overflow condition in the RX FIFO has occurred. The bit is set when the RX FIFO and shift register are full and a transfer is initiated. The bit remains set until cleared by software. 0 RX FIFO overflow has not occurred 1 RX FIFO overflow has occurred
18	Reserved, should be cleared.
17 RFDF	Receive FIFO Drain Flag. The RFDF bit provides a method for the DSPI to request that entries be removed from the RX FIFO. The bit is set while the RX FIFO is not empty. The RFDF bit can be cleared by host software or an acknowledgement from the DMA controller when the RX FIFO is empty. 0 RX FIFO is empty 1 RX FIFO is not empty
16	Reserved, should be cleared.
15–12 TXCTR	TX FIFO Counter. The TXCTR field indicates the number of valid entries in the TX FIFO. The TXCTR is incremented every time the DSPI _PUSHR is written. The TXCTR is decremented every time a SPI command is executed and the SPI data is transferred to the shift register.
11–8 TXNXTPTR	Transmit Next Pointer. The TXNXTPTR field indicates which TX FIFO Entry will be transmitted during the next transfer. The TXNXTPTR field is updated every time SPI data is transferred from the TX FIFO to the shift register. See Section 30.12.7.4, “Transmit FIFO Underflow Interrupt Request,” for more details.
7–4 RXCTR	RX FIFO Counter. The RXCTR field indicates the number of entries in the RX FIFO. The RXCTR is decremented every time the DSPI _POPR is read. The RXCTR is incremented every time data is transferred from the shift register to the RX FIFO.
3–0 POPNXTPTR	Pop Next Pointer. The POPNXTPTR field contains a pointer to the RX FIFO entry that will be returned when the DSPI_POPR is read. The POPNXTPTR is updated when the DSPI_POPR is read. See Section 30.12.3.5, “Receive First In First Out (RX FIFO) Buffering Mechanism,” for more details.

30.11.2.5 DSPI DMA/Interrupt Request Select and Enable Register (DSPI_RSER)

The DSPI_RSER serves two purposes. It enables flag bits in the DSPI_SR to generate DMA requests or interrupt requests. The DSPI_RSER also selects the type of request to be generated. See the individual bit descriptions for information on the types of requests the bits support. The user must not write to the DSPI_RSER while the DSPI is in the Running state.

Address: DSPI_BASE + 0x30 Access:

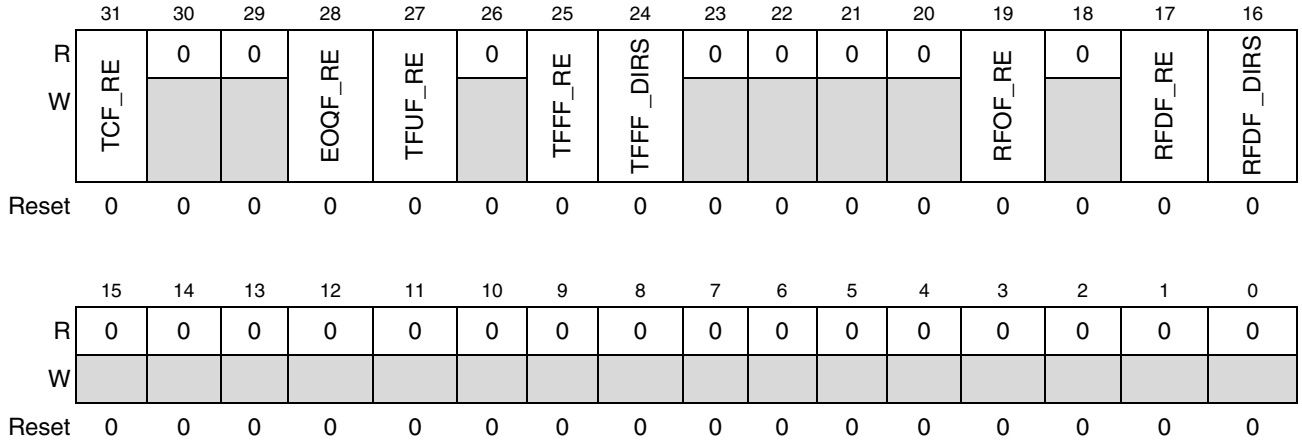


Figure 30-7. DSPI DMA/Interrupt Request Select and Enable Register (DSPI_RSER)

Table 30-18. DSPI_RSER Field Descriptions

Field	Description
31 TCF_RE	Transmission Complete Request Enable. The TCF_RE bit enables TCF flag in the DSPI_SR to generate an interrupt request. 0 TCF interrupt requests are disabled 1 TCF interrupt requests are enabled
30–29	Reserved, should be cleared.
28 EOQF_RE	DSPI Finished Request Enable. The EOQF_RE bit enables the EOQF flag in the DSPI_SR to generate an interrupt request. 0 EOQF interrupt requests are disabled 1 EOQF interrupt requests are enabled
27 TFUF_RE	Transmit FIFO Underflow Request Enable. The TFUF_RE bit enables the TFUF flag in the DSPI_SR to generate an interrupt request. 0 TFUF interrupt requests are disabled 1 TFUF interrupt requests are enabled
26	Reserved, should be cleared.
25 TFFF_RE	Transmit FIFO Fill Request Enable. The TFFF_RE bit enables the TFFF flag in the DSPI_SR to generate a request. The TFFF_DIRS bit selects between generating an interrupt request or a DMA requests. 0 TFFF interrupt requests or DMA requests are disabled 1 TFFF interrupt requests or DMA requests are enabled
24 TFFF_DIRS	Transmit FIFO Fill DMA or Interrupt Request Select. The TFFF_DIRS bit selects between generating a DMA request or an interrupt request. When the TFFF flag bit in the DSPI_SR is set, and the TFFF_RE bit in the DSPI_RSER register is set, this bit selects between generating an interrupt request or a DMA request. 0 Interrupt request will be generated 1 DMA request will be generated
23–20	Reserved, should be cleared.

Table 30-18. DSPI_RSER Field Descriptions (Continued)

Field	Description
19 RFOF_RE	Receive FIFO Overflow Request Enable. The RFOF_RE bit enables the RFOF flag in the DSPI_SR to generate an interrupt requests. 0 RFOF interrupt requests are disabled 1 RFOF interrupt requests are enabled
18	Reserved, should be cleared.
17 RFDF_RE	Receive FIFO Drain Request Enable. The RFDF_RE bit enables the RFDF flag in the DSPI_SR to generate a request. The RFDF_DIRS bit selects between generating an interrupt request or a DMA request. 0 RFDF interrupt requests or DMA requests are disabled 1 RFDF interrupt requests or DMA requests are enabled
16 RFDF_DIRS	Receive FIFO Drain DMA or Interrupt Request Select. The RFDF_DIRS bit selects between generating a DMA request or an interrupt request. When the RFDF flag bit in the DSPI_SR is set, and the RFDF_RE bit in the DSPI_RSER register is set, the RFDF_DIRS bit selects between generating an interrupt request or a DMA request. 0 Interrupt request will be generated 1 DMA request will be generated

30.11.2.6 DSPI PUSH TX FIFO Register (DSPI_PUSHR)

The DSPI_PUSHR provides a means to write to the TX FIFO. Data written to this register is transferred to the TX FIFO. See Section 30.12.3.4, “Transmit First In First Out (TX FIFO) Buffering Mechanism,” for more information. Eight or sixteen bit write accesses to the DSPI_PUSHR will transfer 32 bits to the TX FIFO.

Address: DSPI_BASE + 0x34 Access:

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	CONT	CTAS			EOQ	CTCNT	0	0	PCS7	PCS6	PCS5	PCS4	PCS3	PCS2	PCS1	PCS0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	TXDATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-8. DSPI PUSH TX FIFO Register (DSPI_PUSHR)

Table 30-19. DSPI_PUSHR Field Descriptions

<table>
<tr><th>Field</th><th>Descriptions</th></tr>
<tr><td>31
CONT</td><td>Continuous Peripheral Chip Select Enable. The CONT bit selects a Continuous Selection Format. The bit is used in SPI Master Mode. The bit enables the selected PCS signals to remain asserted between transfers. See Section 30.12.5.5, “Continuous Selection Format,” for more information.
0 Return Peripheral Chip Select signals to their inactive state between transfers
1 Keep Peripheral Chip Select signals asserted between transfers</td></tr>
<tr><td>30–28
CTAS[2:0]</td><td>Clock and Transfer Attributes Select. The CTAS field selects which of the DSPI_CTAR register is used to set the transfer attributes for the associated SPI frame. The field is only used in SPI Master Mode. In SPI Slave Mode DSPI_CTAR0 is used. The table below shows how the CTAS values map to the DSPI_CTAR registers. The number of DSPI_CTAR registers is implementation specific.
<table>
<tr><th>CTAS</th><th>Use Clock and Transfer Attributes from</th><th>CTAS</th><th>Use Clock and Transfer Attributes from</th></tr>
<tr><td>000</td><td>DSPI_CTAR0</td><td>100</td><td>DSPI_CTAR4</td></tr>
<tr><td>001</td><td>DSPI_CTAR1</td><td>101</td><td>DSPI_CTAR5</td></tr>
<tr><td>010</td><td>DSPI_CTAR2</td><td>110</td><td>DSPI_CTAR6</td></tr>
<tr><td>011</td><td>DSPI_CTAR3</td><td>111</td><td>DSPI_CTAR7</td></tr>
</table></td></tr>
<tr><td>27
EOQ</td><td>End Of Queue. The EOQ bit provides a means for host software to signal to the DSPI that the current SPI transfer is the last in a queue. At the end of the transfer the EOQF bit in the DSPI_SR is set.
0 The SPI data is not the last data to transfer
1 The SPI data is the last data to transfer</td></tr>
<tr><td>26
CTCNT</td><td>Clear SPI_TCNT. The CTCNT provides a means for host software to clear the SPI transfer counter. The CTCNT bit clears the SPI_TCNT field in the DSPI_TCR register. The SPI_TCNT field is cleared before transmission of the current SPI frame begins.
0 Do not clear SPI_TCNT field in the DSPI_TCR
1 Clear SPI_TCNT field in the DSPI_TCR</td></tr>
<tr><td>23–16
PCSx</td><td>Peripheral Chip Select 0–7. The PCS bits select which PCS signals will be asserted for the transfer.
0 Negate the PCS[x] signal
1 Assert the PCS[x] signal</td></tr>
<tr><td>15–0
TXDATA[15:0]</td><td>Transmit Data. The TXDATA field holds SPI data to be transferred according to the associated SPI command.</td></tr>
</table>

30.11.2.7 DSPI POP RX FIFO Register (DSPI_POPR)

The DSPI_POPR provides a means to read the RX FIFO. See Section 30.12.3.5, “Receive First In First Out (RX FIFO) Buffering Mechanism,” for a description of the RX FIFO operations. Eight or sixteen bit read accesses to the DSPI_POPR will read from the RX FIFO and update the counter and pointer.

Address: DSPI_BASE + 0x38 Access:

	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																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	RXDATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-9. DSPI POP RX FIFO Register (DSPI_POPR)

Table 30-20. DSPI_POPR Field Descriptions

Field	Description
31–16	Reserved, should be cleared.
15–0 RXDATA[15:0]	Received Data. The RXDATA field contains the SPI data from the RX FIFO entry pointed to by the Pop Next Data Pointer.

30.11.2.8 DSPI Transmit FIFO Registers 0–3 (DSPI_TXFR0–DSPI_TXFR3)

The DSPI_TXFR0 - DSPI_TXFR3 registers provide visibility into the TX FIFO for debugging purposes. Each register is an entry in the TX FIFO. The registers are read-only and cannot be modified. Reading the DSPI_TXFRx registers does not alter the state of the TX FIFO.

Address: DSPI_BASE+0x3C–DSPI_BASE+0x48 Access:

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	TXCMD															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	TXDATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-10. DSPI Transmit FIFO Register 0–3 (DSPI_TXFR0–DSPI_TXFR3)

Table 30-21. DSPI_TXFR*n* Field Descriptions

Field	Description
31–16 TXCMD[15:0]	Transmit Command. The TXCMD field contains the command that sets the transfer attributes for the SPI data. See Section 30.11.2.6, “DSPI PUSH TX FIFO Register (DSPI_PUSHR),” for details on the command field.
15–0 TXDATA[15:0]	Transmit Data. The TXDATA field contains the SPI data to be shifted out.

30.11.2.9 DSPI Receive FIFO Registers 0–3 (DSPI_RXFR0–DSPI_RXFR3)

The DSPI_RXFR0 - DSPI_RXFR3 registers provide visibility into the RX FIFO for debugging purposes. Each register is an entry in the RX FIFO. The DSPI_RXFR registers are read-only. Reading the DSPI_RXFRx registers does not alter the state of the RX FIFO.

Address: DSPI_BASE + 0x4C–DSPI_BASE + 0x58 Access:

	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																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	RXDATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-11. DSPI Receive FIFO Registers 0–3 (DSPI_RXFR0–DSPI_RXFR3

Table 30-22. DSPI_RXFR*n* Field Descriptions

Field	Description
31–16	Reserved, should be cleared.
15–0 RXDATA[15:0]	Receive Data. The RXDATA field contains the received SPI data.

30.12 Functional Description

The Deserial Serial Peripheral Interface (DSPI) block supports full-duplex, synchronous serial communications between MCUs and peripheral devices.

The DSPI_CTAR0 - DSPI_CTAR7 registers hold clock and transfer attributes. The SPI configuration can select which CTAR to use on a frame by frame basis by setting a field in the SPI command. See Section 30.11.2.3, “DSPI Clock and Transfer Attributes Registers 0–7 (DSPI_CTAR0–DSPI_CTAR7),” for information on the fields of the DSPI_CTAR registers.

The 16-bit shift register in the Master and the 16-bit shift register in the Slave are linked by the SOUT and SIN signals to form a distributed 32-bit register. The Master and Slave use 16-bit shift registers. When a data transfer operation is performed, data is serially shifted a pre-determined number of bit positions. Because the registers are linked, data is exchanged between the Master and the Slave; the data that was in the Master's shift register is now in the shift register of the Slave, and vice versa. At the end of a transfer, the TCF bit in the DSPI_SR is set to indicate a completed transfer. Figure 30-12 illustrates how Master and Slave data is exchanged.

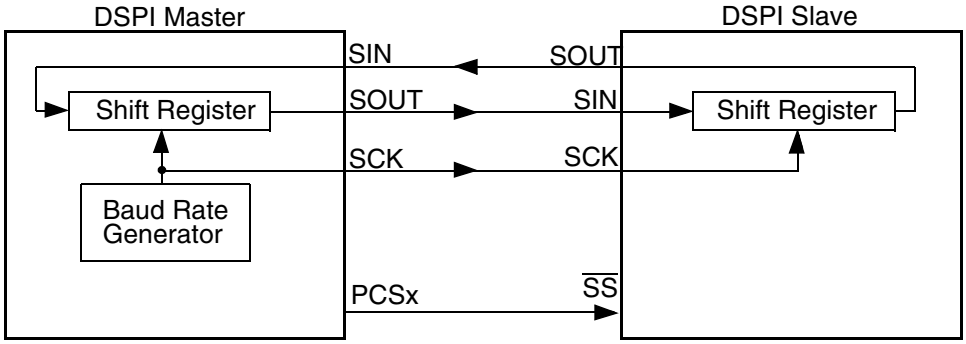


Figure 30-12. SPI Serial Protocol Overview

The DSPI has eight Peripheral Chip Select (PCS) signals that are used to select which of the Slaves to communicate with.

The three DSPI configurations share transfer protocol and timing properties so they are described independently of the configuration in Section 30.12.5, "Transfer Formats". The transfer rate and delay settings are described in Section 30.12.4, "DSPI Baud Rate and Clock Delay Generation."

See Section 30.12.8, "Power Saving Features," for information on the power-saving features of the DSPI.

30.12.1 Modes of Operation

The DSPI has five distinct modes:

- Master Mode
- Slave Mode
- Module Disable Mode
- External Stop Mode
- Debug Mode

Master, Slave, and Module Disable Modes are block-specific mode while External Stop and Debug Modes are MCU-specific modes.

The block-specific modes are determined by bits in the DSPI_MCR. External Stop Mode and Debug Mode are modes that the entire MCU can enter in parallel with the DSPI being configured in one of its block-specific modes.

30.12.1.1 Master Mode

In Master Mode the DSPI can initiate communications with peripheral devices. The DSPI operates as bus master when the MSTR bit in the DSPI_MCR is set. The Serial Communications Clock (SCK) is controlled by the Master DSPI.

Master Mode transfer attributes are controlled by the SPI command in the current TX FIFO entry. The CTAS field in the SPI command selects which of the eight DSPI_CTAR registers will be used to set the transfer attributes. Transfer attribute control is on a frame by frame basis. See Section 30.12.3, “Serial Peripheral Interface (SPI) Configuration,” for more details.

30.12.1.2 Slave Mode

In Slave Mode the DSPI responds to transfers initiated by an SPI master. The DSPI operates as bus slave when the MSTR bit in the DSPI_MCR register is negated. The DSPI slave is selected by a bus master by having the slave’s $\overline{SS}$ asserted.. In Slave Mode the SCK is provided by the bus master. All transfer attributes are controlled by the bus master but clock polarity, clock phase and numbers of bits to transfer must still be configured in the DSPI slave for proper communications.

In SPI Slave Mode the slave transfer attributes are set in the DSPI_CTAR0. The DSPI in Slave Mode transfers data MSB first. The LSBFE field of the associated CTAR is ignored.

30.12.1.3 Module Disable Mode

The Module Disable Mode is used for MCU power management. The clock to the non-memory mapped logic in the DSPI can be stopped while in Module Disable Mode.The DSPI enters the Module Disable Mode when the MDIS bit in DSPI_MCR is set or when a request for the DSPI to enter Doze Mode is asserted by an external controller while the DOZE bit in the DSPI_MCR is asserted. Logic external to the DSPI is needed to implement the Module Disable Mode. See Section 30.12.8, “Power Saving Features,” for more details on the Module Disable Mode.

30.12.1.4 External Stop Mode

The DSPI will not acknowledge the request to enter External Stop Mode until it has reached a frame boundary. When the DSPI has reached a frame boundary it will halt all operations and indicate that it is ready to have its clocks shut off. The DSPI exits External Stop Mode and resumes normal operation once the clocks are turned on. Serial communications or register accesses made while in External Stop Mode are ignored even if the clocks have not been shut off yet. See Section 30.12.8, “Power Saving Features,” for more details on the External Stop Mode.

30.12.1.5 Debug Mode

The Debug Mode is used for system development and debugging. If the SoC enters Debug Mode while the FRZ bit in the DSPI_MCR is set, the DSPI stops all serial transfers and enters a stopped state. If the SoC enters Debug Mode while the FRZ bit is negated, the DSPI behavior is unaffected and remains dictated by the block-specific mode and configuration of the DSPI. The DSPI enters Debug Mode when a debug request is asserted by an external controller. See Figure 30-13 for a state diagram.

30.12.2 Start and Stop of DSPI Transfers

The DSPI has two operating states; STOPPED and RUNNING. The states are independent of DSPI configuration. The default state of the DSPI is STOPPED. In the STOPPED state no serial transfers are initiated in Master Mode and no transfers are responded to in Slave Mode. The STOPPED state is also a safe state for writing the various configuration registers of the DSPI without causing undetermined results. The TXRXS bit in the DSPI_SR is negated in this state. In the RUNNING state serial transfers take place. The TXRXS bit in the DSPI_SR is asserted in the RUNNING state. Figure 30-13 shows a state diagram of the start and stop mechanism. The transitions are described in Table 30-23.

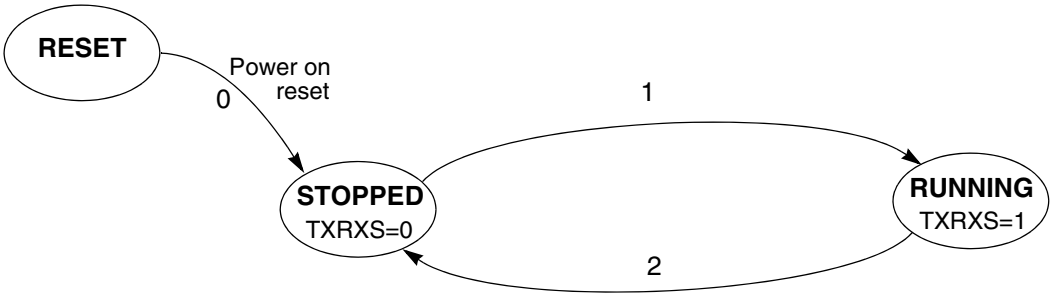


Figure 30-13. DSPI Start and Stop State Diagram

Table 30-23. State Transitions for Start and Stop of DSPI Transfers

Transition #	Current State	Next State	Description
0	RESET	STOPPED	Generic power-on-reset transition
1	STOPPED	RUNNING	The DSPI is started (DSPI transitions to RUNNING) when all of the following conditions are true: • EOQF bit is clear • Debug mode is unselected or the FRZ bit is clear • HALT bit is clear
2	RUNNING	STOPPED	The DSPI stops (transitions from RUNNING to STOPPED) after the current frame for any one of the following conditions: • EOQF bit is set • Debug mode is selected and the FRZ bit is set • HALT bit is set

State transitions from RUNNING to STOPPED occur on the next frame boundary if a transfer is in progress, or on the next system clock cycle if no transfers are in progress.

30.12.3 Serial Peripheral Interface (SPI) Configuration

The SPI Configuration transfers data serially using a shift register and a selection of programmable transfer attributes. The DSPI is in SPI Configuration when the DCONF field in the DSPI_MCR is 0b00. The SPI frames can be from four to sixteen bits long. The data to be transmitted can come from queues stored in RAM external to the DSPI. Host software or a DMA Controller can transfer the SPI data from the queues to a First-In First-Out (FIFO) buffer. The received data is stored in entries in the Receive FIFO

(RX FIFO) buffer. Host software or a DMA Controller transfer the received data from the RX FIFO to memory external to the DSPI. The FIFO buffer operations are described in Section 30.12.3.4, “Transmit First In First Out (TX FIFO) Buffering Mechanism,” and Section 30.12.3.5, “Receive First In First Out (RX FIFO) Buffering Mechanism.” The interrupt and DMA request conditions are described in Section 30.12.7, “Interrupts/DMA Requests.”

The SPI Configuration supports two block-specific modes; Master Mode and Slave Mode. The FIFO operations are similar for the Master Mode and Slave Mode. The main difference is that in Master Mode the DSPI initiates and controls the transfer according to the fields in the SPI command field of the TX FIFO entry. In Slave mode the DSPI only responds to transfers initiated by a bus master external to the DSPI and the SPI command field of the TX FIFO entry is ignored.

30.12.3.1 Master Mode

In SPI Master Mode the DSPI initiates the serial transfers by controlling the Serial Communications Clock (SCK) and the Peripheral Chip Select (PCS) signals. The SPI command field in the executing TX FIFO entry determines which CTAR registers will be used to set the transfer attributes and which PCS signal to assert. The command field also contains various bits that help with queue management and transfer protocol. See Section 30.11.2.6, “DSPI PUSH TX FIFO Register (DSPI_PUSHR)” for details on the SPI command fields. The data field in the executing TX FIFO entry is loaded into the shift register and shifted out on the Serial Out (SOUT) pin. In SPI Master Mode, each SPI frame to be transmitted has a command associated with it allowing for transfer attribute control on a frame by frame basis.

30.12.3.2 Slave Mode

In SPI Slave Mode the DSPI responds to transfers initiated by a SPI bus master. The DSPI does not initiate transfers. Certain transfer attributes such as clock polarity, clock phase and frame size must be set for successful communication with a SPI master. The SPI Slave Mode transfer attributes are set in the DSPI_CTAR0.

30.12.3.3 FIFO Disable Operation

The FIFO disable mechanisms allow SPI transfers without using the TX FIFO or RX FIFO. The DSPI operates as a double-buffered simplified SPI when the FIFOs are disabled. The TX and RX FIFOs are disabled separately. The TX FIFO is disabled by writing a ‘1’ to the DIS_TXF bit in the DSPI_MCR. The RX FIFO is disabled by writing a ‘1’ to the DIS_RXF bit in the DSPI_MCR.

The FIFO Disable mechanisms are transparent to the user and to host software; Transmit data and commands are written to the DSPI_PUSHR and received data is read from the DSPI_POPR.When the TX FIFO is disabled the TFFF, TFUF and TXCTR fields in DSPI_SR behave as if there is a one-entry FIFO but the contents of the DSPI_TXFR registers and TXNXTPTR are undefined. When the RX FIFO is disabled the RFDF, RFOF and RXCTR fields in the DSPI_SR behave as if there is a one-entry FIFO but the contents of the DSPI_RXFR registers and POPNXTPTR are undefined.

30.12.3.4 Transmit First In First Out (TX FIFO) Buffering Mechanism

The TX FIFO functions as a buffer of SPI data and SPI commands for transmission. The TX FIFO holds from one to sixteen words, each consisting of a command field and a data field. The number of entries in the TX FIFO is SoC specific. SPI commands and data are added to the TX FIFO by writing to the DSPI PUSH TX FIFO Register (DSPI_PUSHR). TX FIFO entries can only be removed from the TX FIFO by being shifted out or by flushing the TX FIFO.

The TX FIFO Counter field (TXCTR) in the DSPI Status Register (DSPI_SR) indicates the number of valid entries in the TX FIFO. The TXCTR is updated every time the DSPI _PUSHR is written or SPI data is transferred into the shift register from the TX FIFO.

The TXNXTPTR field indicates which TX FIFO Entry will be transmitted during the next transfer. The TXNXTPTR contains the positive offset from DSPI_TXFR0 in number of 32-bit registers. For example, TXNXTPTR equal to two means that the DSPI_TXFR2 contains the SPI data and command for the next transfer. The TXNXTPTR field is incremented every time SPI data is transferred from the TX FIFO to the shift register.

30.12.3.4.1 Filling the TX FIFO

Host software or other intelligent blocks can add (push) entries to the TX FIFO by writing to the DSPI_PUSHR. When the TX FIFO is not full, the TX FIFO Fill Flag (TFFF) in the DSPI_SR is set. The TFFF bit is cleared when TX FIFO is full and the DMA controller indicates that a write to DSPI_PUSHR is complete or by host software writing a ‘1’ to the TFFF in the DSPI_SR. The TFFF can generate a DMA request or an interrupt request. See Section 30.12.7.2, “Transmit FIFO Fill Interrupt or DMA Request,” for details.

The DSPI ignores attempts to push data to a full TX FIFO, i.e. the state of the TX FIFO is unchanged. No error condition is indicated.

30.12.3.4.2 Draining the TX FIFO

The TX FIFO entries are removed (drained) by shifting SPI data out through the shift register. Entries are transferred from the TX FIFO to the shift register and shifted out as long as there are valid entries in the TX FIFO. Every time an entry is transferred from the TX FIFO to the shift register, the TX FIFO Counter is decremented by one. At the end of a transfer, the TCF bit in the DSPI_SR is set to indicate the completion of a transfer. The TX FIFO is flushed by writing a ‘1’ to the CLR_TXF bit in DSPI_MCR.

If an external bus master initiates a transfer with a DSPI slave while the slave’s DSPI TX FIFO is empty, the Transmit FIFO Underflow Flag (TFUF) in the slave’s DSPI_SR is set. See Section Section 30.12.7.4, “Transmit FIFO Underflow Interrupt Request,” for details.

30.12.3.5 Receive First In First Out (RX FIFO) Buffering Mechanism

The RX FIFO functions as a buffer for data received on the SIN pin. The RX FIFO holds from one to sixteen received SPI data frames. The number of entries in the RX FIFO is SoC specific. SPI data is added to the RX FIFO at the completion of a transfer when the received data in the shift register is transferred into the RX FIFO. SPI data are removed (popped) from the RX FIFO by reading the DSPI POP RX FIFO

Register (DSPI_POPR). RX FIFO entries can only be removed from the RX FIFO by reading the DSPI_POPR or by flushing the RX FIFO.

The RX FIFO Counter field (RXCTR) in the DSPI Status Register (DSPI_SR) indicates the number of valid entries in the RX FIFO. The RXCTR is updated every time the DSPI _POPR is read or SPI data is copied from the shift register to the RX FIFO.

The POPNXTPTR field in the DSPI_SR points to the RX FIFO entry that is returned when the DSPI_POPR is read. The POPNXTPTR contains the positive offset from DSPI_RXFR0 in number of 32-bit registers. For example, POPNXTPTR equal to two means that the DSPI_RXFR2 contains the received SPI data that will be returned when DSPI_POPR is read. The POPNXTPTR field is incremented every time the DSPI_POPR is read.

30.12.3.5.1 Filling the RX FIFO

The RX FIFO is filled with the received SPI data from the shift register. While the RX FIFO is not full, SPI frames from the shift register are transferred to the RX FIFO. Every time a SPI frame is transferred to the RX FIFO the RX FIFO Counter is incremented by one.

If the RX FIFO and shift register are full and a transfer is initiated, the RFOF bit in the DSPI_SR is asserted indicating an overflow condition. Depending on the state of the ROOE bit in the DSPI_MCR, the data from the transfer that generated the overflow is either ignored or shifted in to the shift register. If the ROOE bit is asserted, the incoming data is shifted in to the shift register. If the ROOE bit is negated, the incoming data is ignored.

30.12.3.5.2 Draining the RX FIFO

Host software or other intelligent blocks can remove (pop) entries from the RX FIFO by reading the DSPI POP RX FIFO Register (DSPI_POPR). A read of the DSPI_POPR decrements the RX FIFO Counter by one. Attempts to pop data from an empty RX FIFO are ignored, the RX FIFO Counter remains unchanged. The data returned from reading an empty RX FIFO is undetermined.

When the RX FIFO is not empty, the RX FIFO Drain Flag (RFDF) in the DSPI_SR is set. The RFDF bit is cleared when the RX_FIFO is empty and the DMA controller indicates that a read from DSPI_POPR is complete or by host software writing a ‘1’ to the RFDF.

30.12.4 DSPI Baud Rate and Clock Delay Generation

The SCK frequency and the delay values for serial transfer are generated by dividing the system clock frequency by a prescaler and a scaler with the option for doubling the baud rate. Figure 30-14 shows conceptually how the SCK signal is generated.

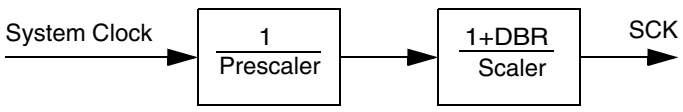


Figure 30-14. Communications Clock Prescalers and Scalers

30.12.4.1 Baud Rate Generator

The Baud Rate is the frequency of the Serial Communication Clock (SCK). The system clock is divided by a prescaler (PBR) and scaler (BR) to produce SCK with the possibity of halving the scaler division. The DBR, PBR and BR fields in the DSPI_CTAR registers select the frequency of SCK by the formula in the BR[3:0] field description. Table 30-24 shows an example of how to compute the baud rate.

Table 30-24. Baud Rate Computation Example

PBR	Prescaler	BR	Scaler	DBR	Fsys	Baud Rate
0b00	2	0b0000	2	0	100 MHz	25 Mb/s
0b00	2	0b0000	2	1	20 MHz	10 Mb/s

30.12.4.2 PCS to SCK Delay (t_{CSC})

The PCS to SCK delay is the length of time from assertion of the PCS signal to the first SCK edge. See Figure 30-16 for an illustration of the PCS to SCK delay. The PCSSCK and CSSCK fields in the DSPI_CTAR*x* registers select the PCS to SCK delay by the formula in the CSSCK[3:0] bit description. Table 30-25 shows an example of how to compute the PCS to SCK delay.

Table 30-25. PCS to SCK Delay Computation Example

PCSSCK	Prescaler	CSSCK	Scaler	Fsys	PCS to SCK Delay
0b01	3	0b0100	32	100 MHz	0.96 us

30.12.4.3 After SCK Delay (t_{ASC})

The After SCK Delay is the length of time between the last edge of SCK and the negation of PCS. See Figure 30-16 and Figure 30-17 for illustrations of the After SCK delay. The PASC and ASC fields in the DSPI_CTAR*x* registers select the After SCK Delay by the formula in the ASC[3:0] field description.. Table 30-26 shows an example of how to compute the After SCK delay.

Table 30-26. After SCK Delay Computation Example

PASC	Prescaler	ASC	Scaler	Fsys	After SCK Delay
0b01	3	0b0100	32	100 MHz	0.96 us

30.12.4.4 Delay after Transfer (t_{DT})

The Delay after Transfer is the length of time between negation of the PCS signal for a frame and the assertion of the PCS signal for the next frame. See Figure 30-16 for an illustration of the Delay after Transfer. The PDT and DT fields in the DSPI_CTAR*x* registers select the Delay after Transfer by the formula in the DT[3:0] field description. Table 30-27 shows an example of how to compute the Delay after Transfer.

Table 30-27. Delay after Transfer Computation Example

PDT	Prescaler	DT	Scaler	Fsys	Delay after Transfer
0b01	3	0b1110	32768	100 MHz	0.98 ms

30.12.4.5 Peripheral Chip Select Strobe Enable ($\overline{\text{PCSS}}$)

The $\overline{\text{PCSS}}$ signal provides a delay to allow the PCS signals to settle after transitioning thereby avoiding glitches. When the DSPI is in Master Mode and PCSSE bit is set in the DSPI_MCR, $\overline{\text{PCSS}}$ provides a signal for an external demultiplexer to decode the PCS[0] -PCS[4] and PCS[6] -PCS[7] signals into as many as 128 glitch-free PCS signals. Figure 30-15 shows the timing of the $\overline{\text{PCSS}}$ signal relative to PCS signals.

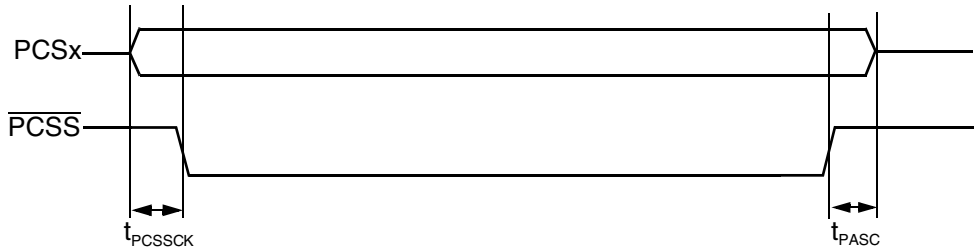


Figure 30-15. Peripheral Chip Select Strobe Timing

The delay between the assertion of the PCS signals and the assertion of $\overline{\text{PCSS}}$ is selected by the PCSSCK field in the DSPI_CTAR based on the following formula:

$$t_{PCSSCK} = \frac{1}{f_{SYS}} \times PCSSCK \qquad \textit{Eqn. 30-6}$$

At the end of the transfer the delay between $\overline{\text{PCSS}}$ negation and PCS negation is selected by the PASC field in the DSPI_CTAR based on the following formula:

$$t_{PASC} = \frac{1}{f_{SYS}} \times PASC \qquad \textit{Eqn. 30-7}$$

Table 30-28 shows an example of how to compute the t_{pcssck} delay.

Table 30-28. Peripheral Chip Select Strobe Assert Computation Example

PCSSCK	Prescaler	Fsys	Delay before Transfer
0b11	7	100 MHz	70.0 ns

Table 30-29 shows an example of how to compute the t_{pasc} delay.

Table 30-29. Peripheral Chip Select Strobe Negate Computation Example

PASC	Prescaler	Fsys	Delay after Transfer
0b11	7	100 MHz	70.0 ns

The $\overline{\text{PCSS}}$ signal is not supported when Continuous Serial Communication SCK is enabled (CONT_SCKE=1).

30.12.5 Transfer Formats

The SPI serial communication is controlled by the Serial Communications Clock (SCK) signal and the PCS signals. The SCK signal provided by the Master device synchronizes shifting and sampling of the data on the SIN and SOUT pins. The PCS signals serve as enable signals for the slave devices.

When the DSPI is the bus master, the CPOL and CPHA bits in the DSPI Clock and Transfer Attributes Registers (DSPI_CTARx) select the polarity and phase of the serial clock, SCK. The polarity bit selects the idle state of the SCK. The clock phase bit selects if the data on SOUT is valid before or on the first SCK edge.

When the DSPI is the bus Slave, CPOL and CPHA bits in the DSPI_CTAR0 select the polarity and phase of the serial clock. Even though the bus Slave does not control the SCK signal, clock polarity, clock phase and number of bits to transfer must be identical for the master device and the slave device to ensure proper transmission.

The DSPI supports four different transfer formats:

- Classic SPI with CPHA=0
- Classic SPI with CPHA=1
- Modified Transfer format with CPHA = 0
- Modified Transfer format with CPHA = 1

A modified transfer format is supported to allow for high-speed communication with peripherals that require longer setup times. The DSPI can sample the incoming data later than halfway through the cycle to give the peripheral more setup time. The MTFE bit in the DSPI_MCR selects between Classic SPI Format and Modified Transfer Format. The Classic SPI Formats are described in Section 30.12.5.1, “Classic SPI Transfer Format (CPHA = 0),” and Section 30.12.5.2, “Classic SPI Transfer Format (CPHA = 1).” The Modified Transfer Formats are described in Section 30.12.5.3, “Modified Transfer Format (MTFE = 1, CPHA = 0),” and Section 30.12.5.4, “Modified SPI Transfer Format (MTFE = 1, CPHA = 1).”

The DSPI provides the option of keeping the PCS signals asserted between frames. See Section 30.12.5.5, “Continuous Selection Format,” for details.

30.12.5.1 Classic SPI Transfer Format (CPHA = 0)

The transfer format shown in Figure 30-16 is used to communicate with peripheral SPI slave devices where the first data bit is available on the first clock edge. In this format, the master and slave sample their SIN pins on the odd-numbered SCK edges and change the data on their SOUT pins on the even-numbered SCK edges.

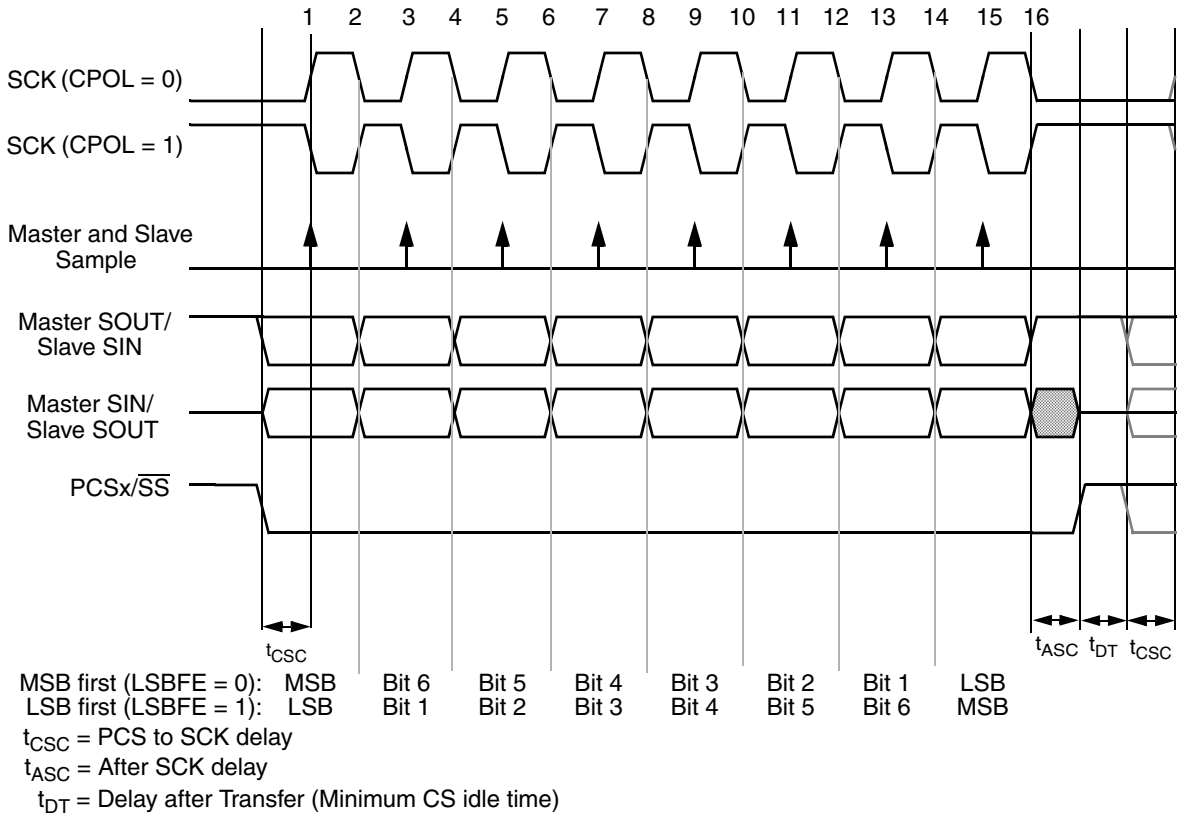


Figure 30-16. DSPI Transfer Timing Diagram (MTFE=0, CPHA=0, FMSZ=8)

The master initiates the transfer by placing its first data bit on the SOUT pin and asserting the appropriate peripheral chip select signals to the slave device. The slave responds by placing its first data bit on its SOUT pin. After the t_{CSC} delay has elapsed, the master outputs the first edge of SCK. This is the edge used by the master and slave devices to sample the first input data bit on their serial data input signals. At the second edge of the SCK the master and slave devices place their second data bit on their serial data output signals. For the rest of the frame the master and the slave sample their SIN pins on the odd-numbered clock edges and changes the data on their SOUT pins on the even-numbered clock edges. After the last clock edge occurs a delay of t_{ASC} is inserted before the master negates the PCS signals. A delay of t_{DT} is inserted before a new frame transfer can be initiated by the master.

30.12.5.2 Classic SPI Transfer Format (CPHA = 1)

This transfer format shown in Figure 30-17 is used to communicate with peripheral SPI slave devices that require the first SCK edge before the first data bit becomes available on the slave SOUT pin. In this format the master and slave devices change the data on their SOUT pins on the odd-numbered SCK edges and sample the data on their SIN pins on the even-numbered SCK edges

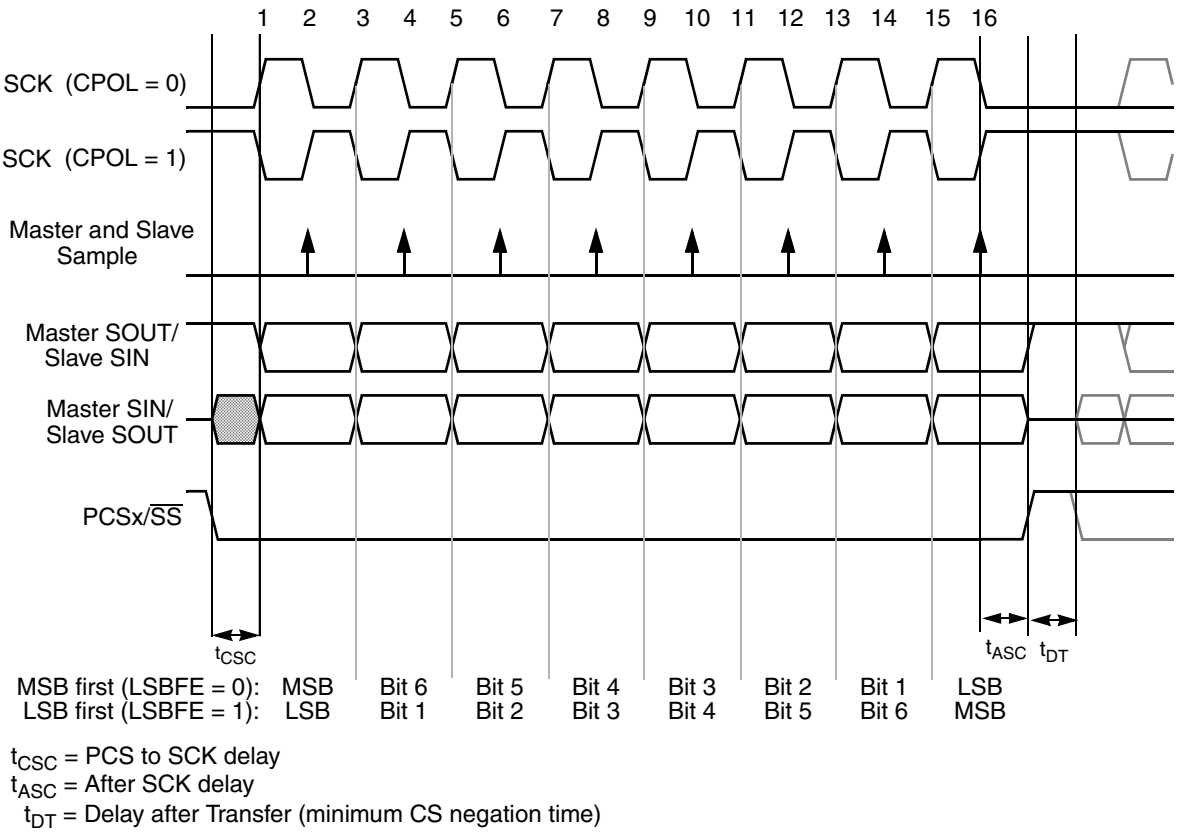


Figure 30-17. DSPI Transfer Timing Diagram (MTFE=0, CPHA=1, FMSZ=8)

The master initiates the transfer by asserting the PCS signal to the slave. After the t_{CSC} delay has elapsed, the master generates the first SCK edge and at the same time places valid data on the master SOUT pin. The slave responds to the first SCK edge by placing its first data bit on its slave SOUT pin.

At the second edge of the SCK the master and slave sample their SIN pins. For the rest of the frame the master and the slave change the data on their SOUT pins on the odd-numbered clock edges and sample their SIN pins on the even-numbered clock edges. After the last clock edge occurs a delay of t_{ASC} is inserted before the master negates the PCS signal. A delay of t_{DT} is inserted before a new frame transfer can be initiated by the master.

30.12.5.3 Modified Transfer Format (MTFE = 1, CPHA = 0)

In this Modified Transfer Format both the Master and the Slave sample later in the SCK period than in Classic SPI mode to allow for delays in device pads and board traces. These delays become a more significant fraction of the SCK period as the SCK period decreases with increasing baud rates.

The Master and the Slave places data on the SOUT pins at the assertion of the PCS signal. After the PCS to SCK delay has elapsed the first SCK edge is generated. The Slave samples the Master SOUT signal on every odd numbered SCK edge. The Slave also places new data on the Slave SOUT on every odd numbered clock edge.

The Master places its second data bit on the SOUT line one system clock after odd numbered SCK edge. The point where the Master samples the Slave SOUT is selected by writing to the SMPL_PT field in the DSPI_MCR. The SMPL_PT field description in Table 30-8 lists the number of system clock cycles between the active edge of SCK and the Master Sample point. The Master sample point can be delayed by one or two system clock cycles.

Figure 30-18 shows the modified transfer format for CPHA = 0. Only the condition where CPOL = 0 is illustrated. The delayed Master sample points are indicated with a lighter shaded arrow.

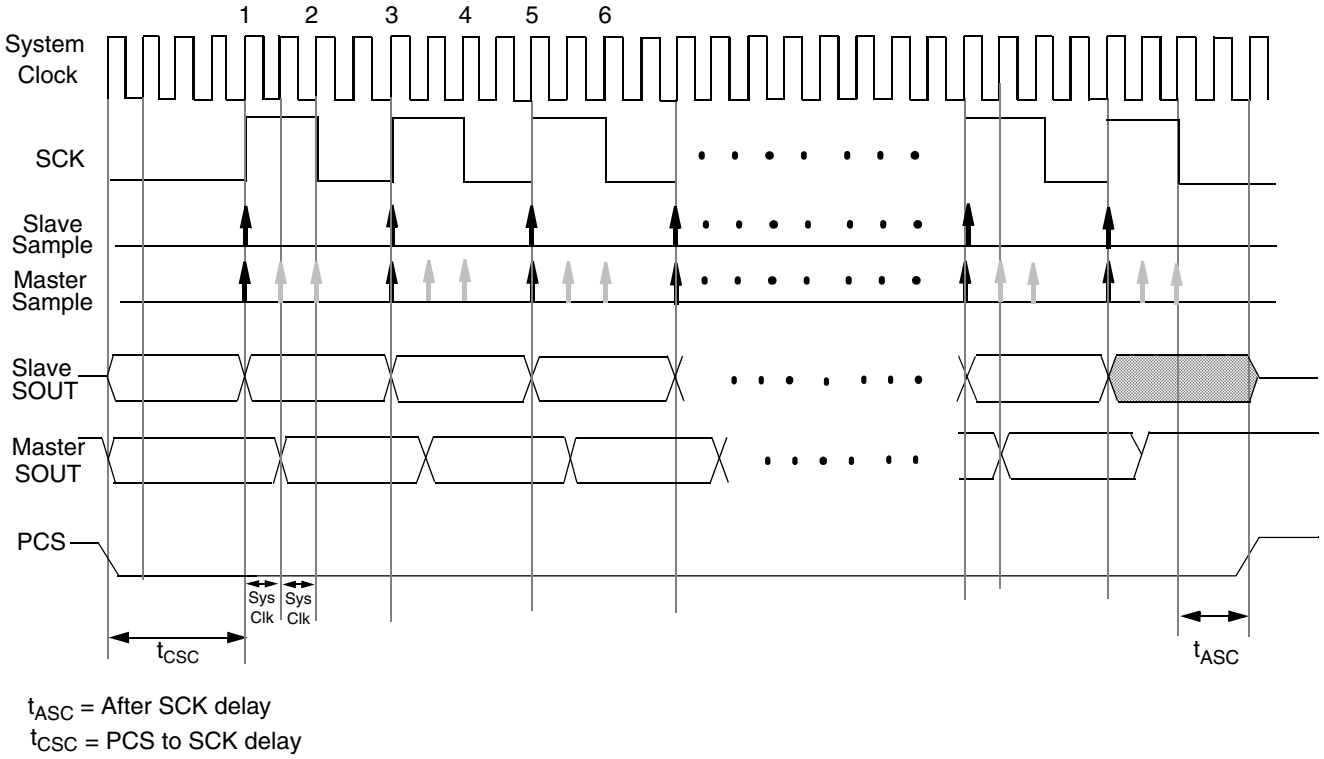
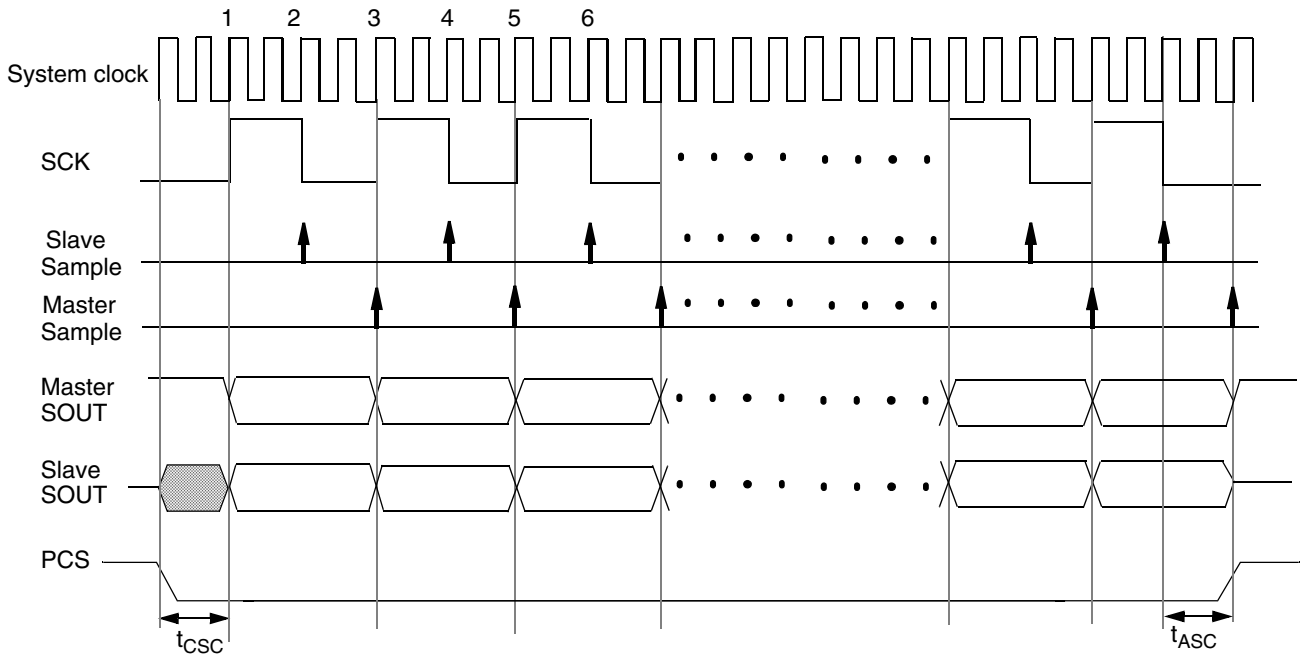


Figure 30-18. DSPI Modified Transfer Format (MTFE=1, CPHA=0, Fsck = Fsys/4)

30.12.5.4 Modified SPI Transfer Format (MTFE = 1, CPHA = 1)

Figure 30-19 shows the Modified Transfer Format for CPHA = 1. Only the condition where CPOL = 0 is described. At the start of a transfer the DSPI asserts the PCS signal to the slave device. After the PCS to SCK delay has elapsed the master and the slave put data on their SOUT pins at the first edge of SCK. The Slave samples the Master SOUT signal on the even numbered edges of SCK. The Master samples the Slave SOUT signal on the odd numbered SCK edges starting with the 3rd SCK edge. The Slave samples the last bit on the last edge of the SCK. The Master samples the last Slave SOUT bit one half SCK cycle after the last edge of SCK. No clock edge will be visible on the Master SCK pin during the sampling of the last bit. The SCK to PCS delay must be greater or equal to half of the SCK period.



t_{CSC} = PCS to SCK delay
t_{ASC} = After SCK delay

Figure 30-19. DSPI Modified Transfer Format (MTFE=1, CPHA=1, Fsck = Fsys/4)

30.12.5.5 Continuous Selection Format

Some peripherals must be deselected between every transfer. Other peripherals must remain selected between several sequential serial transfers. The Continuous Selection Format provides the flexibility to handle both cases. The Continuous Selection Format is enabled for the SPI Configuration by setting the CONT bit in the SPI command.

When the CONT bit = 0, the DSPI drives the asserted Chip Select signals to their idle states in between frames. The idle states of the Chip Select signals are selected by the PCSIS field in the DSPI_MCR. Figure 30-20 shows the timing diagram for two four-bit transfers with CPHA = 1 and CONT = 0.

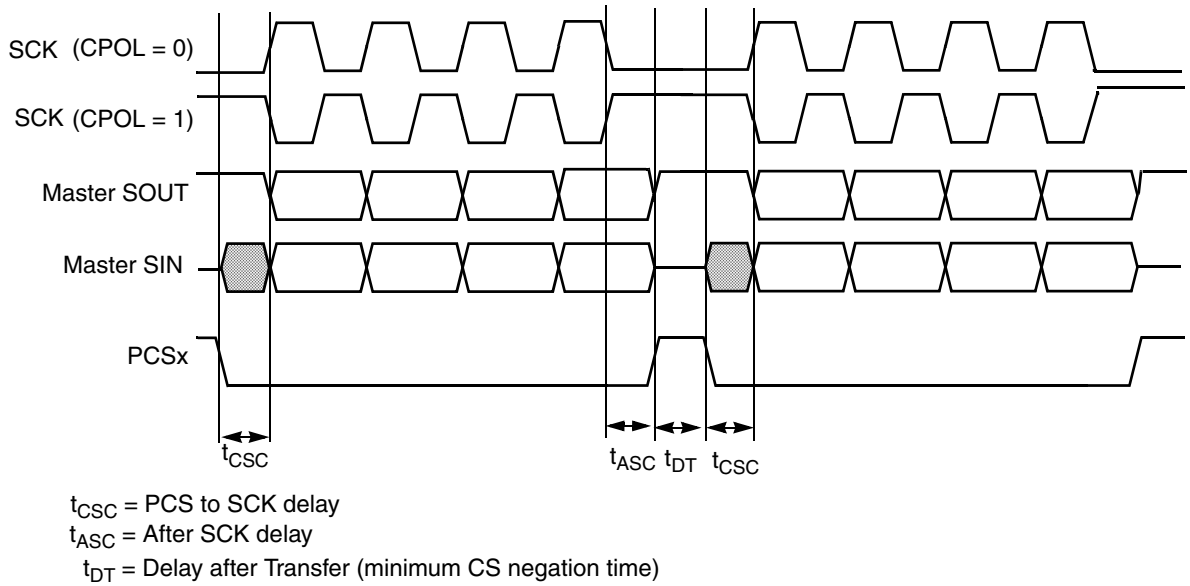


Figure 30-20. Example of Non-Continuous Format (CPHA=1, CONT=0)

When the CONT bit = 1, the PCS signal remains asserted for the duration of the two transfers. The Delay between Transfers (t_{DT}) is not inserted between the transfers. Figure 30-21 shows the timing diagram for two four-bit transfers with CPHA = 1 and CONT = 1.

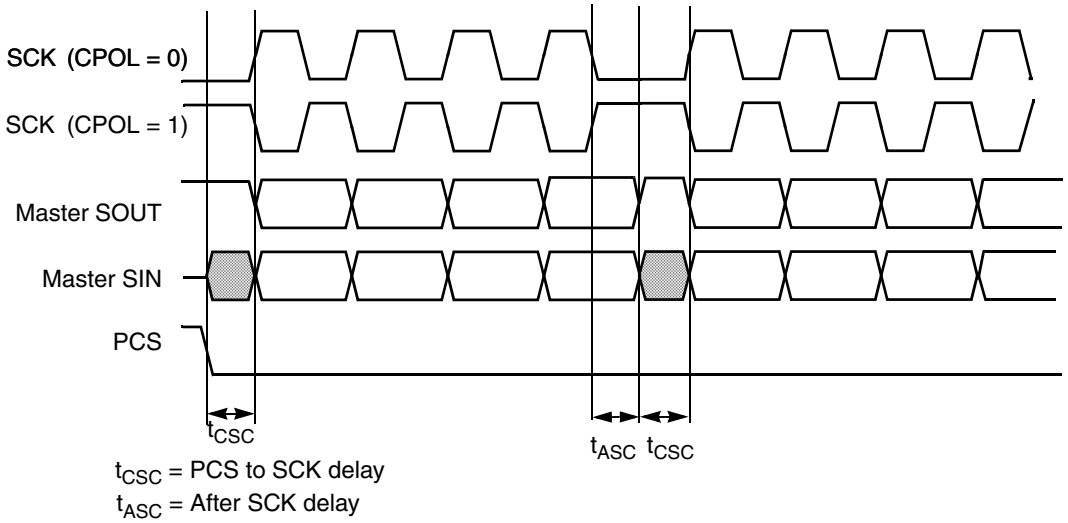


Figure 30-21. Example of Continuous Transfer (CPHA=1, CONT=1)

Switching CTAR registers or changing which PCS signals are asserted between frames while using Continuous Selection can cause errors in the transfer. The PCS signal should be negated before CTAR is switched or different PCS signals are selected.

30.12.6 Continuous Serial Communications Clock

The DSPI provides the option of generating a continuous SCK signal for slave peripherals that require a continuous clock.

Continuous SCK is enabled by setting the CONT_SCKE bit in the DSPI_MCR. Continuous SCK is valid in all configurations.

Continuous SCK is only supported for CPHA=1. Setting CPHA=0 will be ignored if the CONT_SCKE bit is set. Continuous SCK is supported for Modified Transfer Format.

Clock and transfer attributes for the Continuous SCK mode are set according to the following rules:

- CTAR0 shall be used initially. At the start of each SPI frame transfer, the CTAR specified by the CTAS for the frame shall be used.
- The currently selected CTAR shall remain in use until the start of a frame with a different CTAR specified, or the Continuous SCK mode is terminated.

It is recommended that the baud rate is the same for all transfers made while using the Continuous SCK. Switching clock polarity between frames while using Continuous SCK can cause errors in the transfer. Continuous SCK operation is not guaranteed if the DSPI is put into the External Stop Mode or Module Disable Mode.

Enabling Continuous SCK disables the PCS to SCK delay and the Delay after Transfer (T_{DT}) is fixed at one T_{SCK} cycle. Figure 30-22 shows timing diagram for Continuous SCK format with Continuous Selection disabled.

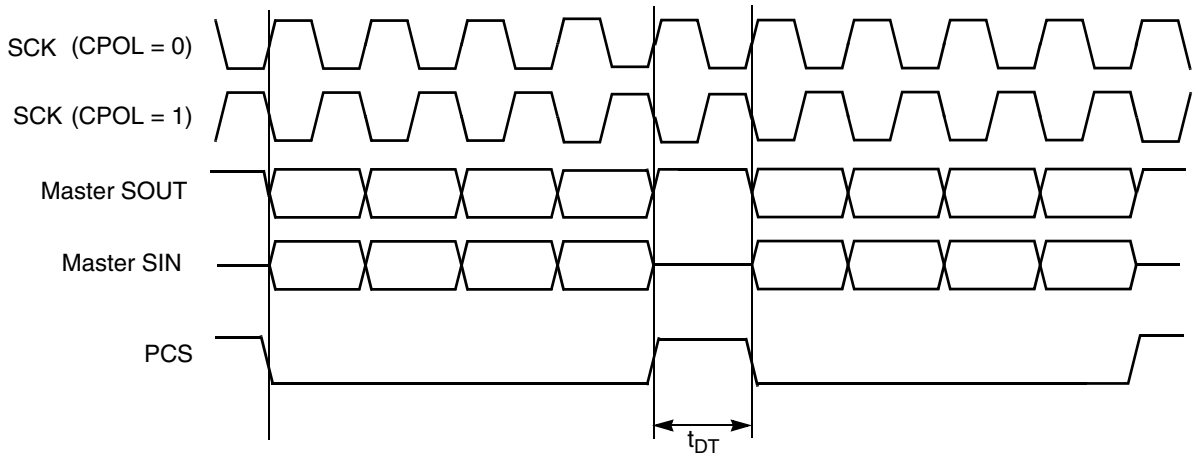


Figure 30-22. Continuous SCK Timing Diagram (CONT=0)

If the CONT bit in the TX FIFO entry is set, PCS remains asserted between the transfers . Under certain conditions, SCK can continue with PCS asserted, but with no data being shifted out of SOUT (SOUT pulled high). This can cause the slave to receive incorrect data. Those conditions include:

- Continuous SCK with CONT bit set, but no data in the transmit FIFO.
- Continuous SCK with CONT bit set and entering STOPPED state (refer to Section 30.12.2, “Start and Stop of DSPI Transfers”).

- Continuous SCK with CONT bit set and entering Stop mode or Module Disable mode.

Figure 30-23 shows timing diagram for Continuous SCK format with Continuous Selection enabled.

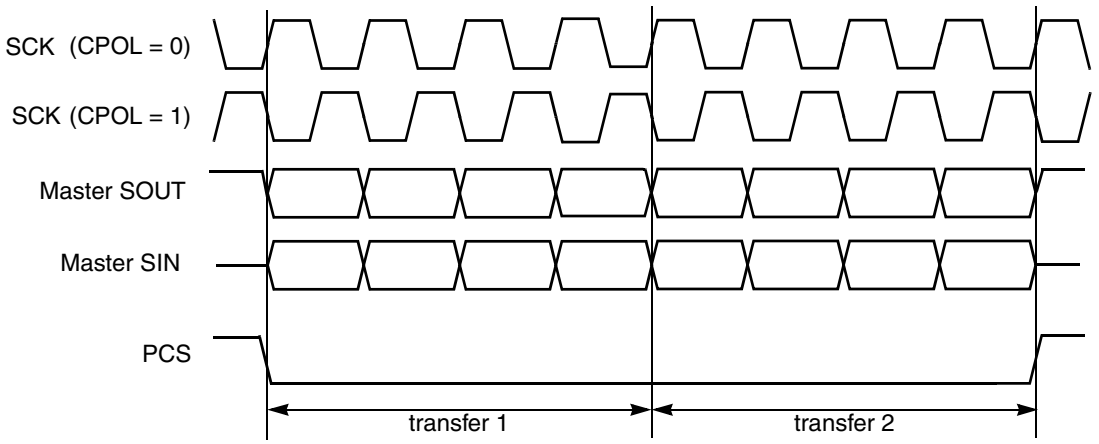


Figure 30-23. Continuous SCK Timing Diagram (CONT=1)

30.12.7 Interrupts/DMA Requests

The DSPI has four conditions that can only generate interrupt requests and two conditions that can generate interrupt or DMA request. Table 30-30 lists the six conditions.

Table 30-30. Interrupt and DMA Request Conditions

Condition	Flag	Interrupt	DMA
End of Queue (EOQ)	EOQF	X	
TX FIFO Fill	TFFF	X	X
Transfer Complete	TCF	X	
TX FIFO Underflow	TFUF	X	
RX FIFO Drain	RFDF	X	X
RX FIFO Overflow	RFOF	X	

Each condition has a flag bit in the DSPI Status Register (DSPI_SR) and an Request Enable bit in the DSPI DMA/Interrupt Request Select and Enable Register (DSPI_RSER). The TX FIFO Fill Flag (TFFF) and RX FIFO Drain Flag (RFDF) generate interrupt requests or DMA requests depending on the TFFF_DIRS and RFDF_DIRS bits in the DSPI_RSER.

30.12.7.1 End of Queue Interrupt Request

The End of Queue Request indicates that the end of a transmit queue is reached. The End of Queue Request is generated when the EOQ bit in the executing SPI command is asserted and the EOQF_RE bit in the DSPI_RSER is asserted.

30.12.7.2 Transmit FIFO Fill Interrupt or DMA Request

The Transmit FIFO Fill Request indicates that the TX FIFO is not full. The Transmit FIFO Fill Request is generated when the number of entries in the TX FIFO is less than the maximum number of possible entries, and the TFFF_RE bit in the DSPI_RSER is asserted. The TFFF_DIRS bit in the DSPI_RSER selects whether a DMA request or an interrupt request is generated.

30.12.7.3 Transfer Complete Interrupt Request

The Transfer Complete Request indicates the end of the transfer of a serial frame. The Transfer Complete Request is generated at the end of each frame transfer when the TCF_RE bit is set in the DSPI_RSER.

30.12.7.4 Transmit FIFO Underflow Interrupt Request

The Transmit FIFO Underflow Request indicates that an underflow condition in the TX FIFO has occurred. The transmit underflow condition is detected only for DSPI blocks operating in slave mode and SPI configuration. The TFUF bit is set when the TX FIFO of a DSPI operating in slave mode and SPI configuration is empty, and a transfer is initiated from an external SPI master. If the TFUF bit is set while the TFUF_RE bit in the DSPI_RSER is asserted, an interrupt request is generated.

30.12.7.5 Receive FIFO Drain Interrupt or DMA Request

The Receive FIFO Drain Request indicates that the RX FIFO is not empty. The Receive FIFO Drain Request is generated when the number of entries in the RX FIFO is not zero, and the RFDF_RE bit in the DSPI_RSER is asserted. The RFDF_DIRS bit in the DSPI_RSER selects whether a DMA request or an interrupt request is generated.

30.12.7.6 Receive FIFO Overflow Interrupt Request

The Receive FIFO Overflow Request indicates that an overflow condition in the RX FIFO has occurred. A Receive FIFO Overflow request is generated when RX FIFO and shift register are full and a transfer is initiated. The RFOF_RE bit in the DSPI_RSER must be set for the interrupt request to be generated.

Depending on the state of the ROOE bit in the DSPI_MCR, the data from the transfer that generated the overflow is either ignored or shifted in to the shift register. If the ROOE bit is set, the incoming data is shifted in to the shift register. If the ROOE bit is negated, the incoming data is ignored.

30.12.8 Power Saving Features

The DSPI supports three power-saving strategies:

- External Stop Mode
- Module Disable Mode - Clock gating of non-memory mapped logic
- Clock gating and clock to memory-mapped logic

The External Stop Mode requires a block external to the DSPI to implement the SoC power management and clock gating control. All power saving features require logic external to the DSPI. Figure 30-24 shows an example on how the DSPI power saving features can be used in an SoC.

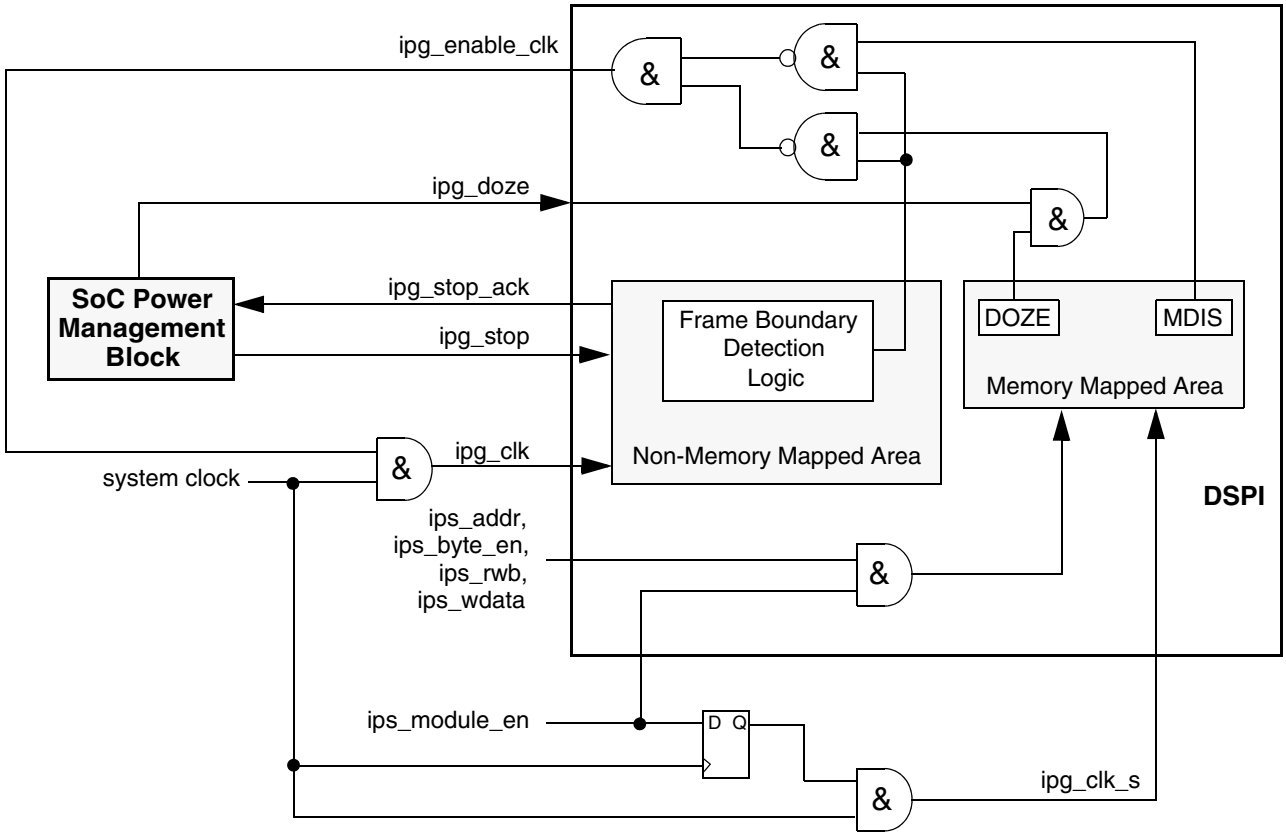


Figure 30-24. Example of a DSPI in an SoC with a Power Management Block

30.12.8.1 External Stop Mode

When a request is made to enter External Stop Mode, the DSPI block acknowledges the request by negating ipg_stop_ack. When the DSPI is ready to have its clocks shut off the ipg_stop_ack signal is asserted. If a serial transfer is in progress, the DSPI waits until it reaches the frame boundary before it asserts ipg_stop_ack. While the clocks are shut off, the DSPI memory-mapped logic is not accessible. The states of the interrupt and DMA request signals cannot be changed while in External Stop Mode.

30.12.8.2 Module Disable Mode

Module Disable Mode is a block-specific mode that the DSPI can enter to save power. Host software can initiate the Module Disable Mode by writing a ‘1’ to the MDIS bit in the DSPI_MCR. The Module Disable Mode can also be initiated by hardware. A power management block can initiate Module Disable Mode by asserting the ipg_doze signal while the DOZE bit in the DSPI_MCR is asserted.

When the MDIS bit is asserted or the ipg_doze signal is asserted while the DOZE bit is asserted, the DSPI negates ipg_enable_clk at the next frame boundary. If implemented, the ipg_enable_clk signal can stop the clock to the non-memory mapped logic. When ipg_enable_clk is negated, the DSPI is in a dormant state, but the memory mapped registers are still accessible. Certain read or write operations have a different affect when the DSPI is in the Module Disable Mode. Reading the RX FIFO Pop Register will not change the state of the RX FIFO. Likewise, writing to the TX FIFO Push Register will not change the state of the

TX FIFO. Clearing either of the FIFOs will not have any affect in the Module Disable Mode. Changes to the DIS_TXF and DIS_RXF fields of the DSPI_MCR will not have any affect in the Module Disable Mode. In the Module Disable Mode, all status bits and register flags in the DSPI will return the correct values when read, but writing to them will have no affect. Writing to the DSPI_TCR during Module Disable Mode will not have any affect. Interrupt and DMA request signals cannot be cleared while in the Module Disable Mode.

30.12.8.3 Signal Gating

The DSPI's module enable signal is used to gate signals such as address, byte enable, read/write and data. This prevents toggling signals from propagating through parts of the DSPI's combinational logic and consuming power unless it is a DSPI access. The module enable signal can also be used to gate the clock (ipg_clk_s) to the memory-mapped logic.

30.13 Initialization/Application Information

30.13.1 How to Change Queues

This section presents an example of how to change queues for the DSPI. The queues are not part of the DSPI, but the DSPI includes features in support of queue management. Queues are primarily supported in SPI Configuration.

1. The last command word from a queue is executed. The EOQ bit in the command word is set to indicate to the DSPI that this is the last entry in the queue.
2. At the end of the transfer, corresponding to the command word with EOQ set is sampled, the EOQ flag (EOQF) in the DSPI_SR is set.
3. The setting of the EOQF flag will disable both serial transmission, and serial reception of data, putting the DSPI in the STOPPED state. The TXRXS bit is negated to indicate the STOPPED state.
4. The DMA will continue to fill TX FIFO until it is full or step 5 occurs.
5. Disable DSPI DMA transfers by disabling the DMA enable request for the DMA channel assigned to TX FIFO and RX FIFO. This is done by clearing the corresponding DMA enable request bits in the DMA Controller.
6. Ensure all received data in RX FIFO has been transferred to memory receive queue by reading the RXCNT in DSPI_SR or by checking RFDF in the DSPI_SR after each read operation of the DSPI_POPR.
7. Modify DMA descriptor of TX and RX channels for "new" queues
8. Flush TX FIFO by writing a '1' to the CLR_TXF bit in the DSPI_MCR, Flush RX FIFO by writing a '1' to the CLR_RXF bit in the DSPI_MCR.
9. Clear transfer count either by setting CTCNT bit in the command word of the first entry in the new queue or via CPU writing directly to SPI_TCNT field in the DSPI_TCR.
10. Enable DMA channel by enabling the DMA enable request for the DMA channel assigned to the DSPI TX FIFO, and RX FIFO by setting the corresponding DMA set enable request bit.
11. Enable serial transmission and serial reception of data by clearing the EOQF bit.

30.13.2 Baud Rate Settings

Table 30-31 shows the baud rate that is generated based on the combination of the baud rate prescaler PBR and the baud rate scaler BR in the DSPI_CTAR registers. The values calculated assume a 100 MHz system frequency and the double baud rate DBR bit is clear.

Table 30-31. Baud Rate Values

		Baud Rate Divider Prescaler Values			
		2	3	5	7
Baud Rate Scaler Values	2	25.0M	16.7M	10.0M	7.14M
	4	12.5M	8.33M	5.00M	3.57M
	6	8.33M	5.56M	3.33M	2.38M
	8	6.25M	4.17M	2.50M	1.79M
	16	3.12M	2.08M	1.25M	893k
	32	1.56M	1.04M	625k	446k
	64	781k	521k	312k	223k
	128	391k	260k	156k	112k
	256	195k	130k	78.1k	55.8k
	512	97.7k	65.1k	39.1k	27.9k
	1024	48.8k	32.6k	19.5k	14.0k
	2048	24.4k	16.3k	9.77k	6.98k
	4096	12.2k	8.14k	4.88k	3.49k
	8192	6.10k	4.07k	2.44k	1.74k
	16384	3.05k	2.04k	1.22k	872
	32768	1.53k	1.02k	610	436

30.13.3 Delay Settings

Table 30-32 shows the values for the Delay after Transfer (t_{DT}) and CS to SCK Delay (t_{CSC}) that can be generated based on the prescaler values and the scaler values set in the DSPI_CTAR registers. The values calculated assume a 100 MHz system frequency.

Table 30-32. Delay Values

		Delay Prescaler Values			
		1	3	5	7
Delay Scaler Values	2	20.0 ns	60.0 ns	100.0 ns	140.0 ns
	4	40.0 ns	120.0 ns	200.0 ns	280.0 ns
	8	80.0 ns	240.0 ns	400.0 ns	560.0 ns
	16	160.0 ns	480.0 ns	800.0 ns	1.1 µs
	32	320.0 ns	960.0 ns	1.6 µs	2.2 µs
	64	640.0 ns	1.9 µs	3.2 µs	4.5 µs
	128	1.3 µs	3.8 µs	6.4 µs	9.0 µs
	256	2.6 µs	7.7 µs	12.8 µs	17.9 µs
	512	5.1 µs	15.4 µs	25.6 µs	35.8 µs
	1024	10.2 µs	30.7 µs	51.2 µs	71.7 µs
	2048	20.5 µs	61.4 µs	102.4 µs	143.4 µs
	4096	41.0 µs	122.9 µs	204.8 µs	286.7 µs
	8192	81.9 µs	245.8 µs	409.6 µs	573.4 µs
	16384	163.8 µs	491.5 µs	819.2 µs	1.1 ms
	32768	327.7 µs	983.0 µs	1.6 ms	2.3 ms
	65536	655.4 µs	2.0 ms	3.3 ms	4.6 ms

30.13.4 Oak Family Compatibility with the DSPI

Table 30-33 shows the translation of commands written to the TX FIFO command halfword with commands written to the Command Ram of the Oak family QSPI. The table illustrates how to configure the DSPI_CTAR registers to match the default cases for the possible combinations of the Oak Family Control Bits in its Command RAM. The defaults for the Oak Family are based on a system clock of 40MHz. All delay variables below will generate the same delay, or as close a possible, from the DSPI 100MHz system clock that an Oak Family part would generate from its 40MHz system clock. For other system clock frequencies, the customer can recompute the values using Delay Settings.

- For BITSE = 0 → 8 bits per transfer
- For DT = 0 → 0.425µs delay: For this value, the closest value in the DSPI is 0.480µs
- For DSCK = 0 → 1/2 SCK period: For this value, the value for the DSPI is 20ns

Table 30-33. Oak Family QSPI Compatibility with the DSPI

Oak Family Control Bits DSPI Corresponding Control Bits						Corresponding DSPI_CTAR Register Configuration					
BITSE	CTAS[2]	DT	CTAS[1]	DSCK	CTAS[0]	DSPI_CTAR*x*	FMSZ	PDT	DT	PCSSCK	CSSCK
0		0		0		0	1111	10	0011	00	0000
0		0		1		1	1111	10	0011	user	user
0		1		0		2	1111	user[1]	user	00	0000
0		1		1		3	1111	user	user	user	user
1		0		0		4	user	10	0011	00	0000
1		0		1		5	user	10	0011	user	user
1		1		0		6	user	user	user	00	0000
1		1		1		7	user	user	user	user	user

1. Selected by user

30.13.5 Calculation of FIFO Pointer Addresses

The user has complete visibility of the TX and RX FIFO contents through the FIFO registers, and valid entries can be identified through a memory mapped pointer and a memory mapped counter for each FIFO. The pointer to the first-in entry in each FIFO is memory mapped. For the TX FIFO the first-in pointer is the Transmit Next Pointer (TXNXTPTR). For the RX FIFO the first-in pointer is the Pop Next Pointer (POPNXTPTR). Figure 30-25 illustrates the concept of first-in and last-in FIFO entries along with the FIFO Counter. The TX FIFO is chosen for the illustration, but the concepts carry over to the RX FIFO. See Section 30.12.3.4, “Transmit First In First Out (TX FIFO) Buffering Mechanism,” and Section 30.12.3.5, “Receive First In First Out (RX FIFO) Buffering Mechanism,” for details on the FIFO operation.

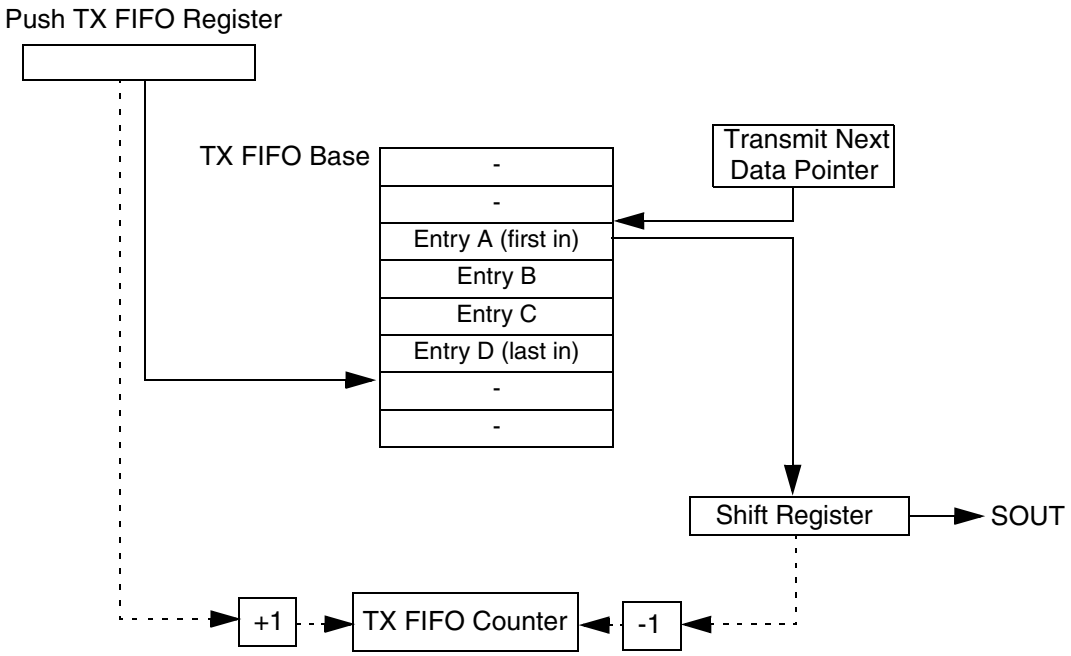


Figure 30-25. TX FIFO Pointers and Counter

30.13.5.1 Address Calculation for the First-in Entry and Last-in Entry in the TX FIFO

The memory address of the first-in entry in the TX FIFO is computed by the following equation:

$$\text{First-in Entry Address} = \text{TX FIFO Base} + (4 \times \text{TXNXTPTR}) \qquad \textit{Eqn. 30-8}$$

The memory address of the last-in entry in the TX FIFO is computed by the following equation:

$$\text{Last-in Entry address} = \text{TX FIFO Base} + 4 \times \text{modulo TX FIFO depth}(\text{TXCTR} + \text{TXNXTPTR} - 1) \qquad \textit{Eqn. 30-9}$$

TX FIFO Base - Base address of TX FIFO
TXCTR - TX FIFO Counter
TXNXTPTR - Transmit Next Pointer
TX FIFO Depth - Transmit FIFO depth, implementation specific

30.13.5.2 Address Calculation for the First-in Entry and Last-in Entry in the RX FIFO

The memory address of the first-in entry in the RX FIFO is computed by the following equation:

$$\text{First-in Entry Address} = \text{RX FIFO Base} + (4 \times \text{POPNXTPTR}) \qquad \textit{Eqn. 30-10}$$

The memory address of the last-in entry in the RX FIFO is computed by the following equation:

$$\text{Last-in Entry address} = \text{RX FIFO Base} + 4 \times \text{modulo RX FIFO depth}(\text{RXCTR} + \text{POPNXTPTR} - 1) \quad \textit{Eqn. 30-11}$$

RX FIFO Base - Base address of RX FIFO
RXCTR - RX FIFO counter
POPNXTPTR - Pop Next Pointer
RX FIFO Depth - Receive FIFO depth, implementation specific

Chapter 31
Enhanced Serial Communications Interface (eSCI)

31.1 Introduction to the eSCI on MAC7200

The ESCI provides asynchronous serial communications for external devices and peripherals. The Enhancement offered by this module over other Freescale SCIs is the inclusion of additional features that support a LIN bus master and comply to the LIN2.0 specification.

Each of the ESCI modules can be independently disabled by writing to the MDIS bit in the modules Control Register 3. Disabling the module will turn off the clock to the module, although it will still allow access to the modules registers by the core across the peripheral bus. The MDIS bit is intended to be used when the module is not required in the application. By default the ESCIs are disabled after reset (MDIS=1), so prior to use, the MDIS bit must be cleared.

31.1.1 Block Diagram

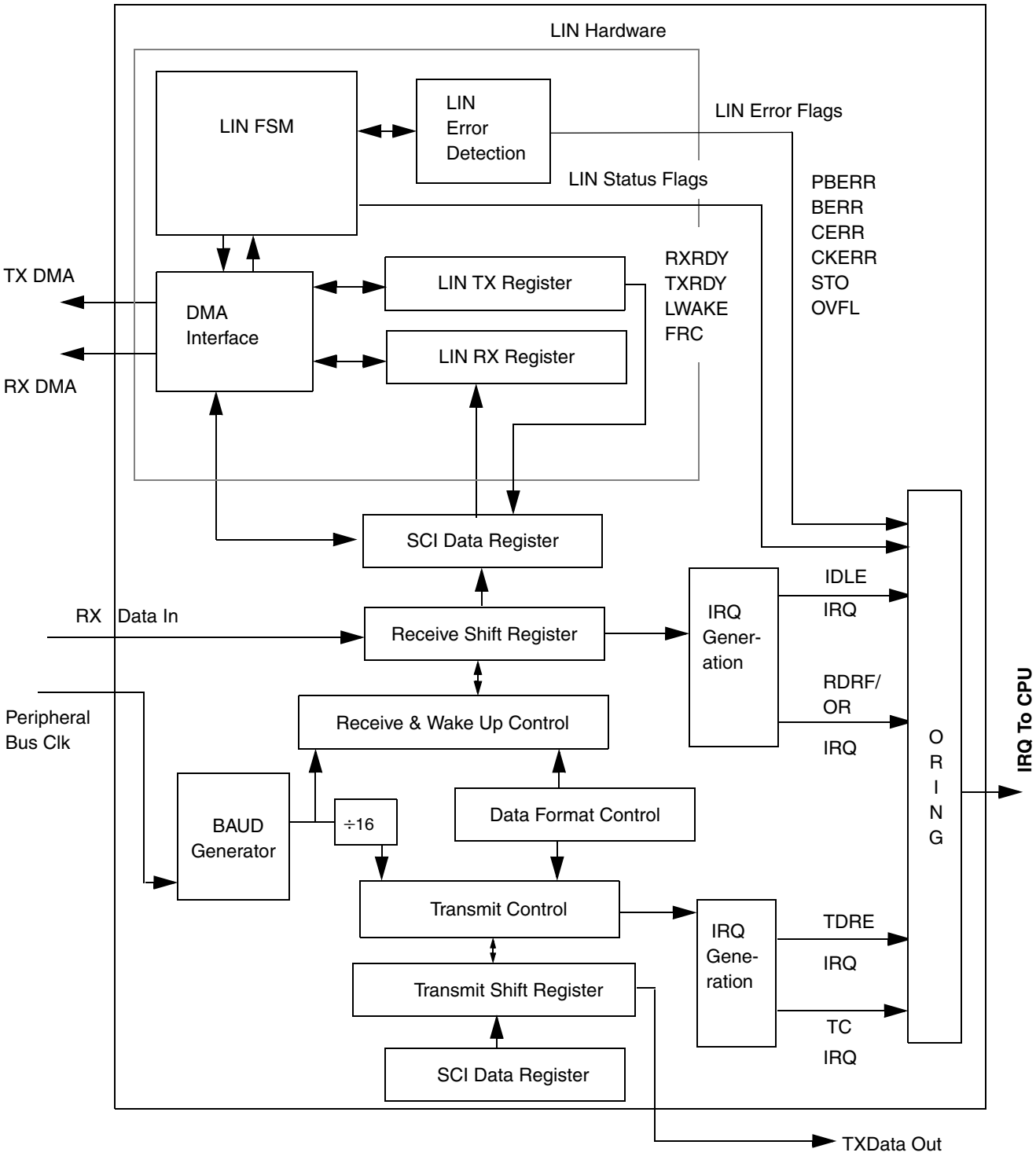


Figure 31-1. eSCI Block Diagram

31.2 eSCI Features

The SCI modules will often be used on the MAC72xx as LIN Masters, with the main functionality being polling of several nodes on a LIN bus. There is currently no requirement for IrDA on the MAC72xx. The eSCI has the following features:

- Memory Map: 32-bit peripheral with 8 bytes, byte/halfword/word addressable
- Standard Non return-to-Zero (NRZ) Mark/Space Format.
- Full-Duplex Operation.
- Software Selectable Word Length (8-bit or 9-bit words).
- 10/11 or 13/14 bit Break Character possible.
- 13-Bit Programmable Baud-Rate Modulus Counter.
- Separately Enabled Transmitter and Receiver.
- Separate receiver and transmitter CPU interrupt requests.
- Programmable transmitter output polarity.
- Two receiver wake up methods:
  - Idle line walk-up
  - Address mark walk-up
- Interrupt-driven operation with eight flags:
  - Transmitter empty
  - Transmission complete
  - Receiver full
  - Idle receiver input
  - Receiver overrun
  - Noise error
  - Framing error
  - Parity error
- Receiver framing error detection.
- Hardware Parity Checking.
- 1/16 bit time noise reduction.
- LIN Master mode state machine
  - Supports generation of LIN message header.
  - Detection and flagging of LIN errors.
- Two DMA request lines on each peripheral for receive and transmit data.

31.2.1 LIN support

The eSCI provides the following LIN features:

- LIN Master Node functionality (master and slave task)
- Compatible with LIN slaves from revisions 1.x and 2.0 of the LIN standard

- Detection of Bit Errors, Physical Bus Errors and Checksum Errors
- All status bit can generate maskable interrupts
- Application layer CRC support
- Programmable CRC polynom
- Double Stop Flag insertion after Bit Errors
- Detection and generation of wakeup characters
- Programmable wakeup delimiter time
- Programmable slave timeout
- Can be configured to include header bits in checksum
- LIN DMA interface

Please refer to the LIN Specification Package for details about the LIN protocol.

31.2.2 Modes of Operation

The SCI functions the same in normal, special, and emulation modes. It also has a low power Doze mode.

- Run Mode
- Doze Mode

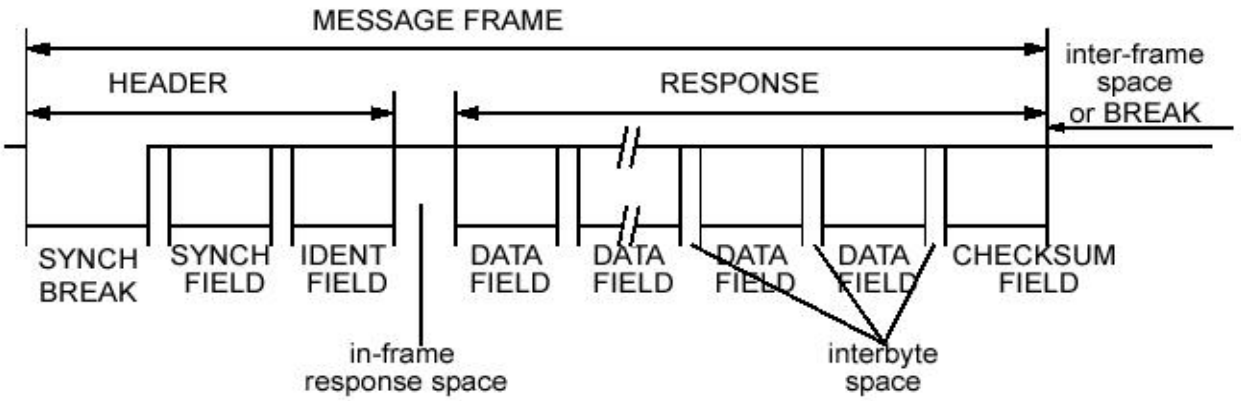
The eSCI delays the system going into stop mode, until it has completely transmitted the current TX byte, or completely received the current RX byte. In LIN mode it will complete any frames which don't require further processor intervention (e.g. transmission of a checksum byte).

31.3 eSCI Protocol

31.3.1 LIN Protocol Summary

The LIN bus has the following characteristics:

- 2-wire, full duplex bus (RXD + TXD)
- Synchronous (the clock is derived from the data stream)
- Single-master, multi-slave
- Transmit "monitoring". The LIN master is responsible for checking that the transmitted data is the same as the data on the bus
- Bit rate from 1 kb/s to 19.2 kb/s, with recommended rates of 2400 bps, 9600 bps and 19200 bps.
- 8N1 byte encoding, LSB first (1 Start + 8 Data + 1 stop bit). In this sense, a "byte" is 10 bits (as measured by the bit time/slice).
- Payload is delivered in a Message Frame, which consists of:



— **HEADER** (sent by the Master Node)
  – **SYNCH BREAK** Field - Consists of (min) 13 “low” bits + (min) 1 “high” bit
  – **SYNCH** Field - 1 byte ($55)
  – **IDENT** Field - 1 byte with 6 IDENTIFIER bits and 2 PARITY bits

— **Inframe response space** - between the HEADER and the RESPONSE, the bus may go high (recessive) for anywhere between 0 and x bits. ‘x’ may be as long as need be, as long as the Maximum Frame timing requirement is not violated.

— **RESPONSE** (sent by the Master or Slave Node)
  – **DATA** Fields (2,4 or 8 per Message Frame) - 1 byte (LSB first) per Field
  – **CHECKSUM** Field - 1 byte (LSB first) per Field

— **Interbyte space** - between each DATA/CHECKSUM byte, the bus may go high (recessive) for anywhere between 0 and x bits. ‘x’ may be as long as need be, as long as the Maximum Frame timing requirement is not violated.

This gives us the following Message Frame timings:

- Min. Message Frame (T_{FRAME_MIN})

```
14 bits (SYNCH BREAK) + 1 byte (SYNCH) + 1 byte (IDENT) + #DATA * 1 byte + 1 byte (CHECKSUM)
= 44 bits + #DATA_FIELDS * 10 bits
= 64 bits (2 byte DATA payload)
= 84 bits (4 byte DATA payload)
= 124 bits (8 byte DATA payload)
```

- Max. Message Frame (T_{FRAME_MAX})

```
(T_FRAME_MIN + 1) * 1.4
= 65 bits * 1.4 = 91 bits (2 byte DATA payload)
= 85 bits * 1.4 = 119 bits (4 byte DATA payload)
= 125 bits * 1.4 = 175 bits (8 byte DATA payload)
```

- Min. HEADER (T_{HEADER_MIN})

```
13 bits + 1 bits (SYNCH BREAK) + 1 byte (SYNCH) + 1 byte (IDENT)
= 34 bits
```

- Max. HEADER (T_{HEADER_MAX})

```
(T_HEADER_MIN + 1) * 1.4
= 35 bits * 1.4 = 49 bits
```

- Bus Idle Timeout (T_{TIME_OUT})

  `25,000 bits`

31.4 eSCI Implementation

The eSCI used on the MAC72xx is identical to the eSCI used on the MAC71xx family of devices, except for the bug fixes and enhancements listed in Section 31.7, “eSCI Differences from MAC71xx.

31.5 eSCI External Pins

Table 31-1. eSCI External Pins

TXD_A, TXD_B	Serial Data Transmit
RXD_A, RXD_B	Serial Data Receive

31.5.1 SCI Transmit Pin (TXD_A, TXD_B)

These pins serve as transmit data outputs of eSCI_A and eSCI_B.

31.5.2 SCI Receive Pin (RXD)

These pins serve as receive data inputs of eSCI_A and eSCI_B.

31.6 eSCI Bus Aborts

The SCI module supports Peripheral Bus bus aborts, and enforces the following memory map:

Table 31-2. eSCI Bus Aborts

Abort	Allowed
	$0000-$001f
$0020-$3fff	

Supervisor Access: Unused.

NOTE

Reserved register space inside the “allowed” address range will return $00 for READs, and WRITEs will be ignored.

31.7 eSCI Differences from MAC71xx

- Added Fast bit error detection in LIN
  - Added FBR bit in the SCICR3 register
  - Added BESM13 and SBSTP bits in the SCICR4 register
- Fixed MUCts01378: Fast bit error detection raised RXRDY
- Fixed MUCts01644: Write on bus abort still writes the register

- Added PMASK Parity Masking
  - Added PMSK bit in the SCICR4 register
- Added unrequested data interrupt
  - Added UREQ bit in the LINSTAT2 register
  - Added UQIE bit in the LINCTRL3 register

31.8 eSCI Application Usage

Please refer to the eSCI Block Guide for information on using the eSCI in an application.

31.8.1 Enabling the eSCI

Before the eSCI can be used, it must be explicitly enabled by clearing the **MDIS** bit.

31.9 Memory Map and Register Definition

This section provides a detailed description of all memory and registers.

31.9.1 Memory Map

The memory map for the eSCI module is given below. The Address listed for each register is the address offset. The total address for each register is the sum of the base address for the eSCI module and the address offset for each register.

Table 31-3. eSCI Memory Map

Offset	Use	Access
0x0	SCI Baud Rate Register High (SCIBDH)	Read/Write
0x1	SCI Baud Rate Register Low (SCIBDL)	Read/Write
0x2	SCI Control Register1 (SCICR1)	Read/Write
0x3	SCI Control Register 2 (SCICR2)	Read/Write
0x4	SCI Control Register 3 (SCICR3)	Read/Write
0x5	SCI Control Register 4 (SCICR4)	Read/Write
0x6	SCI Data Register High (SCIDRH)	Read/Write
0x7	SCI Data Register Low (SCIDRL)	Read/Write
0x8	SCI Status Register 1 (SCISR1)	Read/Write
0x9	SCI Status Register 2 (SCISR2)	Read/Write
0xA	LIN Status Register 1 (LINSTAT1)	Read/Write
0xB	LIN Status Register 2 (LINSTAT2)	Read/Write
0xC	LIN Control Register 1 (LINCTRL1)	Read/Write
0xD	LIN Control Register 2 (LINCTRL2)	Read/Write

Table 31-3. eSCI Memory Map (Continued)

0xE	LIN Control Register 3 (LINCTRL3)	Read/Write
0xF	Reserved	—
0x10	LIN TX Register (LINTX)	Write
0x11–0x13	Reserved	—
0x14	LIN RX Register (LINRX)	Read
0x15–0x17	Reserved	—
0x18	LIN CRC Polynom Register 1 (LINCRCP1)	Read/Write
0x19	LIN CRC Polynom Register 2 (LINCRCP2)	Read/Write
0x1A–0x1F	Reserved	—

NOTE

Reserved registers will read 00h**.**

Table 31-4. eSCI Register Quick Reference

Register name		Bit 7	6	5	4	3	2	1	Bit 0	Addr. offset
SCIBDH (bit 31-24)	Read:	0	0	0	SBR12	SBR11	SBR10	SBR9	SBR8	0x0
	Write:									
SCIBDL (bit 23-16)	Read:	SBR7	SBR6	SBR5	SBR4	SBR3	SBR2	SBR1	SBR0	0x1
	Write:									
SCICR1 (bit 15-8)	Read:	LOOPS	SCISDOZ	RSRC	M	WAKE	ILT	PE	PT	0x2
	Write:									
SCICR2 (bit 7-0)	Read:	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK	0x3
	Write:									
SCICR3 (bit 31-24)	Read:	MDIS	FBR	BSTP	IEBERR	RXDMA	TXDMA	BRK13	TXDIR	0x4
	Write:									
SCICR4 (bit 23-16)	Read:	BESM13	SBSTP	0	PMSK	ORIE	NFIE	FEIE	PFIE	0x5
	Write:									
SCIDRH (bit 15-8)	Read:	R8	T8	0	0	0	0	0	0	0x6
	Write:									

Table 31-4. eSCI Register Quick Reference (Continued)

Register name		Bit 7	6	5	4	3	2	1	Bit 0	Addr. offset
SCIDRL (bit 7-0)	Read:	R7	R6	R5	R4	R3	R2	R1	R0	0x7
	Write:	T7	T6	T5	T4	T3	T2	T1	T0	
SCISR1 (bit 31-24)	Read:	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF	0x8
	Write:									
SCISR2 (bit 23-16)	Read:	0	0	0	BERR	0	0	0	RAF	0x9
	Write:									
LINSTAT1 (bit 15-8)	Read:	RXRDY	TXRDY	LWAKE	STO	PBERR	CERR	CKERR	FRC	0xA
	Write:									
LINSTAT2 (bit 7-0)	Read:	0	0	0	0	0	0	UREQ	OVFL	0xB
	Write:									
LINCTRL1 (bit 31-24)	Read:	LRES	0	WUD1	WUD0	LDBG	DSF	PRTY	LIN	0xC
	Write:		WU							
LINCTRL2 (bit 23-16)	Read:	RXIE	TXIE	WUIE	STIE	PBIE	CIE	CKIE	FCIE	0xD
	Write:									
LINCTRL3 (bit 15-8)	Read:	0	0	0	0	0	0	UQIE	OFIE	0xE
	Write:									
LINTX (bit 31-24)	Read:									0x10
	1st W:	P1	P0	ID5	ID4	ID3	ID2	ID1	ID0	
	2nd W:	L7	L6	L5	L4	L3	L2	L1	L0	
	3rd W:	HDCHK	CSUM	CRC	TX	T11	T10	T9	T8	
	4th W:	T7	T6	T5	T4	T3	T2	T1	T0	
	5th+ W:	D7	D6	D5	D4	D3	D2	D1	D0	
LINRX (bit 31-24)	Read:	D7	D6	D5	D4	D3	D2	D1	D0	0x14
	Write:									
LINCRCP1 (bit 31-24)	Read:	P15	P14	P13	P12	P11	P10	P9	P8	0x18
	Write:									

Table 31-4. eSCI Register Quick Reference (Continued)

Register name		Bit 7	6	5	4	3	2	1	Bit 0	Addr. offset
LINCRCP2 (bit 23-16)	Read:	P7	P6	P5	P4	P3	P2	P1	P0	0x19
	Write:									

NOTE

Registers which belong to the same 32-bit word can be accessed together with one 32-bit access - registers which belong to the same 16-bit word can be accessed together with one 16-bit access. (E.g. SCIDRH/L can be accessed with 16-bit reads and 16-bit writes.)

31.9.2 Register Descriptions

This section consists of register descriptions in address order. Each description includes a standard register diagram with an associated figure number. Writes to a reserved register location do not have any effect and reads of these locations return a zero. Details of register bit and field function follow the register diagrams, in bit order.

NOTE

Before accessing the registers the module must be enabled by writing the MDIS bit in the SCI control register 3.

31.9.2.1 SCI Baud Rate Registers

The SCI Baud Rate Register is used to select the baud rate of the SCI. The formula for calculating the baud rate is:

SCI baud rate = SCI module clock / (16 x BR),

where BR is the content of the SCI baud rate registers, bits SBR12 through SBR0. The baud rate registers can contain a value from 1 to 8191.

Read: anytime. If only SCIBDH is written to, a read will not return the correct data until SCIBDL is written to as well, following a write to SCIBDH.

Write: anytime

Address Offset: 0x0 Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	SBR12	SBR11	SBR10	SBR9	SBR8
W								
Reset	0	0	0	0	0	0	0	0

Figure 31-2. SCI Baud Rate Register High (SCIBDH)

Address Offset: 0x1 Access: User read/write

	7	6	5	4	3	2	1	0
R W	SBR7	SBR6	SBR5	SBR4	SBR3	SBR2	SBR1	SBR0
Reset	0	0	0	0	0	1	0	0

Figure 31-3. SCI Baud Rate Register Low (SCIBDL)

Table 31-5. SCI BDH/L Field Descriptions

Field	Description
7–5	Reserved, should be cleared.
4–0 7–0 SBR*n*	SCI Baud Rate Bits. The baud rate for the SCI is determined by these 13 bits. **Note:** The baud rate generator is disabled until the TE bit or the RE bit is set for the first time after reset. The baud rate generator is disabled when BR = 0. **Note:** Writing to SCIBDH has no effect without writing to SCIBDL, since writing to SCIBDH puts the data in a temporary location until SCIBDL is written to. Normally the Baudrate should be written with a single 16-bit write.

31.9.2.2 SCI Control Register 1

Address Offset: 0x2 Access: User read/write

	7	6	5	4	3	2	1	0
R W	LOOPS	SCISDOZ	RSRC	M	WAKE	ILT	PE	PT
Reset	0	0	0	0	0	0	0	0

Figure 31-4. SCI Control Register 1 (SCICR1)

Table 31-6. SCICR1 Field Descriptions

Field	Description
7 LOOPS	Loop Select Bit. LOOPS enables loop operation. In loop operation, the RXD pin is disconnected from the SCI and the transmitter output is internally connected to the receiver input. Both the transmitter and the receiver must be enabled to use the loop function. 0 Normal operation enabled 1 Loop operation enabled The receiver input is determined by the RSRC bit.
6 SCISDOZ	SCI Stop in Doze Mode Bit. SCISDOZ disables the SCI in Doze mode. In Doze mode it is no longer possible to access all registers, e.g. it is no longer possible to clear interrupts. (Not applicable if the system doesn't support Doze mode.) 0 SCI enabled in Doze mode 1 SCI disabled in Doze mode

Table 31-6. SCICR1 Field Descriptions (Continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>5
RSRC</td><td>Receiver Source Bit. When LOOPS = 1, the RSRC bit determines the source for the receiver shift register input.
0 Receiver input internally connected to transmitter output
1 Receiver input connected externally to transmitter
<table>
<tr><th>LOOPS</th><th>RSRC</th><th>Function</th></tr>
<tr><td>0</td><td>x</td><td>Normal operation</td></tr>
<tr><td>1</td><td>0</td><td>Loop mode with Rx input internally connected to Tx output</td></tr>
<tr><td>1</td><td>1</td><td>Single-wire mode with Rx input connected to TXD</td></tr>
</table></td></tr>
<tr><td>4
M</td><td>Data Format Mode Bit . MODE determines whether data characters are eight or nine bits long.
0 One start bit, eight data bits, one stop bit
1 One start bit, nine data bits, one stop bit</td></tr>
<tr><td>3
WAKE</td><td>Wakeup Condition Bit. WAKE determines which condition wakes up the eSCI: a logic 1 (address mark) in the most significant bit position of a received data character or an idle condition on the RXD.
0 Idle line wakeup
1 Address mark wakeup
Note: This is not a wakeup out of a power-save mode, it refers solely to the receiver standby mode.</td></tr>
<tr><td>2
ILT</td><td>Idle Line Type Bit. ILT determines when the receiver starts counting logic 1s as idle character bits. The counting begins either after the start bit or after the stop bit. If the count begins after the start bit, then a string of logic 1s preceding the stop bit may cause false recognition of an idle character. Beginning the count after the stop bit avoids false idle character recognition, but requires properly synchronized transmissions.
0 Idle character bit count begins after start bit
1 Idle character bit count begins after stop bit</td></tr>
<tr><td>1
PE</td><td>Parity Enable Bit. PE enables the parity function. When enabled, the parity function inserts a parity bit in the most significant bit position of the transmit data. On receive data it will verify the parity bit in the most significant bit position. The received parity bit will not be masked out, however.
0 Parity function disabled
1 Parity function enabled</td></tr>
<tr><td>0
PT</td><td>Parity Type Bit. PT determines whether the eSCI generates and checks for even parity or odd parity. With even parity, an even number of 1s clears the parity bit and an odd number of 1s sets the parity bit. With odd parity, an odd number of 1s clears the parity bit and an even number of 1s sets the parity bit.
0 Even parity
1 Odd parity</td></tr>
</table>

31.9.2.3 SCI Control Register 2

Address Offset: 0x3 Access: User read/write

	7	6	5	4	3	2	1	0
R W	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
Reset	0	0	0	0	0	0	0	0

Figure 31-5. SCI Control Register 2 (SCICR2)

Table 31-7. SCICR2 Field Descriptions

Field	Description
7 TIE	Transmitter Interrupt Enable Bit. TIE enables the transmit data register empty flag TDRE to generate interrupt requests. 0 TDRE interrupt requests disabled 1 TDRE interrupt requests enabled
6 TCIE	Transmission Complete Interrupt Enable Bit. TCIE enables the transmission complete flag TC to generate interrupt requests. 0 TC interrupt requests disabled 1 TC interrupt requests enabled
5 RIE	Receiver Full Interrupt Enable Bit. RIE enables the receive data register full flag RDRF to generate interrupt requests. 0 RDRF interrupt requests disabled 1 RDRF interrupt requests enabled
4 ILIE	Idle Line Interrupt Enable Bit. ILIE enables the idle line flag IDLE to generate interrupt requests. 0 IDLE interrupt requests disabled 1 IDLE interrupt requests enabled
3 TE	Transmitter Enable Bit. TE enables the eSCI transmitter and configures the TXD pin as being controlled by the eSCI. The TE bit can be used to queue an idle preamble. 0 Transmitter disabled 1 Transmitter enabled
2 RE	Receiver Enable Bit. RE enables the eSCI receiver. RE can be cleared to discard the current frame. 0 Receiver disabled 1 Receiver enabled
1 RWU	Receiver Wakeup Bit. Standby state. 0 Normal operation. 1 RWU enables the wakeup function and inhibits further receiver interrupt requests. Normally, hardware wakes the receiver by automatically clearing RWU (see Section 31.10.5.6, “Receiver Wakeup”).
0 SBK	Send Break Bit. Toggling SBK sends one break character (10 or 11 logic 0s, respectively 13 or 14 logics 0s if BRK13 is set). Toggling implies clearing the SBK bit before the break character has finished transmitting. As long as SBK is set, the transmitter continues to send complete break characters (10 or 11 bits, respectively 13 or 14 bits). 0 No break characters 1 Transmit break characters

31.9.2.4 SCI Control Register 3

Address Offset: 0x4 Access: User read/write

	7	6	5	4	3	2	1	0
R W	MDIS	FBR	BSTP	IEBERR	RXDMA	TXDMA	BRK13	TXDIR
Reset	1	0	1	0	0	0	0	0

Figure 31-6. SCI Control Register 3 (SCICR3)

Table 31-8. SCICR3 Field Descriptions

Field	Description
7 MDIS	Module Disable. By default the module is disabled, and needs to be explicitly turned on here (by writing a 0 to it) before it can transmit or receive data. For correct operation the module must be turned on before writing to any of the other registers.
6 FBR	Fast Bit Error Detection. Handles Bit Error Detection on a per Bit basis. If this is not enabled (compatibility mode), bit errors will be detected on a per byte basis. (See Section 31.10.4.6, “Fast Bit Error Detection in LIN mode.”) This bit is only relevant in LIN mode.
5 BSTP	Bit Error / Physical Bus Error Stop. Causes DMA TX requests to be suppressed, as long as the bit error and physical bus error flags are not cleared. This stops further DMA writes, which would otherwise cause data bytes to be interpreted as LIN header information. This bit is only relevant in LIN mode.
4 IEBERR	Enable Bit Error Interrupt. Generates an interrupt, when a Bit Error is detected.
3 RXDMA	Activate RX DMA channel. If this Bit is enabled and the eSCI has received data, it will raise a DMA RX request.
2 TXDMA	Activate TX DMA channel. Whenever the eSCI is able to transmit data, it will raise a DMA TX request.
1 BRK13	Break Transmit character length, This bit determines whether the transmit break character is 10 or 11 bit respectively 13 or 14 bits long. The detection of a framing error is not affected by this bit. LIN 2.0 now requires that a break character is always 13 bits long, so this Bit should always be set to 1. 0 Break Character is 10 or 11 bit long 1 Break character is 13 or 14 bit long
0 TXDIR	Transmitter pin data direction in Single-Wire mode. This bit determines whether the TXD pin is going to be used as an input or output, in the Single-Wire mode of operation. This bit is only relevant in the Single-Wire mode of operation. 0 TXD pin to be used as an input in Single-Wire mode 1 TXD pin to be used as an output in Single-Wire mode

31.9.2.5 SCI Control Register 4

Address Offset: 0x5 Access: User read/write

	7	6	5	4	3	2	1	0
R	BESM13	SBSTP	0	PMSK	ORIE	NFIE	FEIE	PFIE
W								
Reset	0	0	0	0	0	0	0	0

Figure 31-7. SCI Control Register 4 (SCICR4)

Table 31-9. SCICR4 Field Descriptions

Field	Description
7 BESM13	Bit Error Sample Mode, Bit 13. Determines when to sample the incoming bit in order to detect a bit error. (This is only relevant in LIN mode and when FBR is set.) 0 Sample at RT clock 9 1 Sample at RT clock 13 (see Section 31.10.5.3, “Data Sampling”)
6 SBSTP	SCI Bit Error Stop. Stops the SCI when a Bit Error is asserted. This allows to stop driving the LIN bus quickly after a Bit Error has been detected. The SCI won’t start a new byte transmission until the time for the current byte has expired. This bit is only relevant in LIN mode.
5	Reserved, should be cleared.
4 PMSK	Parity Mask. Forces bit 7 in the Data Register to 0 on reads. This can be used to mask the parity bit in application which use 7 data bits and 1 parity bit.
3 ORIE	Overrun Error Interrupt Enable. Generates an interrupt, when Overrun flag is set.
2 NFIE	Noise Flag Interrupt Enable. Generates an interrupt, when Noise Flag is set.
1 FEIE	Frame Error Interrupt Enable. Generates an interrupt, when a Frame Error is detected.
0 PFIE	Parity Flag Interrupt Enable. Generates an interrupt, when Parity Flag is set.

31.9.2.6 SCI Data Registers

Reading this register accesses eSCI receive data register. Writing accesses eSCI transmit data register; writing to R8 has no effect; register is not writable in LIN mode

Address Offset: 0x6 Access: User read/write

	7	6	5	4	3	2	1	0
R	R8	T8	0	0	0	0	0	0
W								
Reset	0	0	0	0	0	0	0	0

Figure 31-8. SCI Data Register High (SCIDRH)

Address Offset: 0x7 Access: User read/write

	7	6	5	4	3	2	1	0
R	R7	R6	R5	R4	R3	R2	R1	R0
W	T7	T6	T5	T4	T3	T2	T1	T0
Reset	0	0	0	0	0	0	0	0

Figure 31-9. SCI Data Register Low (SCIDRL)

Table 31-10. SCIDRH/L Field Descriptions

Field	Description
7 R8	Received Bit 8. R8 is the ninth data bit received when the eSCI is configured for 9-bit data format (M = 1).
6 T8	Transmit Bit 8. T8 is the ninth data bit transmitted when the eSCI is configured for 9-bit data format (M = 1).
5–0	Reserved, should be cleared.
7–0 R*n*	Received bits seven through zero for 9-bit or 8-bit data formats
7–0 T*n*	Transmit bits seven through zero for 9-bit or 8-bit formats

NOTES

If the value of T8 is the same as in the previous transmission, T8 does not have to be rewritten.The same value is transmitted until T8 is rewritten

In 8-bit data format, only SCI data register low (SCIDRL) needs to be accessed.

When transmitting in 9-bit data format and using 8-bit write instructions, write first to SCI data register high (SCIDRH), then SCIDRL.

To increase performance when using 9-bit data format, it is recommended to access the register with 16-bit reads and writes.

This register should not be used in LIN mode, writes to the register will be blocked in LIN mode.

Even if parity generation/checking is enabled via PE in the SCICR1 register, the parity bit will not be masked out. The bit 7 of the received data will be masked out if the PMSK bit in SCICR4 is set, however.

31.9.2.7 SCI Status Register 1

The SCISR1 and SCISR2 registers indicate the current status. The status flags can be polled, and can also be used to generate interrupts. The flags can be cleared by writing a 1 to them.

Address Offset: 0x8 Access: User read (write to clear)

	7	6	5	4	3	2	1	0
R W	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF
Reset	0	0	0	0	0	0	0	0

Figure 31-10. SCI Status Register 1 (SCISR1)

Table 31-11. SCISR1 Field Descriptions

Field	Description
7 TDRE	Transmit Data Register Empty Flag. TDRE is being set when the transmit data register (SCIDRH/L) becomes empty and can receive a new value to transmit. TDRE is 0 immediately after reset, a short time after the module is turned on (setting MDIS=0) the eSCI will become ready, and TDRE will be set. Clear TDRE by writing it with 1.
6 TC	Transmit Complete Flag. TC is being set after a byte of data, a preamble, or a break character has been transmitted and no data is queued via the SCI Data Register (SCIDR, Figure 31-9). When TC is set it indicates that the TXD out signal has become idle (logic 1). Clear TC by writing it with 1. After the device is switched on (by clearing the MDIS bit, see SCICR3 Figure 31-6) a preamble is transmitted - if no byte is written to the the SCI Data Register then the completion of the preamble can be monitored using the TC flag.
5 RDRF	Receive Data Register Full Flag. RDRF is being set when the SCI data register is loaded with new received data. Clear RDRF by writing it with 1.
4 IDLE	Idle Line Flag. IDLE is being set when 10 consecutive logic 1s (if M=0) or 11 consecutive logic 1s (if M=1) have appeared on the receiver input. Once the IDLE flag is cleared, a valid frame must again set the RDRF flag before an idle condition can set the IDLE flag. Clear IDLE by writing it with 1. **Note:** When the receiver wakeup bit (RWU) is set, an idle line condition does not set the IDLE flag.
3 OR	Overrun Flag. OR is set when software fails to read the SCI data register before the receive shift register receives the next frame. The OR bit is set immediately after the stop bit has been completely received for the second frame. The data in the shift register is lost, but the data already in the SCI data registers is not affected. Clear OR by writing it with 1.
2 NF	Noise Flag. NF is set when the eSCI detects noise on the receiver input. NF bit is set during the same cycle as the RDRF flag but does not get set in the case of an overrun. Clear NF by writing it with 1.
1 FE	Framing Error Flag. FE is set when a logic 0 is accepted as the stop bit. FE bit is set during the same cycle as the RDRF flag but does not get set in the case of an overrun. Clear FE by writing it with 1. **Note:** In LIN mode this bit will not be set when break characters are transmitted.
0 PF	Parity Error Flag. PF is set when the parity enable bit PE is set and the parity of the received data does not match its parity bit. Clear PF by writing it with 1.

31.9.2.8 SCI Status Register 2

Address Offset: 0x9 Access: User read/write (write to clear)

	7	6	5	4	3	2	1	0
R	0	0	0	BERR	0	0	0	RAF
W								
Reset	0	0	0	0	0	0	0	0

Figure 31-11. SCI Status Register 2 (SCISR2)

Table 31-12. SCISR2 Field Descriptions

Field	Description
7–5	Reserved, should be cleared.
4 BERR	Bit Error. Indicates a Bit on the bus did not match the transmitted Bit. Checking happens after a complete byte has been transmitted and received again. (A Bit Error will cause the LIN FSM to reset.) This bit is only used for LIN mode. If an unrequested byte is received (i.e. a byte which is not part of an RX frame) which is not recognized as a wakeup flag then this flag is also set. (Since the data on the RX line does not match the idle state which was assigned to the TX line.) The flag is set when the condition is detected, cleared by a write with 1.
3–1	Reserved, should be cleared.
0 RAF	Receiver Active Flag. RAF is set when the receiver detects a logic 0 during the RT1 time period of the start bit search. RAF is cleared when the receiver detects an idle character. Writing RAF has no effect 0 No reception in progress 1 Reception in progress

31.9.2.9 LIN Status Register 1

Address Offset: 0xA Access: User read/write (write to clear)

	7	6	5	4	3	2	1	0
R	RXRDY	TXRDY	LWAKE	STO	PBERR	CERR	CKERR	FRC
W								
Reset	0	0	0	0	0	0	0	0

Figure 31-12. LIN Status Register 1 (LINSTAT1)

Table 31-13. LINSTAT1 Field Descriptions

Field	Description
7 RXRDY	Receive Data Ready. The eSCI has received LIN data. This bit is set when the LINRX register receives a byte. Clear RXRDY by writing it with 1.
6 TXRDY	Transmit Data Ready. The LIN FSM can accept another write to LINTX. This bit is set when the LINTX register becomes free. TXRDY is 0 immediately after reset, a short time after the module is turned on (setting MDIS=0) the eSCI will become ready, and TXRDY will be set. Clear TXRDY by writing it with 1.

Table 31-13. LINSTAT1 Field Descriptions (Continued)

Field	Description
5 LWAKE	Received LIN 1.x Wakeup Signal. A LIN slave has sent a LIN 1.x wakeup signal (80h, 00h or C0h) on the bus. When this signal is detected, the LIN FSM will reset. If the setup of a frame had already started, the setup therefore needs to be repeated. When using this feature the guidelines in Section 31.10.11.5, "LIN Wakeup," should be noted. This flag will also be set if the ESCI receives a LIN 2.0 Wakeup Signal under the conditions described in Section 31.10.11.5, "LIN Wakeup." The flag is set when the condition is detected, cleared by a write with 1.
4 STO	Slave TimeOut. Represents a NO_RESPONSE_ERROR - this is set if a slave doesn't complete a frame within the specified maximum frame length. For LIN 1.3 the formula $T_{FRAME\ MAX} = (10 \cdot N_{DATA} + 45) \times 1.4$ ***Eqn. 31-1*** is used (N_{DATA} = number of data bytes in a frame, please refer to the LIN specification for details). The flag is set when the condition is detected, cleared by a write with 1.
3 PBERR	Physical Bus Error. No valid message can be generated on the bus. This is set if after the start of a byte transmission the input remains unchanged for 31 cycles. If the line toggles at least once during the transmission, the bit will not be set. A PBERR condition will reset the LIN FSM. The flag is set when the condition is detected, cleared by a write with 1.
2 CERR	CRC Error. The CRC pattern received with an extended frame was not correct. The flag is set when the condition is detected, cleared by a write with 1.
1 CKERR	Checksum Error. Checksum Error on a received frame. The flag is set when the condition is detected, cleared by a write with 1.
0 FRC	Frame Complete. LIN frame completely transmitted / All LIN data bytes (including checksum/CRC if applicable) received. The flag is set when the condition is detected, cleared by a write with 1.

31.9.2.10 LIN Status Register 2

Address Offset: 0x8 Access: User read/write (write to clear)

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	UREQ	OVFL
W								
Reset	0	0	0	0	0	0	0	0

Figure 31-13. LIN Status Register 2 (LINSTAT2)

Table 31-14. LINSTAT2 Field Descriptions

Field	Description
7–2	Reserved, should be cleared.

Table 31-14. LINSTAT2 Field Descriptions (Continued)

Field	Description
1 UREQ	Unrequested Data on LIN Bus. Unrequested activity has been detected on the LIN bus. Since the eSCI is used as a master node, this is normally an error condition. The flag is not set if the activity is identified as a wakeup or break character. In addition the RXRDY flag will also be set, the LINRX register must be read before normal operations can proceed. The flag is set when the condition is detected, cleared by a write with 1.
0 OVFL	RX Register Overflow. The RX register hasn't been read before a new data byte, CRC or checksum byte has been received from the LIN bus. The flag is set when the condition is detected, cleared by a write with 1.

31.9.2.11 LIN Control Registers

Address Offset: 0xC Access: User read/write (with no ongoing transmissions)

	7	6	5	4	3	2	1	0
R	LRES	0	WUD1	WUD0	LDBG	DSF	PRTY	LIN
W		WU						
Reset	0	0	0	0	0	0	0	0

Figure 31-14. LIN Control Register 1 (LINCTRL1)

Table 31-15. LINCTRL1 Field Descriptions

Field	Description
7 LRES	LIN Resync. Causes the LIN protocol engine to return to start state. This happens automatically after bit errors, but software can force that behaviour manually via this bit. The bit needs to be first set then cleared, so that the protocol engine is operational again. Status flags will not be affected by this bit and will need to be cleared manually, if desired.
6 WU	LIN Bus WakeUp. Generates a wakeup signal on the LIN bus. This needs to be send before a transmission, if the bus is in sleep mode, e.g. because it has been idle for 25000 bit times. This bit will auto-clear, so a read from this bit will always return 0. When using this feature the guidelines in Section 31.10.11.5, "LIN Wakeup," should be noted.
5–4 WUD*n*	WakeUp Delimiter Time. Determines how long the LIN engine waits after generating a wakeup signal, before starting a new frame. The eSCI will not set TXRDY before this time expires. Note that in addition to this delimiter time, the CPU and the eSCI will require some setup time to start a new transmission, typically there'll be an additional bit time delay.
3 LDBG	LIN Debug Mode. Prevents the LIN FSM from automatically resetting, after an exception (Bit Error, Physical Bus Error, Wakeup Flag) has been detected. This is for debug purposes only.

WUD1	WUD0	bit times
0	0	4
0	1	8
1	0	32
1	1	64

Table 31-15. LINCTRL1 Field Descriptions (Continued)

Field	Description
2 DSF	Double Stop Flags. When a bit error has been detected, this will add an additional stop flag to the byte in which the error occurred.
1 PRTY	Activating Parity Generation. Generate the two Parity Bits in the LIN header.
0 LIN	LIN Mode. Switch device into LIN mode. **Note:** In LIN mode data transfer should be done only via the LIN interface. The SBK bit (in SCICR2) and the SCI data registers (SCIDRH/L) should not be used. Similarly all data transfer handshaking should be done via the LIN interface, the TDRE, TC and RDRF flags in the SCISR1 should not be used for that purpose.

Address Offset: 0xD Access: User read/write (with no ongoing transmissions)

	7	6	5	4	3	2	1	0
R W	RXIE	TXIE	WUIE	STIE	PBIE	CIE	CKIE	FCIE
Reset	0	0	0	0	0	0	0	0

Figure 31-15. LIN Control Register 2 (LINCTRL2)

Table 31-16. LINCTRL2 Field Descriptions

Field	Description
7 RXIE	LIN RXREG Ready Interrupt Enable. Generates an Interrupt when new data is available in the LIN RXREG.
6 TXIE	LIN TXREG Ready Interrupt Enable. Generates an Interrupt when new data can be written to the LIN TXREG.
5 WUIE	RX WakeUP Interrupt Enable. Generates an Interrupt when a wakeup flag from a LIN 1.x slave has been received. When using this feature the guidelines in Section 31.10.11.5, “LIN Wakeup,” should be followed.
4 STIE	Slave Timeout Error Interrupt Enable. Generates an Interrupt when the slave response is too slow.
3 PBIE	Physical Bus Error Interrupt Enable. Generates an Interrupt when no valid message can be generated on the bus.
2 CIE	CRC Error Interrupt Enable. Generates an Interrupt when a CRC error on a received extended frame is detected.
1 CKIE	Checksum Error Interrupt Enable. Generates an Interrupt on a detected Checksum Error.
0 FCIE	Frame Complete Interrupt Enable. Generates an Interrupt after complete transmission of a TX frame, or after the last byte of an RX frame is received. (The complete frame includes all header, data, CRC and checksum bytes as applicable.)

Address Offset: 0xE Access: User read/write (with no ongoing transmissions)

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	UQIE	OFIE
W								
Reset	0	0	0	0	0	0	0	0

Figure 31-16. LIN Control Register 3 (LINCTRL3)

Table 31-17. LINCTRL3 Field Descriptions

Field	Description
7–2	Reserved, should be cleared.
1 UQIE	Unrequested Data Interrupt Enable. Generates an Interrupt when unrequested data has been observed on the LIN bus.
0 OFIE	Overflow Interrupt Enable. Generates an Interrupt when a data byte in the LINRX register hasn't been read before the next databyte is received.

31.9.2.12 LIN TX Register

The first byte written to the register selects the transmit address, the second byte determines the frame length, the third and fourth byte set various frame options and determine the timeout counter. (Header parity will be automatically generated if the PRTY bit is set.) For TX frames the fourth byte (bits T7 - T0) will be skipped, since the timeout function does not apply. All following bytes are data bytes for the frame. CRC and Checksum bytes will be automatically appended when the appropriate options are selected.

When a Bit error is detected an interrupt is set, and the transmission aborted. The register can only be written again once the interrupt is cleared. Afterwards a new frame starts, and the first byte needs to contain a header again.

Additionally it is possible to flush the LINTX register by writing to the LRES bit.

NOTE

Not all values written to the LINTX registers will generate valid LIN frames. The values need to be

determined according to the LIN specification.

Address Offset: 0x10 Access: User write (when TXRDY is set)

Byte	7	6	5	4	3	2	1	0
R								
First W	P1	P0	ID5	ID4	ID3	ID2	ID1	ID0
Second W	L7	L6	L5	L4	L3	L2	L1	L0
Third W	HDCHK	CSUM	CRC	TX	T11	T10	T9	T8
Fourth W	T7	T6	T5	T4	T3	T2	T1	T0
Fifth+ W	D7	D6	D5	D4	D3	D2	D1	D0
Reset	0	0	0	0	0	0	0	0

Figure 31-17. LIN TX Register (LINTX)

Table 31-18. LINTX Field Descriptions

Field	Description
ID*n*	Header Bit *n.* The LIN address, for LIN 1.x standard frames the length bits need to be set appropriately (see the table below), extended frames will be recognized by their specific patterns. **ID5** / **ID4** / **data bytes**: 0/0/2; 0/1/2; 1/0/4; 1/1/8 **Note:** The values 3C, 3D, 3E and 3F of the ID-field (ID0-5) indicate command and extended frames.
P*n*	Parity Bit *n.* When enabled, the parity bits can be generated automatically (PRTY bit in register LINCTRL1, see Figure 31-16). Otherwise they can be provided here.
L*n*	Length Bit *n.* Defines the length of the frame - 0 to 255 data bytes - this information is needed by the LIN state machine in order to insert the checksum or CRC pattern as required. LIN 1.x slaves will only accept frames with 2, 4 or 8 data bytes.
HDCHK	Header Checksum Enable. Include the header fields into the mod 256 checksum of the standard frames.
CSUM	Checksum Enable. Append a checksum byte to the end of a TX frame - verify the checksum byte of a RX frame.
CRC	CRC Enable. Append two CRC bytes to the end of a TX frame - verify the two CRC bytes of a RX frame are correct. If both CSUM and CRC bits are set, the LIN FSM will first append the CRC bytes, then the checksum byte, and will expect them in this order, as well. If HDCHK is set, the CRC calculation will include header and data bytes, otherwise just the data bytes. CRC bytes are not part of the LIN standard - they are normal data bytes and belong to a higher-level protocol.
TX	Transmit Direction. Indicates a TX frame i.e. the eSCI will transmit data to a slave. Otherwise an RX frame will be assumed, and the eSCI will only transmit the header, the data bytes will be received from the slave. 0 RX frame 1 TX frame

Table 31-18. LINTX Field Descriptions (Continued)

Field	Description
T*n*	Timeout Bit *n.* Sets the counter to determine a NO_RESPONSE_ERROR, if the frame is a read access to a LIN slave - according to LIN standard rev 1.3 the value needs to be $(10 \cdot N_{DATA} + 45) \times 1.4$ (N_{DATA} is the number of data bytes in the frame) ***Eqn. 31-2*** The counter value represent the maximum time available for a complete RX frame (please refer to the LIN specification for details). For transmissions the accessible timeout bits have to be set to 0. The timeout bits 7-0 will not be written on a TX frame, at all. So for TX frames the 4th byte written to the TX register is the first data byte, for RX it contains timeout bits 7-0. The time is specified in multiples of Bit times. The timeout period starts with the transmission of the LIN break character.
D*n*	Data Bit *n.* Data bits for transmission.

31.9.2.13 LIN RX Register

Address Offset: 0x14 Access: User read (when RXRDY is set)

	7	6	5	4	3	2	1	0
R	D7	D6	D5	D4	D3	D2	D1	D0
W								
Reset	0	0	0	0	0	0	0	0

Figure 31-18. LIN RX Register (LINRX)

Table 31-19. LINRX Field Descriptions

Field	Description
7–0 D*n*	Data Bit *n.* This register will provide received data bytes from RX frames. It is only valid when the RXRDY flag is set, CRC and checksum information will not be available in the RX register unless they are treated as data. It is possible to treat CRC and checksum bytes as data, by deactivating the CSUM respectively CRC control bits in the LINTX register - however then CRC and CSUM checking has to be performed by software.

NOTE

The application software must ensure that LINRX is read before new Bytes (data Bytes, CRC or checksum Bytes) are received from the LIN bus. Similarly if data is available in LINRX, the register must be read before starting a new frame.

31.9.2.14 LIN CRC Polynomial Register

Address Offset: 0x18 Access: User read/write (write with no ongoing transmissions)

	7	6	5	4	3	2	1	0
R W	P15	P14	P13	P12	P11	P10	P9	P8
Reset	1	1	0	0	0	1	0	1

Figure 31-19. LIN CRC Polynomial Register 1 (LINCRCP1)

Address Offset: 0x19 Access: User read/write (write with no ongoing transmissions)

	7	6	5	4	3	2	1	0
R W	P7	P6	P5	P4	P3	P2	P1	P0
Reset	1	0	0	1	1	0	0	1

Figure 31-20. LIN CRC Polynomial Register 2 (LINCRCP2)

Table 31-20. LINCRCP1–2 Field Descriptions

Field	Description
7–0 P*n*	Polynomial bit x^n. Used to define the LIN polynomial - standard is $x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1$ (the polynomial used for the CAN protocol).

31.10 Functional Description

31.10.1 General

This section provides a complete functional description of the eSCI block, detailing the operation of the design from the end user perspective in a number of subsections.

Figure 31-21 shows the structure of the eSCI module. The eSCI allows full duplex, asynchronous, NRZ serial communication between the CPU and remote devices, including other CPUs. The eSCI transmitter and receiver operate independently, although they use the same baud rate generator. The CPU monitors the status of the eSCI, writes the data to be transmitted, and processes received data.

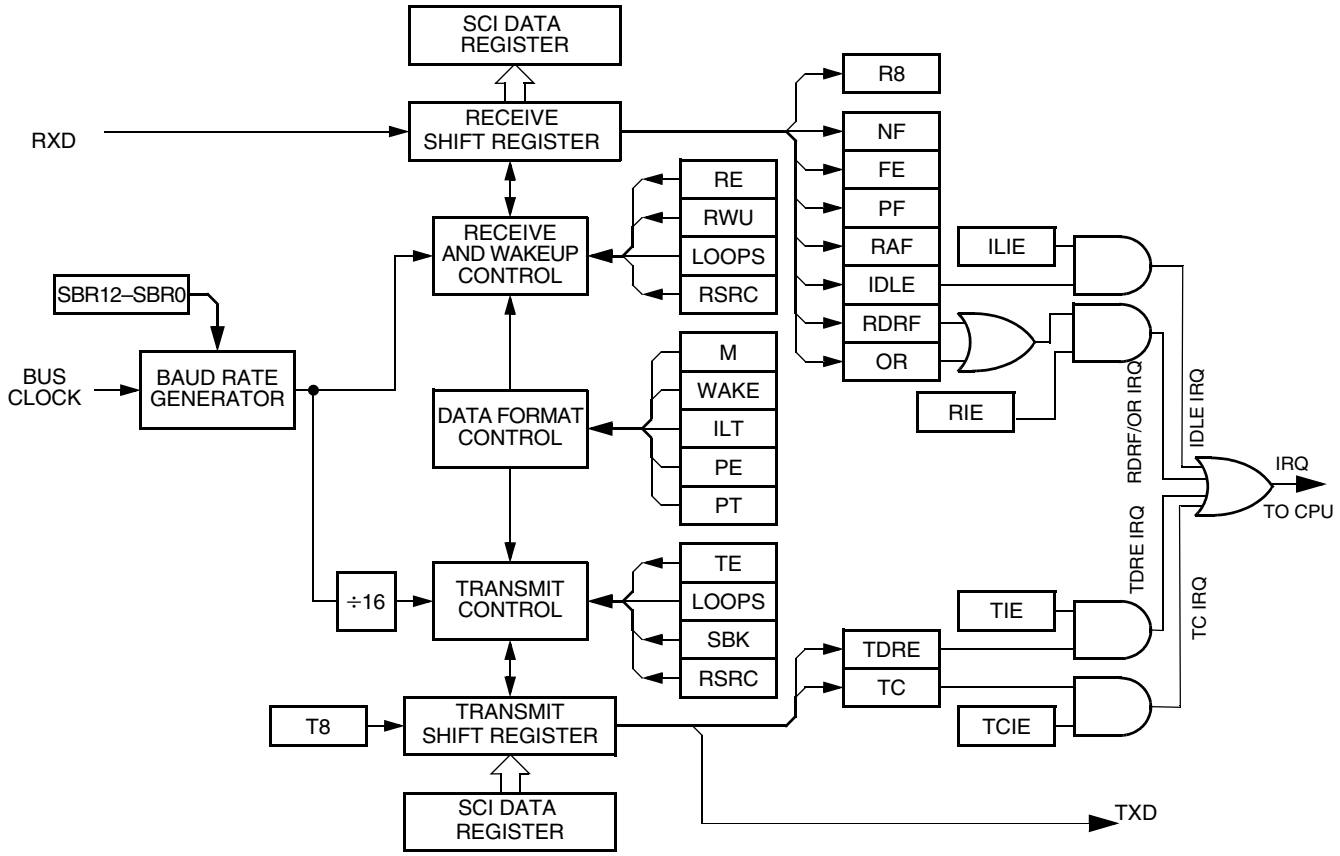


Figure 31-21. eSCI Block Diagram

31.10.2 Data Format

The eSCI uses the standard NRZ mark/space data format illustrated in Figure 31-22 below.

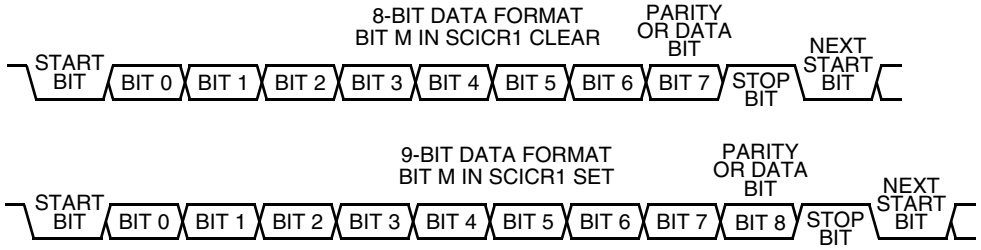


Figure 31-22. SCI Data Formats

Each data character is contained in a frame that includes a start bit, eight or nine data bits, and a stop bit. Clearing the M bit in SCI control register 1 configures the eSCI for 8-bit data characters.A frame with eight data bits has a total of 10 bits. Setting the M bit configures the eSCI for nine-bit data characters. A frame with nine data bits has a total of 11 bits.

Table 31-21. Example of 8-bit Data Formats

Start Bit	Data Bits	Address Bits	Parity Bits	Stop Bit
1	8	0	0	1
1	7	0	1	1
1	7	1[1]	0	1

1. The address bit identifies the frame as an address character. See Section 31.10.5.6, “Receiver Wakeup.”

When the eSCI is configured for 9-bit data characters, the ninth data bit is the T8 bit in SCI data register high (SCIDRH). It remains unchanged after transmission and can be used repeatedly without rewriting it. A frame with nine data bits has a total of 11 bits.

Table 31-22. Example of 9-Bit Data Formats

Start Bit	Data Bits	Address Bits	Parity Bits	Stop Bit
1	9	0	0	1
1	8	0	1	1
1	8	1[1]	0	1

1. The address bit identifies the frame as an address character. See Section 31.10.5.6, “Receiver Wakeup.”

31.10.3 Baud Rate Generation

A 13-bit modulus counter in the baud rate generator derives the baud rate for both the receiver and the transmitter. The value from 0 to 8191 written to the SBR12–SBR0 bits determines the module clock divisor. The SBR bits are in the SCI baud rate registers (SCIBDH and SCIBDL). The baud rate clock is synchronized with the bus clock and drives the receiver. The baud rate clock divided by 16 drives the transmitter. The receiver has an acquisition rate of 16 samples per bit time.

Baud rate generation is subject to one source of error:

- Integer division of the module clock may not give the exact target frequency.

Figure 31-23 lists some examples of achieving target baud rates with a module clock frequency of 10.2 MHz.

$$\text{SCI baud rate} = \frac{\text{SCI module clock}}{16 \times \text{SCIBR[12:0]}} \qquad \textit{Eqn. 31-3}$$

Table 31-23. Baud Rates (Example: Module Clock = 10.2 Mhz)

Bits SBR[12-0]	Receiver Clock (Hz)	Transmitter Clock (Hz)	Target Baud Rate	Error (%)
17	600,000.0	37,500.0	38,400	2.3
33	309,090.9	19,318.2	19,200	.62

Table 31-23. Baud Rates (Example: Module Clock = 10.2 Mhz) (Continued)

Bits SBR[12-0]	Receiver Clock (Hz)	Transmitter Clock (Hz)	Target Baud Rate	Error (%)
66	154,545.5	9659.1	9600	.62
133	76,691.7	4793.2	4800	.14
266	38,345.9	2396.6	2400	.14
531	19,209.0	1200.6	1200	.11
1062	9604.5	600.3	600	.05
2125	4800.0	300.0	300	.00
4250	2400.0	150.0	150	.00
5795	1760.1	110.0	110	.00

31.10.4 Transmitter

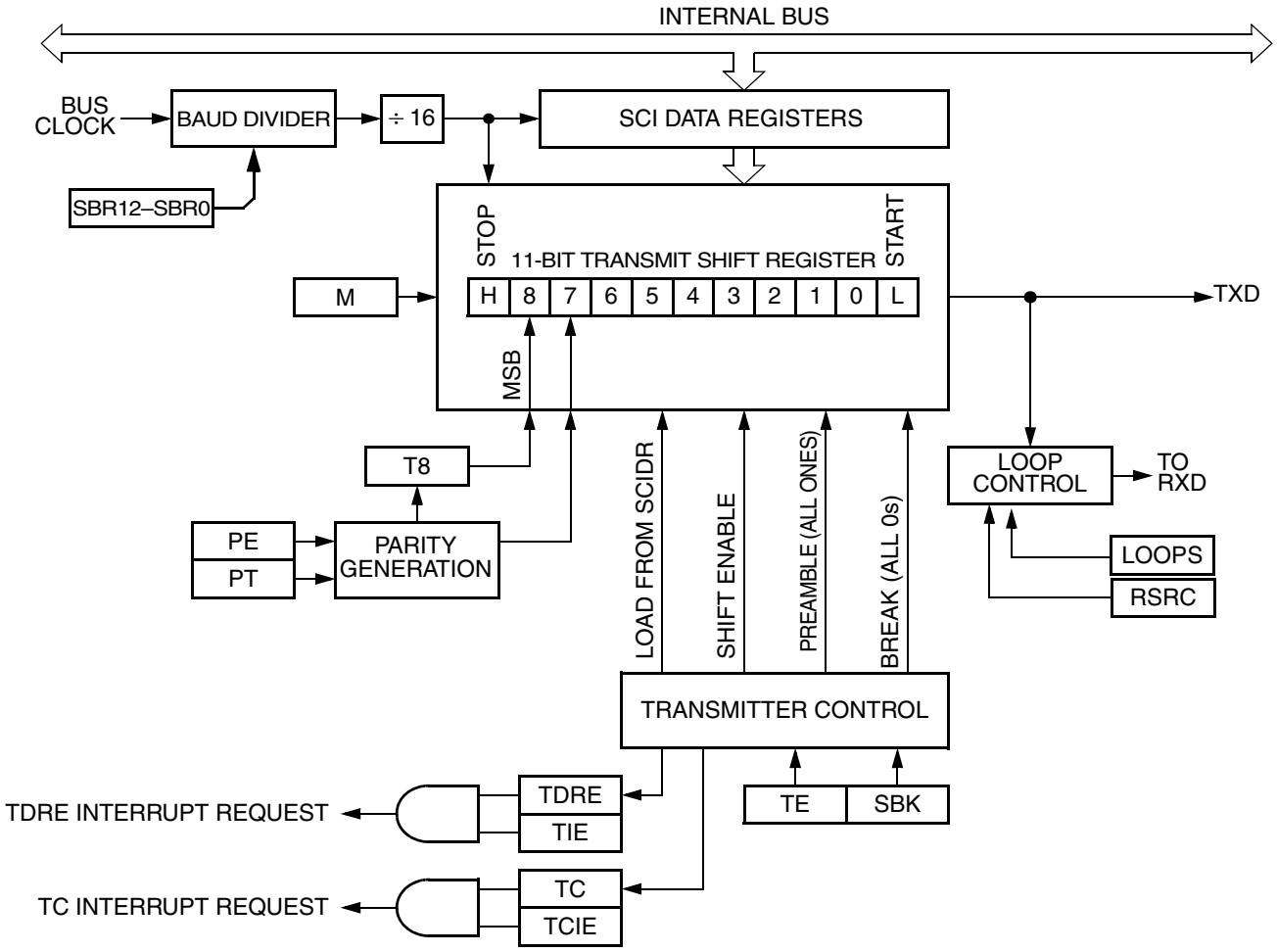


Figure 31-23. Transmitter Block Diagram

31.10.4.1 Transmitter Character Length

The eSCI transmitter can accommodate either 8-bit or 9-bit data characters. The state of the M bit in SCI control register 1 (SCICR1) determines the length of data characters. When transmitting 9-bit data, bit T8 in SCI data register high (SCIDRH) is the ninth bit (bit 8).

31.10.4.2 Character Transmission

To transmit data, the MCU writes the data bits to the SCI data registers (SCIDRH/SCIDRL), which in turn are transferred to the transmitter shift register. The transmit shift register then shifts a frame out through the **Tx output** signal, after it has prefaced them with a start bit and appended them with a stop bit. The SCI data registers (SCIDRH and SCIDRL) are the write-only buffers between the internal data bus and the transmit shift register.

The eSCI also sets a flag, the transmit data register empty flag (TDRE), every time it transfers data from the buffer (SCIDRH/L) to the transmitter shift register.The transmit driver routine may respond to this flag by writing another byte to the Transmitter buffer (SCIDRH/SCIDRL), while the shift register is still shifting out the first byte.

To initiate an SCI transmission:

12. Configure the eSCI:
    a) Turn on the module by writing the MDIS bit to 0.
    b) Select a baud rate. Write this value to the SCI baud registers (SCIBDH/L) to start the baud rate generator. Remember that the baud rate generator is disabled when the baud rate is zero. Writing to the SCIBDH has no effect without also writing to SCIBDL.
    c) Write to SCICR1 to configure word length, parity, and other configuration bits (LOOPS, RSRC, M, WAKE, ILT, PE, PT).
    d) Enable the transmitter, interrupts, receive, and wake up as required, by writing to the SCICR2 register bits (TIE, TCIE, RIE, ILIE, TE, RE, RWU, SBK). A preamble or idle character will now be shifted out of the transmitter shift register.
13. Transmit Procedure for Each Byte:
    a) Poll the TDRE flag by reading the SCISR1 or responding to the TDRE interrupt. Keep in mind that the TDRE bit resets to one.
    b) If the TDRE flag is set, write the data to be transmitted to SCIDRH/L, where the ninth bit is written to the T8 bit in SCIDRH if the eSCI is in 9-bit data format.
14. Repeat step 2 for each subsequent transmission.

NOTE

The TDRE flag is set when the shift register is loaded with the next data to be transmitted from SCIDRH/L, which happens, generally speaking, a little over half-way through the stop bit of the previous frame. Specifically, this transfer occurs 9/16ths of a bit time AFTER the start of the stop bit of the previous frame.

Writing the TE bit from 0 to a 1 automatically loads the transmit shift register with a preamble of 10 logic 1s (if M = 0) or 11 logic 1s (if M = 1). After the preamble shifts out, control logic transfers the data from

the SCI data register into the transmit shift register. A logic 0 start bit automatically goes into the least significant bit position of the transmit shift register. A logic 1 stop bit goes into the most significant bit position.

Hardware supports odd or even parity. When parity is enabled, the most significant bit (msb) of the data character is the parity bit.

The transmit data register empty flag, TDRE, in SCI status register 1 (SCISR1) becomes set when the SCI data register transfers a byte to the transmit shift register. The TDRE flag indicates that the SCI data register can accept new data from the internal data bus. If the transmit interrupt enable bit, TIE, in SCI control register 2 (SCICR2) is also set, the TDRE flag generates a transmitter interrupt request.

When the transmit shift register is not transmitting a frame, the Tx output signal goes to the idle condition, logic 1. If at any time software clears the TE bit in SCI control register 2 (SCICR2), the transmitter enable signal goes low and the transmit signal goes idle.

If software clears TE while a transmission is in progress (TC = 0), the frame in the transmit shift register continues to shift out. To avoid accidentally cutting off the last frame in a message, always wait for TDRE to go high after the last frame before clearing TE.

To separate messages with preambles with minimum idle line time, use this sequence between messages:

1. Write the last byte of the first message to SCIDRH/L.
2. Wait for the TDRE flag to go high, indicating the transfer of the last frame to the transmit shift register.
3. Queue a preamble by clearing and then setting the TE bit.
4. Write the first byte of the second message to SCIDRH/L.

31.10.4.3 Break Characters

Writing a logic 1 to the send break bit, SBK, in SCI control register 2 (SCICR2) loads the transmit shift register with a break character. A break character contains all logic 0s and has no start, stop, or parity bit. Break character length depends on the M bit in the SCI control register 1 (SCICR1) and on the BRK13 bit in the SCI control register 3 (SCICR3). As long as SBK is at logic 1, transmitter logic continuously loads break characters into the transmit shift register. After software clears the SBK bit, the shift register finishes transmitting the last break character and then transmits at least one logic 1. The automatic logic 1 at the end of a break character guarantees the recognition of the start bit of the next frame.

The eSCI recognizes a break character when a start bit is followed by eight or nine logic 0 data bits and a logic 0 where the stop bit should be. Receiving a break character has these effects on eSCI registers:

- Sets the framing error flag, FE
- Sets the receive data register full flag, RDRF
- Clears the SCI data registers (SCIDRH/L)
- May set the overrun flag, OR, noise flag, NF, parity error flag, PE, or the receiver active flag, RAF (see Section 31.9.2.7, “SCI Status Register 1,” and Section 31.9.2.8, “SCI Status Register 2.”)

31.10.4.4 Idle Characters

An idle character contains all logic 1s and has no start, stop, or parity bit. Idle character length depends on the M bit in SCI control register 1 (SCICR1). The preamble is a synchronizing idle character that begins the first transmission initiated after writing the TE bit from 0 to 1.

If the TE bit is cleared during a transmission, the Tx output signal becomes idle after completion of the transmission in progress. Clearing and then setting the TE bit during a transmission queues an idle character to be sent after the frame currently being transmitted.

NOTE

When queueing an idle character, return the TE bit to logic 1 before the stop bit of the current frame shifts out through the Tx output signal. Setting TE after the stop bit appears on Tx output signal causes data previously written to the SCI data register to be lost. Toggle the TE bit for a queued idle character while the TDRE flag is set and immediately before writing the next byte to the SCI data register.

31.10.4.5 Standard Bit Error Detection

Standard Bit Error Detection implements the Bit Error Detection algorithm that is normally used by a software implementation of the LIN protocol. During a transmit frame or while the header of a RX frame is sent, it reads the received byte and compares it to the one which was sent out. If there are any mismatches between the 8 transmitted databits and the 8 received databits, the BERR flag will be set.

31.10.4.6 Fast Bit Error Detection in LIN mode

Fast Bit Error Detection has been designed to allow flagging of LIN bit errors while they occur, rather than flagging them after a byte transmission has completed (see Figure 31-24).

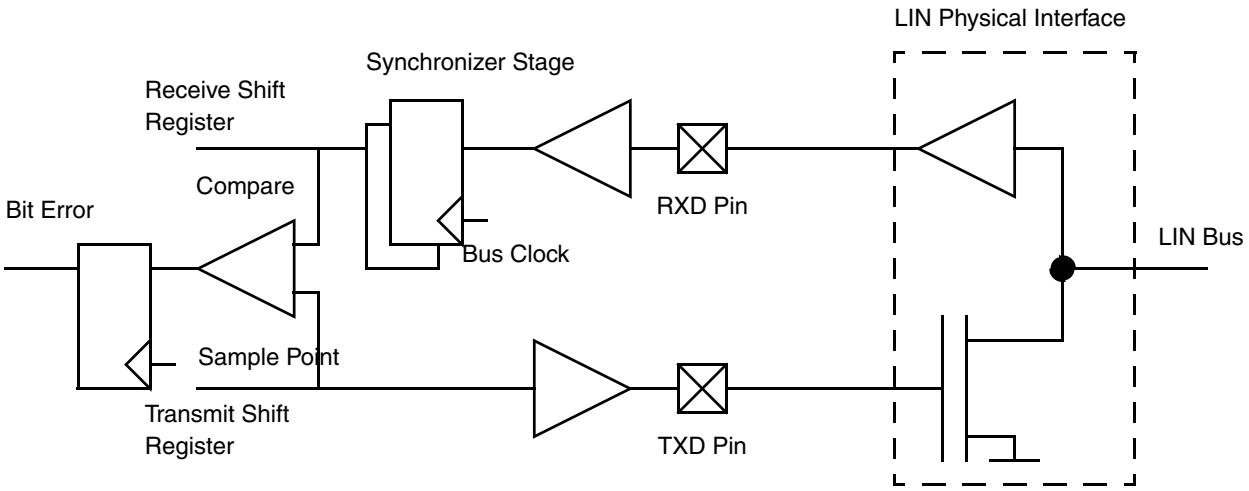


Figure 31-24. Fast Bit Error Detection on a LIN Bus

If fast bit error detection is enabled (FBR = 1) the eSCI will compare the transmitted and the received data stream while the transmitter is active (not idle). Once a mismatch between the transmitted data and the received data is detected the following actions are performed:

- The bit error flag BERR will be set
- if SBSTP is 0 the remainder of the byte will be transmitted normally
- if SBSTP is 1 the remaining bits in the byte after the error bit are transmitted as 1s (idle)

To adjust to different bus loads the sample point at which the incomming bit is compared to the one which was transmitted can be selected with the BESM13 bit (see Figure 31-25). If set the comparison will be performed at RT clock 13, otherwise at RT clock 9 (also see section Section 31.10.5.3, “Data Sampling).

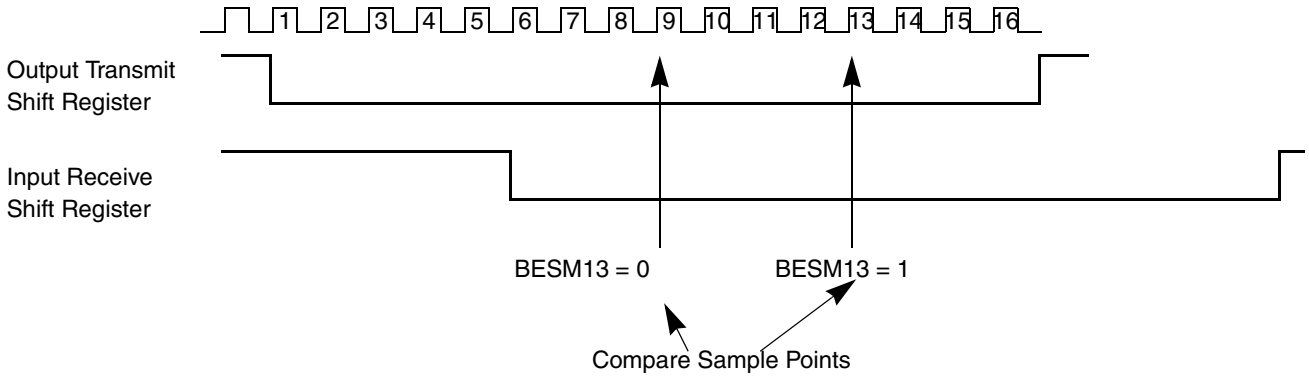


Figure 31-25. Timing Diagram Fast Bit Error Detection

NOTE

To calculate the exact position of the sample point with regard to the RX pin, the delays through the pads and the two Bus Clock cycle delay through the input synchronizer also needs to be taken into account.

31.10.5 Receiver

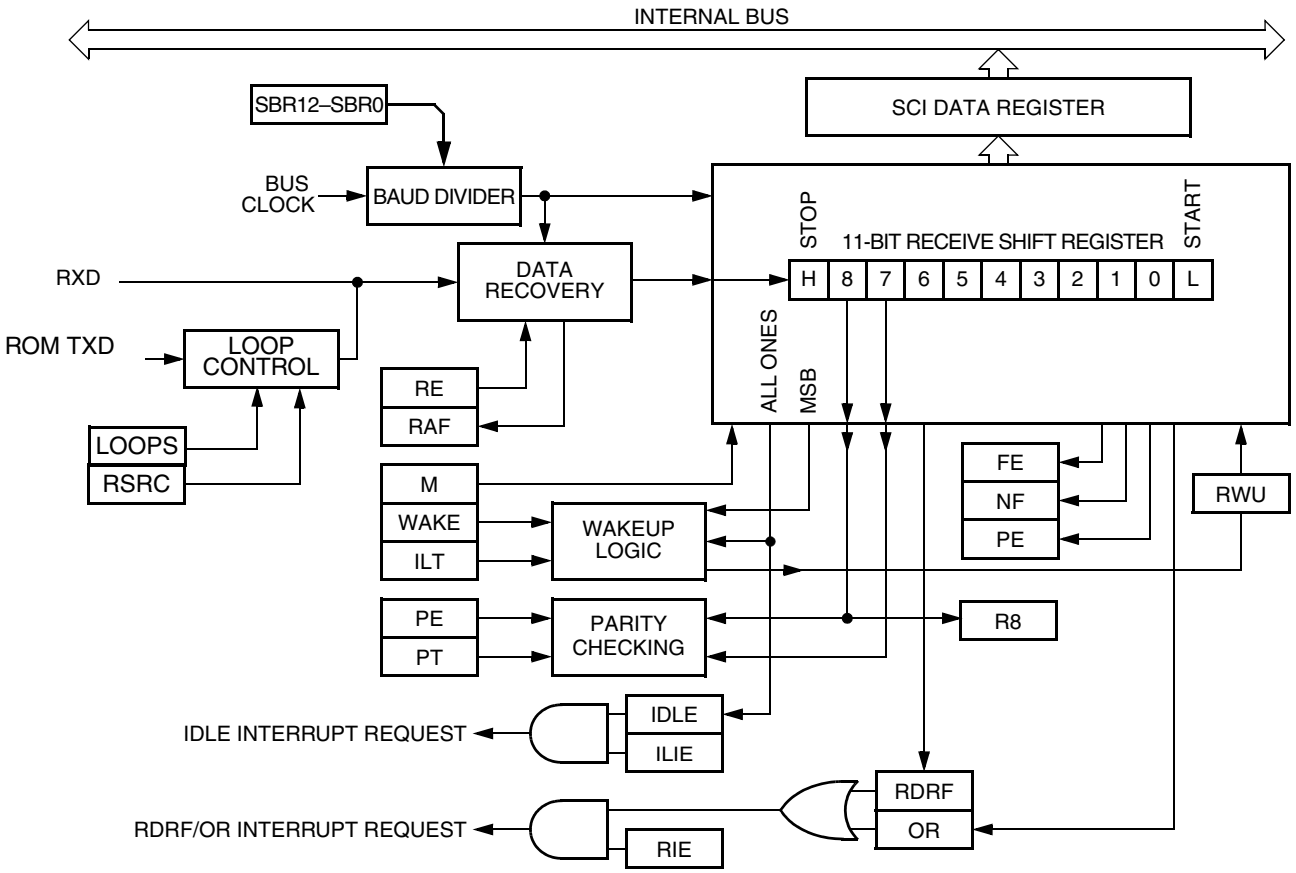


Figure 31-26. eSCI Receiver Block Diagram

31.10.5.1 Receiver Character Length

The eSCI receiver can accommodate either 8-bit or 9-bit data characters. The state of the M bit in SCI control register 1 (SCICR1) determines the length of data characters. When receiving 9-bit data, bit R8 in SCI data register high (SCIDRH) is the ninth bit (bit 8).

31.10.5.2 Character Reception

During an SCI reception, the receive shift register shifts a frame in from the Rx input signal. The SCI data register is the read-only buffer between the internal data bus and the receive shift register.

After a complete frame shifts into the receive shift register, the data portion of the frame transfers to the SCI data register. The receive data register full flag, RDRF, in SCI status register 1 (SCISR1) becomes set, indicating that the received byte can be read. If the receive interrupt enable bit, RIE, in SCI control register 2 (SCICR2) is also set, the RDRF flag generates an RDRF interrupt request.

If the RE bit is cleared the current frame is discarded, no receive flags will be set.

31.10.5.3 Data Sampling

The receiver samples the Rx input signal at the RT clock rate. The RT clock is an internal signal with a frequency 16 times the baud rate. To adjust for baud rate mismatch, the RT clock (see Figure 31-27) is re-synchronized:

- After every start bit
- After the receiver detects a data bit change from logic 1 to logic 0 (after the majority of data bit samples at RT8, RT9, and RT10 returns a valid logic 1 and the majority of the next RT8, RT9, and RT10 samples returns a valid logic 0)

To locate the start bit, data recovery logic does an asynchronous search for a logic 0 preceded by three logic 1s.When the falling edge of a possible start bit occurs, the RT clock begins to count to 16.

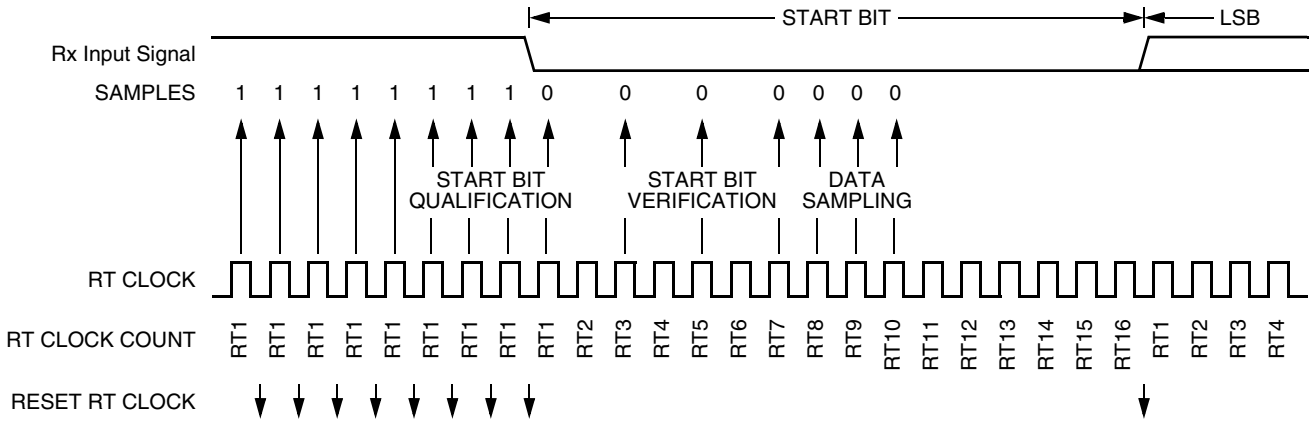


Figure 31-27. Receiver Data Sampling

To verify the start bit and to detect noise, data recovery logic takes samples at RT3, RT5, and RT7. Table 31-24 summarizes the results of the start bit verification samples.

Table 31-24. Start Bit Verification

RT3, RT5, and RT7 Samples	Start Bit Verification	Noise Flag
000	Yes	0
001	Yes	1
010	Yes	1
011	No	0
100	Yes	1
101	No	0
110	No	0
111	No	0

If start bit verification is not successful, the RT clock is reset and a new search for a start bit begins.

To determine the value of a data bit and to detect noise, recovery logic takes samples at RT8, RT9, and RT10. Table 31-25 summarizes the results of the data bit samples.

Table 31-25. Data Bit Recovery

RT8, RT9, and RT10 Samples	Data Bit Determination	Noise Flag
000	0	0
001	0	1
010	0	1
011	1	1
100	0	1
101	1	1
110	1	1
111	1	0

NOTE

The RT8, RT9, and RT10 samples do not affect start bit verification. If any or all of the RT8, RT9, and RT10 start bit samples are logic 1s following a successful start bit verification, the noise flag (NF) is set, however.

To verify a stop bit and to detect noise, recovery logic takes samples at RT8, RT9, and RT10. Table 31-26 summarizes the results of the stop bit samples.

Table 31-26. Stop Bit Recovery

RT8, RT9, and RT10 Samples	Framing Error Flag	Noise Flag
000	1	0
001	1	1
010	1	1
011	0	1
100	1	1
101	0	1
110	0	1
111	0	0

In Figure 31-28 the verification samples RT3 and RT5 determine that the first low detected was noise and not the beginning of a start bit. The RT clock is reset and the start bit search begins again. The noise flag is not set because the noise occurred before the start bit was found.

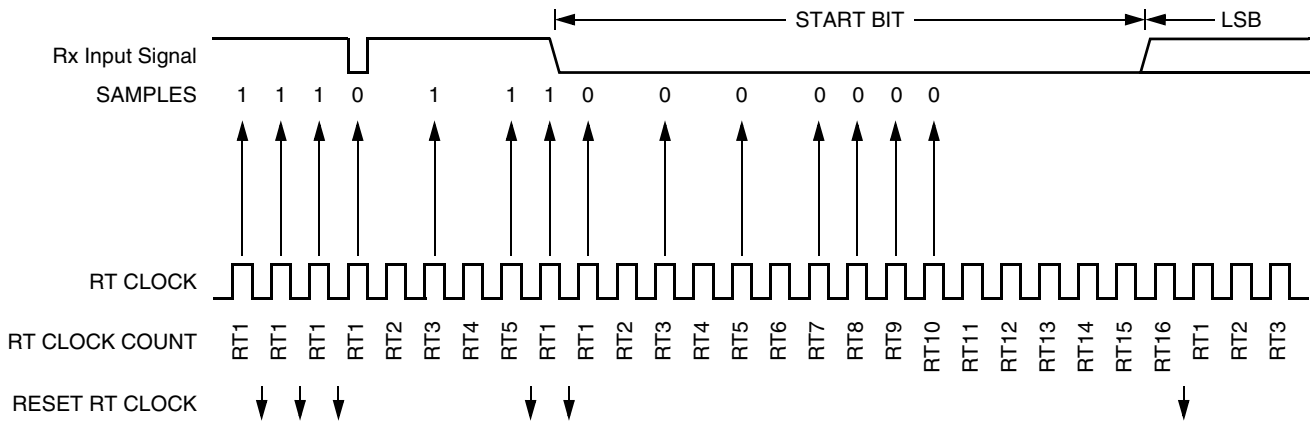


Figure 31-28. Start Bit Search Example

In Figure 31-29, verification sample at RT3 is high. The RT3 sample sets the noise flag. Although the perceived bit time is misaligned, the data samples RT8, RT9, and RT10 are within the bit time and data recovery is successful.

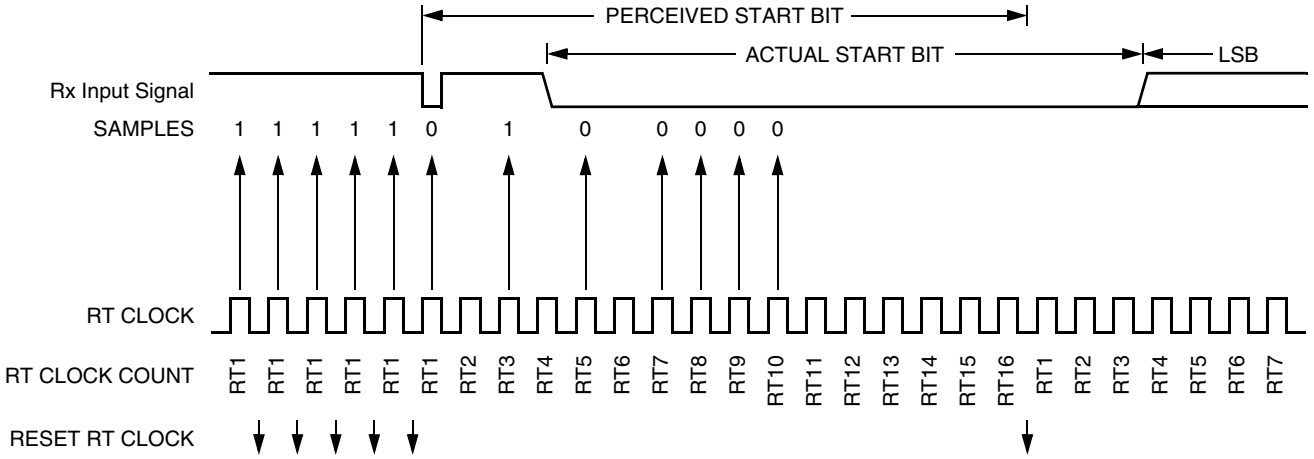


Figure 31-29. Start Bit Search Example 2

In Figure 31-30, a large burst of noise is perceived as the beginning of a start bit, although the test sample at RT5 is high. The RT5 sample sets the noise flag. Although this is a worst-case misalignment of perceived bit time, the data samples RT8, RT9, and RT10 are within the bit time and data recovery is successful.

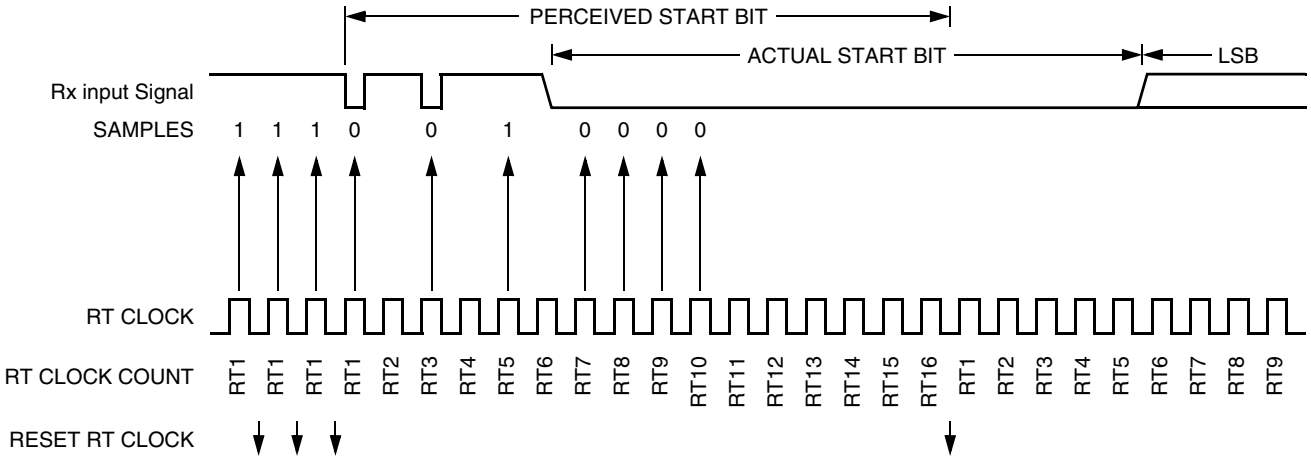


Figure 31-30. Start Bit Search Example 3

Figure 31-31 shows the effect of noise early in the start bit time. Although this noise does not affect proper synchronization with the start bit time, it does set the noise flag.

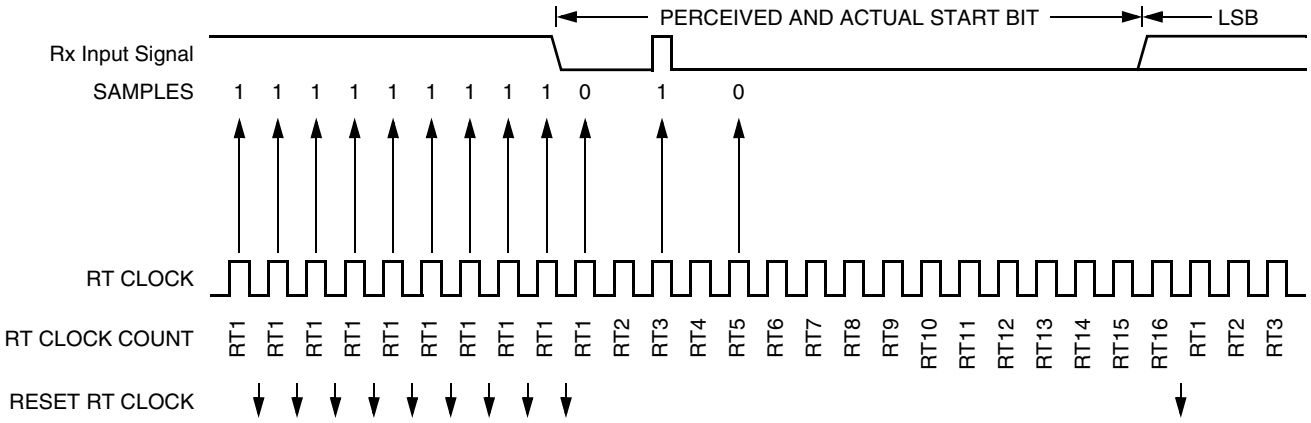


Figure 31-31. Start Bit Search Example 4

Figure 31-32 shows a burst of noise near the beginning of the start bit that resets the RT clock. The sample after the reset is low but is not preceded by three high samples that would qualify as a falling edge. Depending on the timing of the start bit search and on the data, the frame may be missed entirely or it may set the framing error flag.

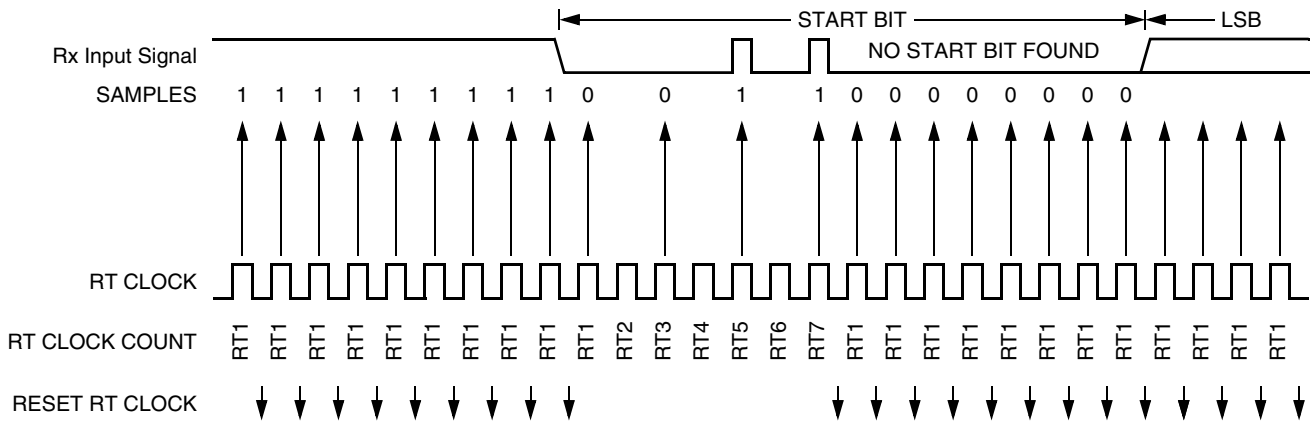


Figure 31-32. Start Bit Search Example 5

In Figure 31-33, a noise burst makes the majority of data samples RT8, RT9, and RT10 high. This sets the noise flag but does not reset the RT clock. In start bits only, the RT8, RT9, and RT10 data samples are ignored.

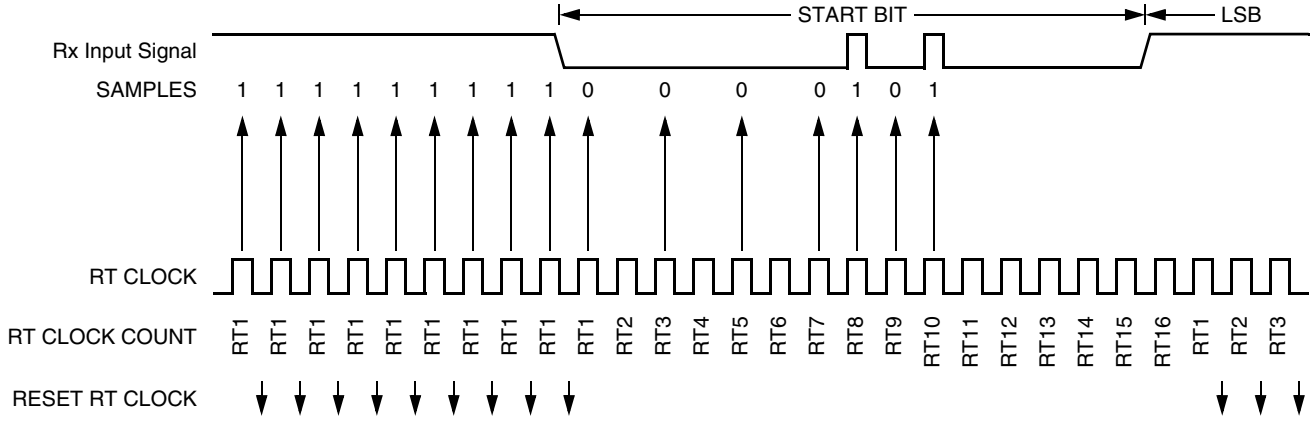


Figure 31-33. Start Bit Search Example 6

31.10.5.4 Framing Errors

If the data recovery logic does not detect a logic 1 where the stop bit should be in an incoming frame, it sets the framing error flag, FE, in SCI status register 1 (SCISR1). A break character also sets the FE flag because a break character has no stop bit.

31.10.5.5 Baud Rate Tolerance

A transmitting device may be operating at a baud rate below or above the receiver baud rate. Accumulated bit time misalignment can cause one of the three stop bit data samples (RT8, RT9, and RT10) to fall outside the actual stop bit.A noise error will occur if the RT8, RT9, and RT10 samples are not all the same logical values. A framing error will occur if the receiver clock is misaligned in such a way that the majority of the RT8, RT9, and RT10 stopbit samples are a logic zero.

As the receiver samples an incoming frame, it re-synchronizes the RT clock on any valid falling edge within the frame. Re-synchronization within frames will correct a misalignment between transmitter bit times and receiver bit times.

31.10.5.5.1 Slow Data Tolerance

Figure 31-34 shows how much a slow received frame can be misaligned without causing a noise error or a framing error. The slow stop bit begins at RT8 instead of RT1 but arrives in time for the stop bit data samples at RT8, RT9, and RT10.

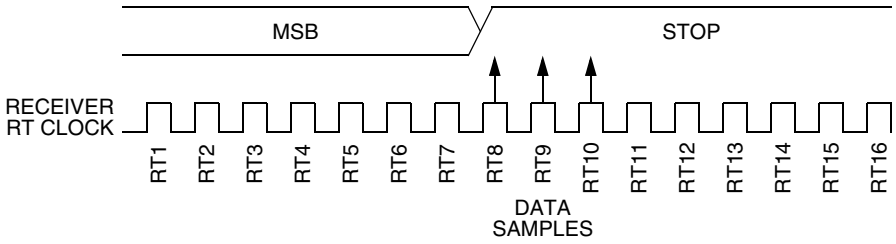


Figure 31-34. Slow Data

In the following discussion RTr represents the receiver RT clock, RTt the transmitter RT clock.

For an 8-bit data character, it takes the receiver

$$9 \text{ bit times} \times 16 \text{ RTr cycles} + 7 \text{ RTr cycles} = 151 \text{ RTr cycles} \qquad \textit{Eqn. 31-4}$$

to start data sampling of the stop bit.

With the misaligned character shown in Figure 31-34, the receiver counts 151 RTr cycles at the point when the count of the transmitting device is

$$9 \text{ bit times} \times 16 \text{ RTt cycles} = 144 \text{ RTt cycles} \qquad \textit{Eqn. 31-5}$$

The maximum percent difference between the receiver count and the transmitter count of a slow 8-bit data character with no errors is:

$$\left(\frac{151 - 144}{151}\right) \times 100 = 4.63\% \qquad \textit{Eqn. 31-6}$$

For a 9-bit data character, it takes the receiver

$$10 \text{ bit times} \times 16 \text{ RTr cycles} + 7 \text{ RTr cycles} = 167 \text{ RTr cycles} \qquad \textit{Eqn. 31-7}$$

to start data sampling of the stop bit.

With the misaligned character shown in Figure 31-34, the receiver counts 167 RTr cycles at the point when the count of the transmitting device is 10 bit times x 16 RTt cycles = 160 RTt cycles.

The maximum percent difference between the receiver count and the transmitter count of a slow 9-bit character with no errors is:

$$\left(\frac{167 - 160}{167}\right) \times 100 = 4.19\% \qquad \textit{Eqn. 31-8}$$

31.10.5.5.2 Fast Data Tolerance

Figure 31-35 shows how much a fast received frame can be misaligned. The fast stop bit ends at RT10 instead of RT16 but is still sampled at RT8, RT9, and RT10.

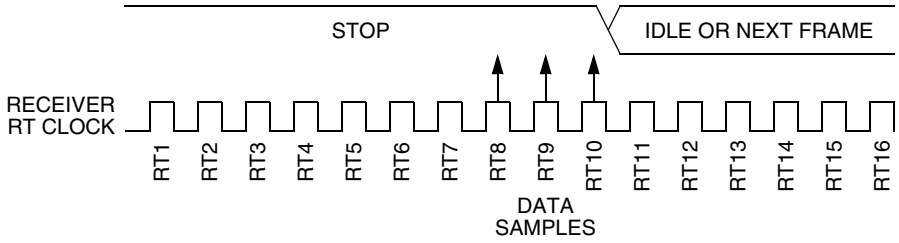


Figure 31-35. Fast Data

FoFor an 8-bit data character, it takes the receiver

$$9 \text{ bit times} \times 16 \text{ RTr cycles} + 10 \text{ RTr cycles} = 154 \text{ RTr cycles} \qquad \textit{Eqn. 31-9}$$

to finish data sampling of the stop bit.

With the misaligned character shown in Figure 31-35, the receiver counts 154 RTr cycles at the point when the count of the transmitting device is

$$10 \text{ bit times} \times 16 \text{ RTt cycles} = 160 \text{ RTt cycles} \qquad \textit{Eqn. 31-10}$$

The maximum percent difference between the receiver count and the transmitter count of a fast 8-bit character with no errors is:

$$\left(\frac{160 - 154}{160}\right) \times 100 = 3.75\% \qquad \textit{Eqn. 31-11}$$

For a 9-bit data character, it takes the receiver

$$10 \text{ bit times} \times 16 \text{ RTr cycles} + 10 \text{ RTr cycles} = 170 \text{ RTr cycles} \qquad \textit{Eqn. 31-12}$$

to finish data sampling of the stop bit.

With the misaligned character shown in Figure 31-35, the receiver counts 170 RTr cycles at the point when the count of the transmitting device is

$$11 \text{ bit times} \times 16 \text{ RTt cycles} = 176 \text{ RTt cycles} \qquad \textit{Eqn. 31-13}$$

The maximum percent difference between the receiver count and the transmitter count of a fast 9-bit character with no errors is:

$$\left(\frac{176 - 170}{176}\right) \times 100 = 3.40\% \qquad \textit{Eqn. 31-14}$$

31.10.5.6 Receiver Wakeup

To enable the eSCI to ignore transmissions intended only for other receivers in multiple-receiver systems, the receiver can be put into a standby state. Setting the receiver wakeup bit, RWU, in SCI control register 2 (SCICR2) puts the receiver into standby state during which receiver interrupts are disabled.The eSCI will still load the receive data into the SCIDRH/L registers, but it will not set the RDRF flag.

The transmitting device can address messages to selected receivers by including addressing information in the initial frame or frames of each message.

The WAKE bit in SCI control register 1 (SCICR1) determines how the eSCI is brought out of the standby state to process an incoming message. The WAKE bit enables either idle line wakeup or address mark wakeup.

31.10.5.6.1 Idle Input Line Wakeup (WAKE = 0)

In this wakeup method, an idle condition on the Rx Input signal clears the RWU bit and wakes up the eSCI. The initial frame or frames of every message contain addressing information. The CPU can read and evaluate the addressing information and decide whether it should receive the frame. If it decides not to receive, it can set the eSCI's RWU bit and return the eSCI to the standby state. The RWU bit remains set and the receiver remains on standby until another idle character appears on the Rx Input signal.

Idle line wakeup requires that messages be separated by at least one idle character and that no message contains idle characters.

The idle character that wakes a receiver does not set the receiver idle bit, IDLE, or the receive data register full flag, RDRF.

The idle line type bit, ILT, determines whether the receiver begins counting logic 1s as idle character bits after the start bit or after the stop bit. ILT is in SCI control register 1 (SCICR1).

31.10.5.6.2 Address Mark Wakeup (WAKE = 1)

In this wakeup method, a logic 1 in the most significant bit (msb) position of a frame clears the RWU bit and wakes up the eSCI. The logic 1 in the msb position marks a frame as an address frame that contains addressing information. All receivers evaluate the addressing information, and the receivers for which the message is addressed process the frames that follow.Any receiver for which a message is not addressed can set its RWU bit and return to the standby state. The RWU bit remains set and the receiver remains on standby until another address frame appears on the Rx Input signal.

The logic 1 msb of an address frame clears the receiver's RWU bit before the stop bit is received and sets the RDRF flag.

Address mark wakeup allows messages to contain idle characters but requires that the msb be reserved for use in address frames.

NOTE

With the WAKE bit clear, setting the RWU bit after the Rx Input signal has been idle can cause the receiver to wake up immediately.

31.10.6 Single-Wire Operation

Normally, the eSCI uses two pins for transmitting and receiving. In single-wire operation, the RXD pin is disconnected from the eSCI. The eSCI uses the TXD pin for both receiving and transmitting.

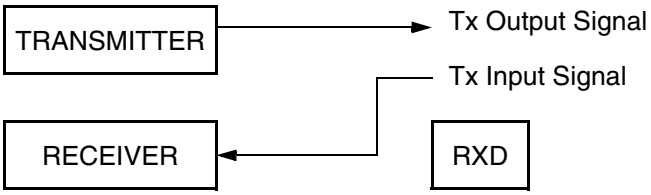


Figure 31-36. Single-Wire Operation (LOOPS = 1, RSRC = 1)

Enable single-wire operation by setting the LOOPS bit and the receiver source bit, RSRC, in SCI control register 1 (SCICR1). Setting the LOOPS bit disables the path from the Rx Input signal to the receiver. Setting the RSRC bit connects the receiver input to the output of the TXD pin driver. Both the transmitter and receiver must be enabled (TE=1 and RE=1).The TXDIR bit (SCICR3[1]) determines whether the TXD pin is going to be used as an input (TXDIR = 0) or an output (TXDIR = 1) in this mode of operation.

31.10.7 Loop Operation

In loop operation the transmitter output goes to the receiver input. The Rx Input signal is disconnected from the eSCI.

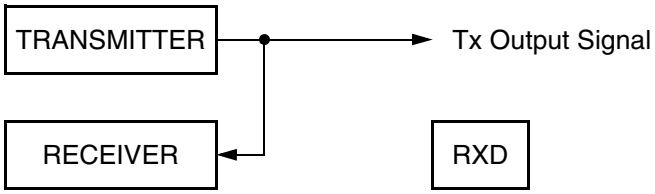


Figure 31-37. Loop Operation (LOOPS = 1, RSRC = 0)

Enable loop operation by setting the LOOPS bit and clearing the RSRC bit in SCI control register 1 (SCICR1). Setting the LOOPS bit disables the path from the Rx Input signal to the receiver. Clearing the RSRC bit connects the transmitter output to the receiver input. Both the transmitter and receiver must be enabled (TE = 1 and RE = 1).

31.10.8 Modes of Operation

31.10.8.1 Run Mode

Normal mode of operation.

31.10.8.2 Doze Mode

eSCI operation in Doze mode depends on the state of the SCISDOZ bit in the SCI control register 1 (SCICR1).

- If SCISDOZ is clear, the SCI operates normally when the system is in Doze mode.
- If SCISDOZ is set and the system is in Doze mode, the eSCI module enters a power-conservation state as soon as the current operation is completed. In normal SCI mode, this means the eSCI will

enter Doze mode, as soon as the current byte is transmitted or received, respectively. In LIN mode, it will enter Doze mode as soon as the current RX or TX frame has completed.

If the eSCI is not disabled in Doze mode, an eSCI interrupt request can be used to bring the system out of Doze mode.

31.10.8.3 Module Disable

The Module Disable Bit (MDIS) in the SCI Control Register 3 can be used to turn off the eSCI. This will prevent the eSCI core to be clocked, and thus save power. By default the eSCI is disabled, so the first step for using the eSCI is to enable it by setting the MDIS Bit to 0.

31.10.9 Interrupt Operation

31.10.9.1 Interrupt Flags and Masks

Table 31-27 lists the interrupt sources that can generate an eSCI interrupt to the CPU.

Table 31-27. eSCI Interrupt Flags and Mask Bits

Interrupt Source	Flag	Local Enable
Transmitter	TDRE	TIE
Transmitter	TC	TCIE
Receiver	RDRF	RIE
Receiver	IDLE	ILIE
Receiver	PF	PFIE
Receiver	FE	FEIE
Receiver	NF	NFIE
Receiver	OR	ORIE
LIN	BERR	IEBERR
LIN	RXRDY	RXIE
LIN	TXRDY	TXIE
LIN	LWAKE	WUIE
LIN	STO	STIE
LIN	PBERR	PBIE
LIN	CERR	CIE
LIN	CKERR	CKIE
LIN	FRC	FCIE
LIN	OVFL	OFIE

The following describes how the eSCI generates a request and how the MCU should acknowledge that request. The interrupt vector offset and interrupt number are chip dependent. The eSCI only has a single interrupt line (eSCI Interrupt Signal, active high operation) and all the following interrupts, when generated, are ORed together and issued through that port.

31.10.9.2 Interrupt Description

Table 31-28 describes the interrupt sources of the eSCI. The enables for these sources, are listed in Table 4-7.

Table 31-28. eSCI Interrupt Sources

Interrupt	Source	Description
TDRE	SCISR1[7]	Active high level detect. Indicates that a byte was transferred from SCIDRH/L to the transmit shift register.
TC	SCISR1[6]	Active high level detect. Indicates that a transmit is complete.
RDRF	SCISR1[5]	Active high level detects. The RDRF interrupt indicates that received data is available in the SCI data register.
IDLE	SCISR1[4]	Active high level detect. Indicates that receiver input has become idle.
OR	SCISR1[3]	Active high level detects. This interrupt indicates that an overrun condition has occurred.
NF	SCISR1[2]	Active high level detects. This interrupt indicates that noise has been detected.
FE	SCISR1[1]	Active high level detects. This interrupt indicates that a frame error has occurred.
PF	SCISR1[0]	Active high level detects. This interrupt indicates that a parity error has occurred.
BERR	SCISR2[4]	Detected a Bit Error, only valid in LIN mode
RXRDY	LINSTAT1[7]	Indicates LIN hardware has received a data byte
TXRDY	LINSTAT1[6]	Indicates LIN hardware can accept a control or data byte
LWAKE	LINSTAT1[5]	A Wakeup Character has been received from a LIN frame
STO	LINSTAT1[4]	The response of the slave has been too slow (Slave TimeOut)
PBERR	LINSTAT1[3]	Physical Bus Error detected
CERR	LINSTAT1[2]	CRC Error detected
CKERR	LINSTAT1[1]	Checksum Error detected
FRC	LINSTAT1[0]	LIN Frame completed
OVFL	LINSTAT2[0]	LINRX Register Overflow

31.10.9.3 TDRE Description

The TDRE interrupt is set high by the eSCI when the transmit shift register receives a byte from the SCI data register. A TDRE interrupt indicates that the transmit data register (SCIDRH/L) is empty and that a new byte can be written to the SCIDRH/L for transmission. Clear TDRE by writing it with 1.

31.10.9.4 TC Description

The TC interrupt is set by the eSCI when a transmission has been completed.A TC interrupt indicates that there is no transmission in progress. TC is set high when the TDRE flag is set and no data, preamble, or break character is being transmitted. When TC is set, the TXD pin becomes idle (logic 1). Clear TC by writing it with 1.

31.10.9.5 RDRF Description

The RDRF interrupt is set when the data in the receive shift register transfers to the SCI data register. A RDRF interrupt indicates that the received data has been transferred to the SCI data register and that the byte can now be read by the MCU. The RDRF interrupt is cleared by writing it with 1.

31.10.9.6 PF Description

The interrupt is set when the parity of the received data is not correct. PF is cleared by writing it with 1.

31.10.9.7 FE Description

The interrupt is set when the stop bit is read as a 0 - which violates the SCI protocol. FE is cleared by writing it with 1.

31.10.9.8 NF Description

The NF interrupt is set when the eSCI detects noise on the receiver input. NF is cleared by writing it with 1.

31.10.9.9 OR Description

The OR interrupt is set when software fails to read the SCI data register before the receive shift register receives the next frame. The newly acquired data in the shift register will be lost in this case, but the data already in the SCI data registers is not affected. The OR interrupt is cleared by writing it with 1.

31.10.9.10 IDLE Description

The IDLE interrupt is set when 10 consecutive logic 1s (if M=0) or 11 consecutive logic 1s (if M=1) appear on the receiver input. Once the IDLE is cleared, a valid frame must again set the RDRF flag before an idle condition can set the IDLE flag. Clear IDLE by reading by writing it with 1.

31.10.9.11 BERR Description

The BERR flag is set when the eSCI is in LIN mode, and at least one bit in the last byte that which was transmitted, is not read back with the same value. The flag is cleared by writing it with 1.

31.10.9.12 RXRDY Description

The RXRDY flag is set, when the eSCI is in LIN mode, and has received a valid data byte in an RX frame. This will not be set for bytes which the receiver obtains by reading back the data which the LIN FSM has sent out. The flag is cleared by writing it with 1.

31.10.9.13 TXRDY Description

The TXRDY flag is set, when the eSCI is in LIN mode, and can accept a control or data byte. The flag is cleared by writing it with 1.

31.10.9.14 LWAKE Description

The LWAKE flag is set when the LIN hardware has detected a wakeup character, sent out by one of the LIN slaves. This should only occur when the LIN bus is in sleep mode. The flag can be cleared by writing it with 1.

31.10.9.15 STO Description

The STO flag is set during an RX frame, when the LIN slave has not transmitted all requested data bytes, before the specified timeout period. The flag can be cleared by writing it with 1.

31.10.9.16 PBERR Description

If the RX input remains stuck at a fixed value for 15 cycles after a transmission has started, the LIN hardware will set the PBERR (Physical Bus Error) flag. The flag can be cleared by writing it with 1.

31.10.9.17 CERR Description

If an RX frame had the CRC checking flag set and the two CRC bytes did not match the calculated CRC pattern, the CERR (CRC Error) flag will be set. The flag can be cleared by writing it with 1.

31.10.9.18 CKERR Description

If an RX frame had the Checksum checking flag set (that should normally always be the case) and the last byte did not match the calculated checksum, the CKERR (Checksum Error) flag will be set. The flag can be cleared by writing it with 1.

31.10.9.19 FRC Description

The FRC (Frame Complete) Flag will be set after the last byte of a TX frame has been sent out, or after the last byte of an RX frame has been received (normally that means that the checksum has been compared to the expected value, too). This can be used to indicate to the CPU that the next frame can be set up, however it should be noted that this might be raised before the DMA controller has transferred the last byte from the eSCI to the memory. If the intent is to process the data, the appropriate interrupt of the DMA controller should be used instead. The flag can be cleared by writing it with 1.

31.10.9.20 OVFL Description

The OVFL (LINRX Overflow) Flag will be set when a byte is received in the LINRX register before the previous byte has been read. As the system is responsible for reading the register before a new byte arrives this indicates a problem with CPU load. The flag can be cleared by writing it with 1.

31.10.10 Using the eSCI in 9-bit data mode

Sometimes it is desirable to set the 9th data bit for each write and to retrieve the 9th data bit for each read. This can be accomplished by defining the data structure for the SCI Data Registers (see Figure 31-8 and Figure 31-9) appropriately.

Normally the header files for the eSCI are defined as in the example below:

```
...
volatile reg8 sci_SCIDRH;                /* SCIx_REGISTER_MAP_OFFSET + 0x06 */
volatile reg8 sci_SCIDRL;                /* SCIx_REGISTER_MAP_OFFSET + 0x07 */
...
```

This format requires to access the data register with two writes or two reads, respectively - e.g.:

```
SCI_A->sci_SCIDRH = tx_dat8;
SCI_A->sci_SCIDRL = tx_dat7_0;

rx_dat8_0 = SCI_A->sci_SCIDRH <<1 | SCI_A->sci_SCIDRL;
```

For more efficient access the header files can be written like this:

```
...
volatile reg16 sci_SCIDR;                /* SCIx_REGISTER_MAP_OFFSET + 0x06 */
...
```

Now a single write or read can be used to access all 9 bits:

```
SCI_A->sci_SCIDR = tx_dat8_0;

rx_dat8_0 = SCI_A->SCIDR;
```

(Note that the 9th data bit is in position 6 for TX and position 7 for RX, so some re-formatting may be necessary.)

If the 9th bit is unused or only written and read occassionally, the declaration with two 8 bit registers is preferrable. However if the 9th bit needs to be written and read on every access to the SCI data register the declaration with a single 16 bit register is more effecient.

31.10.11 Using the LIN hardware

The eSCI provides special support for the LIN protocol. It can be used to automate most tasks of a LIN master. In conjunction with the DMA interface it is possible to transmit entire frames (or sequences of frames) and receive data from LIN slaves without any CPU intervention. There is no special support for LIN slave mode, if required it could be implemented in software.

A LIN frame consists of a break character (10/13 bits, configurably), a sync field, an ID field, n data fields (n could be 0) and a checksum field. The data and checksum bytes are either provided by the LIN master (TX frame) or by the LIN slave (RX frame). The header fields will always be generated by the LIN master.

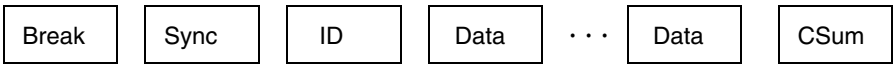


Figure 31-38. Typical LIN frame

The LIN hardware is highly configurable, and allows to generate frames for LIN slaves from all revisions of the LIN standard. The settings need to be adjusted for the capabilities of the slave device.

In order to activate the LIN hardware, the LIN mode bit in the LINCTRL register needs to be set, other settings like double stop flags after bit errors and automatic parity bit generation, are also available.

The eSCI settings need to be made according to the LIN specification, it needs to be configured for 2-wire operation (2 wires connected to the LIN transceiver) with 8 data bytes and no parity. Normally a 13-bit break is used, but it can also be configured for 10-bit breaks as required by the application.

31.10.11.1 Generating a TX Frame

The following procedure illustrates how to generate a basic TX frame.

The frame is controlled via the TX register. Initially the application software will need to check the TXRDY bit (either using an interrupt, the TX DMA interface, or by polling the LIN Status Register). If the bit is set to 1 the register is writable. Before each write the bit needs to be checked (automatic in DMA mode). The first byte written to the TX register has to contain the LIN ID field, then the length of the frame is specified (0 to 255 Bytes), and a control byte (frame direction, checksum/crc settings) are written. (The timeout bits are skipped for TX frames, since they only refer to LIN slaves). After this frame data is known the LIN hardware will start to generate the LIN frame.

First it will transmit a break field, then the sync field and the ID field. Afterwards the TX register will accept data bytes, and the LIN hardware will transmit these bytes as soon as they are available and can be sent out. After the last step it will automatically append the checksum field.

It is possible to setup a DMA channel to handle all the tasks required to send a TX frame (see Figure 31-39). For this, the TX DMA channel has to be activated by setting the TXDMA bit. The control information for the LIN frame (ID, message length, TX/RX type etc.) and the data bytes can be stored at an appropriate memory location. The DMA controller will then be set up to transfer this block of memory to a location (the TX Register). After transmission is complete either the DMA controller or the LIN hardware can generate an interrupt to the CPU, everything else can be handled by the eSCI and the DMA controller.

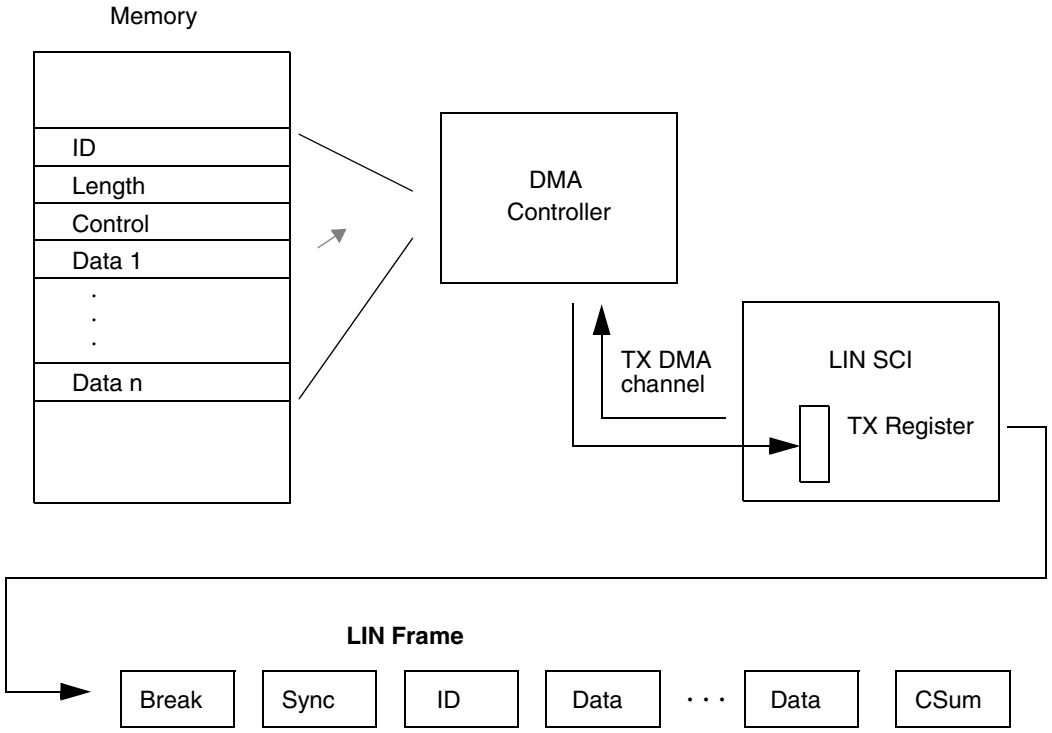


Figure 31-39. DMA transfer of a TX frame

31.10.11.2 Generating an RX frame

For RX frames the header information will still be provided by the LIN master - the data, CRC and checksum bytes (as enabled) will be provided by the LIN slave. The LIN master will verify CRC and checksum bytes transmitted by the slave.

For an RX frame the control information needs to be written to the TX register in the same manner as for the TX frames. Additionally the timeout bits need to be written, to define the time to complete the entire frame. Afterwards the RXRDY bit needs to be checked (either with an interrupt, RX DMA interface, or by polling) to detect the incoming data bytes. The checksum byte will normally not appear in the RX register, instead the LIN hardware will verify the checksum and raise an interrupt, if it is not correct.

Two DMA channels can be used when executing an RX frame - one to transfer the header/control information from a memory location to the TX register, and one to transfer the incoming data bytes from the RX register to a table in memory (see Figure 31-40). After the last byte has been stored, the DMA controller can indicate completion to the CPU.

It is also possible to setup a whole sequence of RX and TX frames, and generate a single event at the end of that sequence.

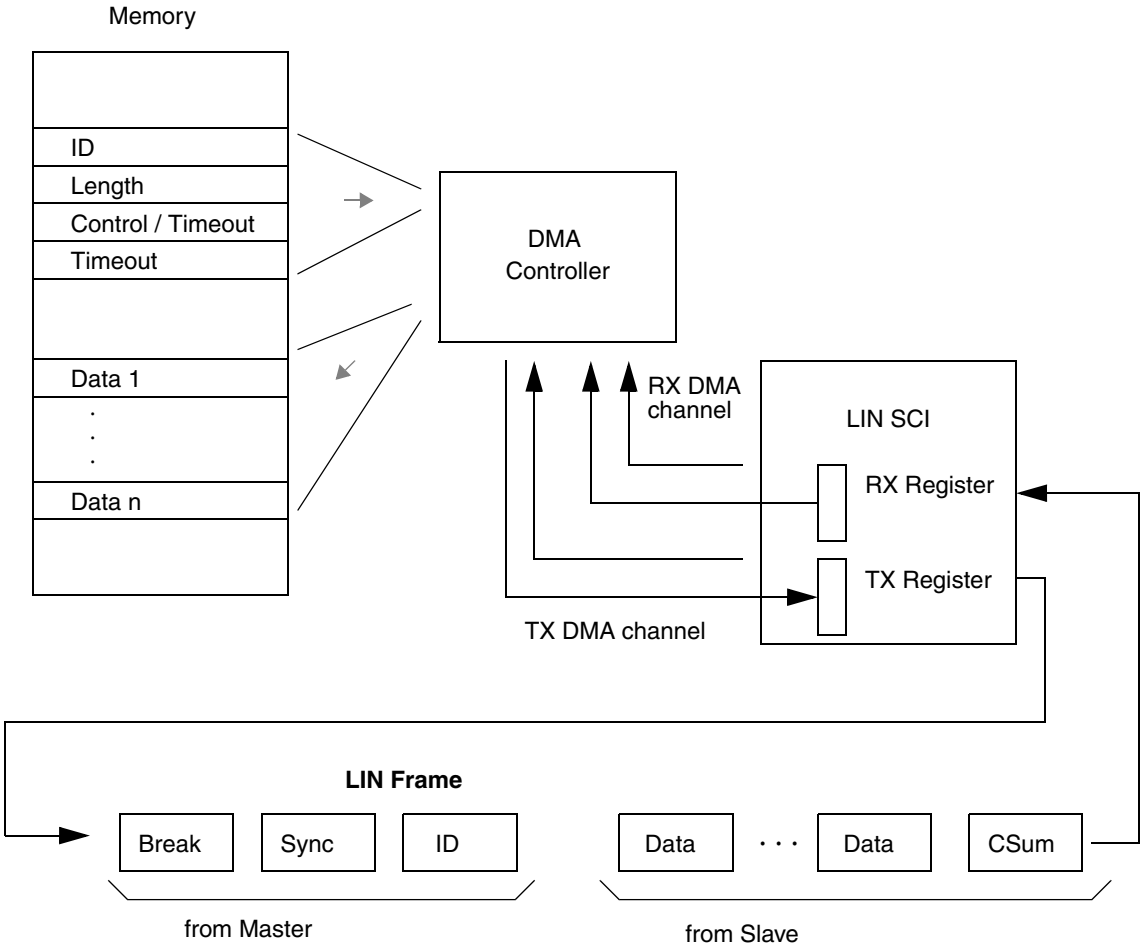


Figure 31-40. DMA transfer of a RX frame

31.10.11.3 Features of the LIN Hardware

The LIN hardware has several features to support different revisions of the LIN slaves. In the TX register it can be configured whether header bits should be included in the checksum (on a frame by frame basis, to support LIN slaves with different LIN revisions). The LIN control register allows to decide whether the parity bits in the ID field should be calculated automatically, and whether double stop flags should be inserted after a Bit Error. The BRK13 bit in the SCI control register 3 decides whether to generate 10 or 13 bit break characters.

The application software can decide to turn off the checksum generation/verification on a per frame basis and handle that function on its own. Also it can decide to let the LIN hardware append two CRC Bytes (Figure 31-41). These are not part of the LIN standard, but could be part of the application layer i.e. the would be treated as data bytes by the LIN protocol. This can be useful when very long frames are transmitted. By default the CRC polynom used, is the same polynom as for the CAN protocol.

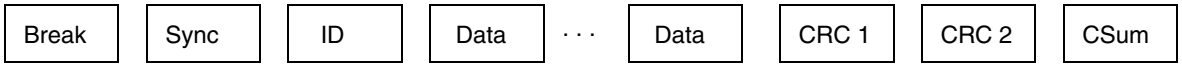


Figure 31-41. LIN frame with CRC bytes

It is possible to force a resync of the LIN FSM, with the LRES bit in the LIN control register, however normally the LIN hardware will automatically abort a frame after detecting a bit error.

31.10.11.4 LIN Error Handling

The LIN hardware can detect several error conditions of the LIN protocol. It will receive every byte that was transmitted, and compare it with the intended values. If there is a mismatch, a bit error will be raised, and the LIN FSM will return to its start state.

For a RX frame the LIN hardware can detect a slave timeout error - the exact value can be set via the timeout bits in the TX register. If the frame is not complete within the number of clock cycles specified there, the LIN FSM will return to its start state, and the STO (Slave Time-Out) interrupt will be raised.

The LIN protocol supports a sleep mode - after 25000 bus cycles of inactivity the bus is assumed to be in sleep mode. Normally entering sleep mode can be avoided, if the LIN master is regularly creating some bus activity. Otherwise the timeout state needs to be detected by the application software - e.g. by setting a timer.

Both LIN masters and LIN slaves can cause the bus to exit sleep mode by sending a break signal. The LIN hardware will generate such a break, when the WU bit in the LIN control register is written. After transmitting this break the LIN hardware will not send out data (i.e. not raise the TXRDY flag) before the Wakeup Delimiter period has expired. This period can be selected by setting the WUD bits in the LIN control register.

Break signals sent by a LIN slave are detected by the LIN hardware, and indicated by setting the WAKE flag in the LIN status register.

A Physical Bus Error (LIN bus is permanently stuck at a fixed value) will set several error flags. If the input is permanently low, the eSCI will set the Framing Error Flag (FE) in the SCI status register. If the RX input remains stuck at a fixed value for 15 cycles, after a transmission has started, the LIN hardware will set the PBERR flag in the LIN status register. In addition a bit error may be generated.

31.10.11.5 LIN Wakeup

The LIN hardware automatically detects LIN 1.x wakeup characters and can generate them with the WU bit in LINCTRL1. For LIN 2.0 wakeup characters the requirements are more flexible - instead of a 80h character at the currently selected baudrate, the spec requires a low pulse of 250 µs to 5 ms. For this the baud rate needs to be set to 32k down to 1.6k baud.

So in order to generate a valid wakeup character according to LIN 2.0, the eSCI first needs to be programmed to a baud rate lower than 32 kBaud, then WU can be set. Should the application require a higher baud rate, then this rate can be set once the wakeup character has been transmitted.

For wakeup detection the length of the wakeup pulse depends on the LIN slave node. A wakeup which does not conform to LIN 1.3 will show up as a frame error FE. Provided that the low pulse is longer than 8 bit times (this can be controlled by determining the baud rate), the LWAKE flag will be also be set. The application then needs to wait until the wakeup pulse has completed (e.g. wait until 5 ms have expired) and clear the FE and LWAKE bits. A further 100 ms wait may be required to ensure that all connected

slaves are ready to receive frames (for the exact definitions please refer to section 5 - *Network Management* in the LIN 2.0 specification).

31.10.11.6 System Wakeup on LIN Bus Activity

It may be desirable to generate a wakeup interrupt to the system, when a LIN wakeup character is received. This can be implemented by switching the RXD-SCI receive pin into GPIO mode and setting up the GPIO interrupt appropriately so that it can create wakeup interrupts, before entering STOP mode. The length of the wakeup pulse needs to conform to the requirements specified for the GPIO interrupt. LIN 2.0 compliant wakeups should fulfill these requirements.

Alternatively an additional pin can be dedicated just for the wakeup interrupt. This pin would be brought into GPIO mode, and externally connected to the RXD-SCI receive pin.

31.10.11.7 LIN Setup

Since the eSCI is for general purpose, some of the settings are not applicable for LIN operation. The following setup should apply for all applications, regardless which kind of LIN slave is addressed:

a) The module needs to be enabled by writing the MDIS bit to 0
b) Both Transmitter and Receiver need to be enabled (TE=1, RE=1)
c) The data format bit M, needs to be set to 0 (8 data bits), the parity needs to be disabled (PE=0)
d) TIE, TCIE, RIE interrupt enable bits should be inactive, instead the LIN interrupts should be used
e) The eSCI needs to be switched into LIN mode (LIN=1)
f) The LIN standard requires that the break character always be 13 bits long (BRK13=1)
g) Normally Bit Errors should cause the LIN FSM to reset, stop driving the bus immediately and stop further DMA requests until the BERR flag has been cleared, so LDBG=0, SBSTP=1 and BSTP=1 should be set
h) Fast Bit Error Detection provides superior Error Checking, so FBR should be set, normally it will be used with BESM13=1
i) If available a pulldown should be enabled on the RX input
j) The error indicators NF, FE, BERR, STO, PBERR, CERR, CKERR and OVFL should be enabled
k) Initially a wakeup character may need to be transmitted on the LIN bus, so that the LIN slaves activate

Other settings like baud rate, DMA interface etc will depend on the LIN slaves to which the eSCI is connected and on the desired operation of the eSCI.

Chapter 32
Modular I/O Subsystem (eMIOS)

32.1 Introduction

The Enhanced Modular Input/Output Subsystem has a similar architecture to that of the Modular Input/Output Subsystem (MIOS) previously implemented on other Freescale devices. This module has an interface to the peripheral bus that communicates to timer channel sub-modules on a local inter-module bus with the sub-modules providing the timer and counter functions needed by the applications. Common time bases can be shared between sub-modules using counter buses in order to offer synchronous operation. The most notable difference of the eMIOS implementation is the introduction of the Unified Channel sub-module. The eMIOS is constructed with only Unified Channels, which provide all of the required timer functions. This provides a more flexible implementation to that of its predecessor, the MIOS, where different sub-modules would provide a much narrower range of functions. Each Unified Channel is identical and can be configured to provide a wide range of timer functions.

On MAC72x2, the eMIOS has 8 Unified timer Channels with all channel counters being 16-bits wide. The module implements 2 counter buses: Counter bus A and Counter bus B. Counter bus A is driven by Unified channel 7 and can be shared across all Unified Channels. Counter bus B is driven by channel 0 and can be shared across all Unified Channels.

On MAC72x1, the eMIOS has 16 Unified timer Channels with all channel counters being 16-bits wide. The module implements 3 counter buses: Counter bus A, Counter bus B and Counter bus C. Counter bus A is driven by Unified channel 15 and can be shared across all Unified Channels. Counter bus B is driven by channel 0 and can be shared across Channels 0-7. Counter bus C is driven by channel 8 and can be shared across Channels 8-15.

Counter bus A cannot be driven from a Shared Timer And Counter (STAC), as no STAC is implemented on the MAC7200 family of devices.

A single timer pin is shared for each of the Unified Channels input and output signals. This results in a timer with 8 external pins available for the user. The eMIOS output disable channels are assigned to Unified channel 0, 1, 2 and 3, and are used to drive the MTS_ODIS_0, MTS_ODIS_1, MTS_ODIS_2 and MTS_ODIS_3 respectively in order to disable the channels output based on the ODIS and ODISSL[0:1] bits in the channel control register MTSCn.

The eMIOS module can be independently disabled by writing to the MDIS bit in the module's MCR register. Disabling the module will turn off the clock to the module, although some of the module registers (MTSMCT, MTSOUDIS & MTSUCDIS) remain available to be accessed by the core across the peripheral bus. The MDIS bit is intended to be used when the module is not required in the application. By default, the eMIOS is disabled after reset (MDIS=1), so the MDIS bit must be cleared, prior to use.

NOTE

The terms “MTS” and “eMIOS” are used interchangeably in this document.

32.2 eMIOS Features on MAC72xx

- 8 or 16 unified channels, with every channel able to provide all timer functions and modes.
- All channels can be enabled for eDMA service.
- Channels can be individually disabled to assist with power saving.
- Two or three 16-bit counter buses (A and B, or A, B and C) for sharing time base around the module.
- One global prescaler and an individual prescaler available for each channel.
- Modulus counter mode on all channels.
- Single action input capture or output compare modes on all channels.
- Input pulse width and period measurement modes on all channels.
- Double action output compare mode on all channels.
- Output pulse width and frequency modulation modes on all channels.
- Output pulse width modulation modes on all channels.
- Center aligned output pulse width modulation with dead time insertion modes on all channels.
- Pulse or edge accumulation and counting modes on all channels.
- Windowed programmable time accumulation mode on all channels.
- Quadrature decode modes on all channels.
- General purpose I/O available on unused eMIOS pins.
- Center-aligned PWM with dead-time insertion

32.3 eMIOS Protocol

As the eMIOS supports several different protocols, please refer to the eMIOS Block Guide for a complete description of eMIOS modes.

32.4 eMIOS Implementation

The eMIOS module has several different configuration options. On the MAC72xx devices, the eMIOS module has the following configuration:

- MDIS Reset Value: The reset value of the **MDIS** bit is 1, meaning that the eMIOS is in a disabled state after reset, and must be explicitly enabled before it can be used.
- Number of channels: The number of unified eMIOS channels on the MAC72x2 is 8. In this configuration, the Counter Bus A is driven by Channel 7. The number of unified eMIOS channels on the MAC72x1 is 16. In this configuration, the Counter Bus A is driven by Channel 15.
- Timer Width: The width of all eMIOS timer channels on the MAC72xx is 16 bits.

32.5 eMIOS External Pins

Table 32-1. eMIOS External Pins

Signal	Description
EMIOS0	Unified Channel 0
EMIOS1	Unified Channel 1
EMIOS2	Unified Channel 2
EMIOS3	Unified Channel 3
EMIOS4	Unified Channel 4
EMIOS5	Unified Channel 5
EMIOS6	Unified Channel 6
EMIOS7	Unified Channel 7

32.6 eMIOS Bus Aborts

The eMIOS module supports Peripheral Bus bus aborts, and enforces the following memory map:

Table 32-2. eMIOS Bus Aborts

Abort	Allowed
	0x0000–0x000f
0x0010–0x001f	
	0x0020–0x011f
0x0120–0x3fff	

Supervisor Access: Unused.

NOTE

The above memory map is for an 8 channel eMIOS only.The address decoding is based on the number of channels implemented.

32.7 eMIOS Differences betwee MAC71xx and MAC72x2

- Switched from 16 channels (Channels 0-15) to 8 channels (Channels 0-7)
  - — Use channels 0-7
  - — Channel 7 can now drive Counter Bus A
  - — Flag Out feature uses 3:0 instead of 15:12
- Fixed MUCts01651: Write on bus abort still writes the register
- Fixed MUCts01526: New eMIOS mode requested by customer (Block Guide not yet updated, See eMIOS White Paper v0.7)
  - a) Latch the A2 and B2 registers into A1 and B1, respectively, on a match on B.
  - b) Always match on B, regardless of whether a match on A has occurred.

c) If A == 0, provide a 0% duty cycle without having to manipulate the EDPOL bit.
d) If A >= B, provide a 100% duty cycle without having to manipulate the EDPOL bit.

- Fixed MUCts01793: Corner case violates coherent access in IPWM/IPM modes

32.8 eMIOS Differences between MAC72x2 and MAC72x1

- Switched from 8 channels (Channels 0-7) to 16 channels (Channels 0-15)
  - Use channels 0-15
  - Channel 15 can now drive Counter Bus A

32.9 Enabling the eMIOS

Before the eMIOS can be used, it must be explicitly enabled by clearing the **MDIS** bit.

32.10 The eMIOS Module

32.10.1 Overview

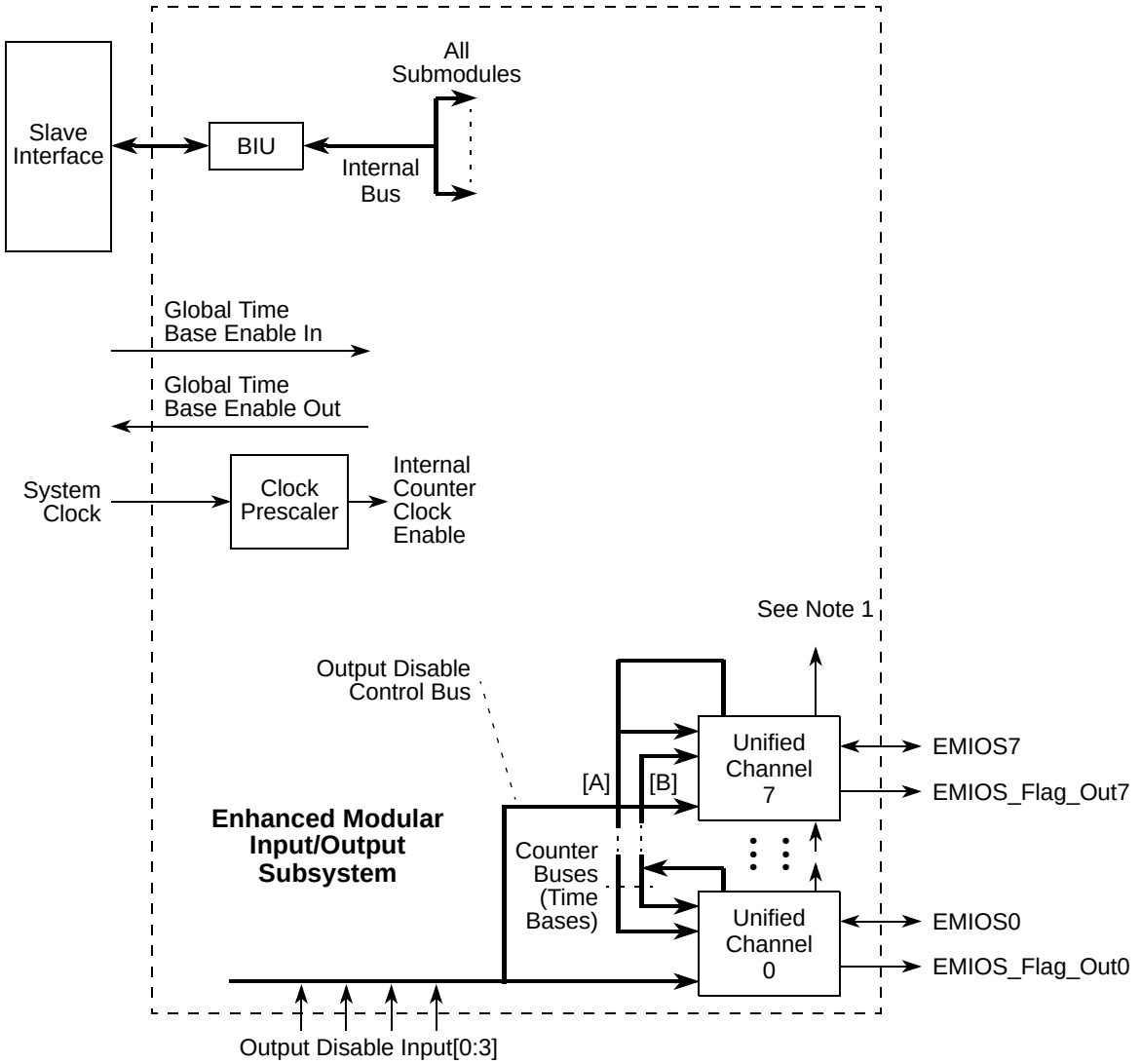
The Enhanced Modular I/O Subsystem (eMIOS) provides functionality to generate or measure timed events. Its overall architecture resembles that of the MIOS .

The MIOS provides a framework where a set of sublocks with different timer functions are assembled to attend the specific needs of an MCU.

The eMIOS builds on this concept by using a Unified Channel module that provides a superset of the functionality of all the individual MIOS channels, while providing a consistent user interface. This allows more flexibility as each Unified Channel can be programmed for different functions in different applications of the MCU.

32.10.2 Block Diagram

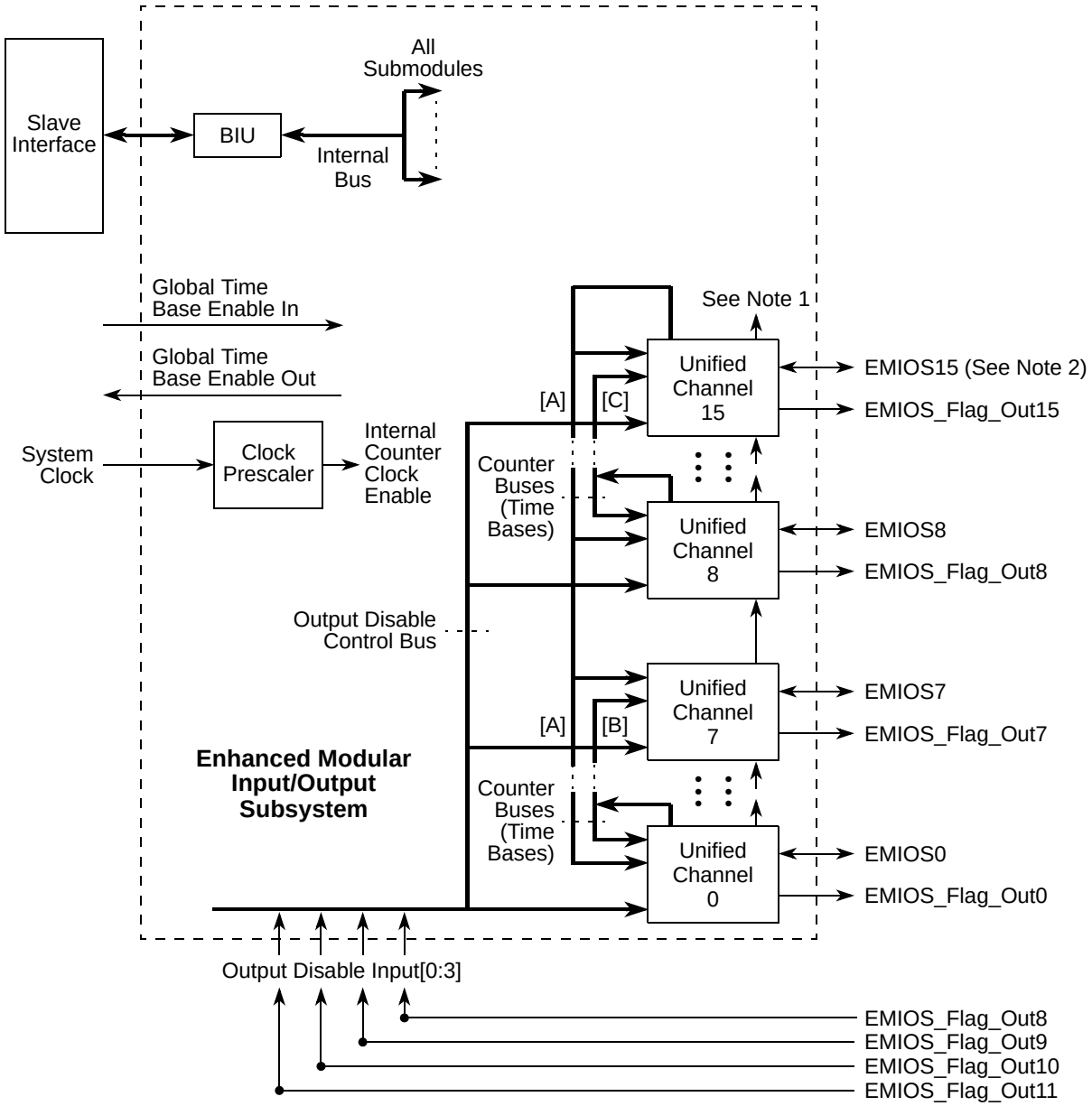
Figure 32-2 shows the block diagram of the eMIOS as implemented on the MAC72x2.



Note 1: Connection between UC[n-1] and UC*n* necessary to implement QDEC mode.

Figure 32-1. eMIOS Block Diagram for MAC72x2

Figure 32-2 shows the block diagram of the eMIOS as implemented on the MAC72x1.



Note 1: Connection between UC[n-1] and UC*n* necessary to implement QDEC mode.
Note 2: On channels 12–15, there is no input from EMIOS[12:15], but only from the DSPI module.

Figure 32-2. eMIOS Block Diagram for MAC72x1

32.10.3 Features

The basic features of the eMIOS on MAC72xx are as follows:

- 8/16 Unified Channels
- Data registers 16 bits wide

- Counter bus A can be driven by Unified Channel 7 (MAC72x2) or Unified Channel 15 (MAC72x1)
- Counter bus B can be driven by Unified Channel 0
- Counter bus C can be driven by Unified Channel 8
- Each channel has its own time base, alternative to the counter buses
- One Global prescaler
- One Prescaler per channel (CP)
- Shared timebases through the counter buses
- Control and Status bits grouped in a single register
- Synchronization among timebases
- Shadow FLAG register
- State of the UC can be frozen for debug purposes
- Motor control capability

32.10.4 Modes of Operation

The channels can be configured to operate in the following modes:

- General purpose input/output
- Single Action Input Capture
- Single Action Output Compare
- Input Pulse Width Measurement
- Input Period Measurement
- Double Action Output compare
- Pulse/Edge Accumulation
- Pulse/Edge Counting
- Quadrature Decode
- Windowed Programmable Time Accumulation
- Modulus Counter
- Modulus Counter Buffered
- Output Pulse Width and Frequency Modulation
- Output Pulse Width and Frequency Modulation Buffered
- Center Aligned Output Pulse Width Modulation with dead time insertion
- Center Aligned Output Pulse Width Modulation with dead time insertion Buffered
- Output Pulse Width Modulation
- Output Pulse Width Modulation Buffered

These modes are described in Section 32.13, “Functional Description”.

32.11 External Signal Description

32.11.1 Overview

While each Unified Channel has one external input and one external output signal, as described in Table 32-3, on the MAC72xx, the input and output signals for each channel are connected to a single bidirectional pin EMIOSn (see Chapter 34, “Port Integration Module (PIM_MAC7202)”).

32.11.2 Detailed Signal Descriptions

Table 32-3. External signals

Signal	Direction	Function	Reset State	Pull up
EMIOSIn	input	eMIOS Unified Channel n input	-	chip dependent
EMIOSOn	output	eMIOS Unified Channel n output	0/ Hi-Z[1]	chip dependent

1. Value “0” refers to the reset value of the signal. Hi-Z refers to the state of the external pin if a tristate output buffer is controlled by the corresponding ipp_obe_mts_ucn signal.

32.11.2.1 EMIOSIn - eMIOS Unified Channel Input Signal

EMIOSIn is synchronized and filtered by the input programmable filter (IPF). The output of the IPF is then used by the channel logic and is available to be read by the MCU through the UCIN bit of the UCSRn register.

32.11.2.2 EMIOSOn - eMIOS Unified Channel Output Signal

EMIOSOn is a registered output and is available for reading by the MCU through the UCOUT bit of the UCSRn register.

32.12 Memory Map/Register Definition

32.12.1 Memory Map

Addresses of Unified Channel registers are specified as offsets from the channel’s base address, otherwise the eMIOS base address is used as reference.

The overall address map organization is shown in Table 32-4. Table 32-5 describes the Unified Channel registers.

Attempts to access reserved addresses will result in a bus abort exception.

Table 32-4. eMIOS Memory Map

eMIOS Addresses	Register Description
0x0000–0x0003	Module Configuration register (MCR)
0x0004–0x0007	Global FLAG register (GFLAG)
0x0008–0x000B	Output Update Disable (OUDIS)

Table 32-4. eMIOS Memory Map

0x000C–0x000F	Disable Channel (UCDIS)
0x0010–0x001F	reserved
0x0020–0x003F	Unified Channel 0 (UC0)
0x0040–0x005F	Unified Channel 1(UC1)
0x0060–0x007F	Unified Channel 2 (UC2)
0x0080–0x009F	Unified Channel 3 (UC3)
0x00A0–0x00BF	Unified Channel 4 (UC4)
0x00C0–0x00DF	Unified Channel 5 (UC5)
0x00E0–0x00FF	Unified Channel 6 (UC6)
0x0100–0x011F	Unified Channel 7 (UC7)
0x0120–0x013F	Unified Channel 8 (UC8)
0x0140–0x015F	Unified Channel 9 (UC9)
0x0160–0x017F	Unified Channel 10 (UC10)
0x0180–0x019F	Unified Channel 11 (UC11)
0x01A0–0x01BF	Unified Channel 12 (UC12)
0x01C0–0x01DF	Unified Channel 13 (UC13)
0x01E0–0x01FF	Unified Channel 14 (UC14)
0x0200–0x021F	Unified Channel 15 (UC15)
0x0220–0x0FFF	reserved

Table 32-5. UC Memory Map

UCn Base Address	Description
0x000	Channel A Data Register (UCAn)
0x0004	Channel B Data Register (UCBn)
0x0008	UC Counter Register (UCCNTn)
0x000C	Channel Control Register (UCCRN)
0x0010	UC Status Register (UCSRn)
0x0014	Alternate A Register (ALTAn)
0x0018 - 0x001F	reserved

32.12.2 Register Descriptions

All eMIOS registers on MAC72xx are 32 bits wide, even though the eMIOS on MAC72xx has 8/16 Unified Channels and 16-bit wide data registers.

32.12.2.1 eMIOS Module Configuration Register (MCR)

The MCR register contains Global Control bits for the eMIOS block.

Address eMIOS base address + 0x00

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	DOZEEN	MDIS	FRZ	GTBE	0	GPREN	0	0	0	0	0	0	0			
W																
Reset:	0	1[1]	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	GPRE[7:0]								0	0	0	0	0	0	0	0
W																
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 32-3. eMIOS Module Configuration Register (MCR)

1. In MAC7200, the eMIOS is disabled after reset (MDIS = 1). This bit must be cleared after reset to enable the eMIOS.

Table 32-6. MCR Field Descriptions

Field	Descriptions
31 DOZEEN	Doze Enable bit. Enable the eMIOS to enter low power mode when Doze Mode is requested at MCU level. The doze mode is used to stop the clock of the block, except the access to registers MCR, OUDIS and UCDIS. 0 Not enable to enter low power mode when Doze Mode is requested. 1 Enable to enter low power mode when Doze Mode is requested
30 MDIS	Module Disable bit. Puts the eMIOS in low power mode. The MDIS bit is used to stop the clock of the block, except the access to registers MCR, OUDIS and UCDIS. 0 Clock is running. 1 Enter low power mode.
29 FRZ	Freeze bit. Enable the eMIOS to freeze the registers of the Unified Channels when Debug Mode is requested at MCU level. Each Unified Channel should have FREN bit set in order to enter freeze mode. While in Freeze mode, the eMIOS continues to operate to allow the MCU access to the Unified Channels registers. The Unified Channel will remain frozen until the FRZ bit is written to zero or the MCU exits Debug mode or the Unified Channel FREN bit is cleared. 0 Exit freeze mode. 1 Stops Unified Channels operation when in Debug mode and the FREN bit is set in the UCCRn register
28 GTBE	Global Time Base Enable bit. The GTBE bit is used to export a Global Time Base Enable from the module and provide a method to start time bases of several blocks simultaneously. 1 Global Time Base Enable Out signal asserted 0 Global Time Base Enable Out signal negated **Note:** The Global Time Base Enable input pin controls the internal counters. When asserted, Internal counters are enabled. When negated, Internal counters disabled.
27	Reserved, should be cleared.

Table 32-6. MCR Field Descriptions (Continued)

Field	Descriptions
26 GPREN	Global Prescaler Enable bit. The GPREN bit enables the prescaler counter. 0 Prescaler disabled (no clock) and prescaler counter is cleared 1 Prescaler enabled
25–16	Reserved, should be cleared.
15–8 GPRE[7:0	Global Prescaler bits. The GPRE[7:0] bits select the clock divider value for the global prescaler, as shown in Table 32-7.

Table 32-7. Global Prescaler Clock Divider

GPRE[7:0]	Divide ratio
00000000	1
00000001	2
00000010	3
00000011	4
. . . .	. . . .
11111110	255
11111111	256

7–0	Reserved, should be cleared.

32.12.2.2 eMIOS Global FLAG Register (GFLAG)

The GFLAG is a read-only register that groups the FLAG bits from all channels. This organization improves interrupt handling on simpler devices. These bits are mirrors of the FLAG bits of each channel register (UCSR).

Address eMIOS base address + 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																
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	F7	F6	F5	F4	F3	F2	F1	F0
W																
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 32-4. eMIOS Global FLAG Register (GFLAG)

32.12.2.3 eMIOS Output Update Disable (OUDIS)

Address eMIOS base address + 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																
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	OU7	OU6	OU5	OU4	OU3	OU2	OU1	OU0
W																
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 32-5. eMIOS Output Update Disable Register (OUDIS)

Table 32-8. OUDIS Field Descriptions

Field	Descriptions
31–8	Reserved, should be cleared.
7–0 OUn	Channel n Output Update Disable bit. When running MC mode or an output mode, values are written to registers A2 and B2. OUn bits are used to disable transfers from registers A2 to A1 and B2 to B1. Each bit controls one channel. 0 Transfer enabled. Depending on the operation mode, transfer may occur immediately or in the next period. Unless stated otherwise, transfer occurs immediately. 1 Transfers disabled

32.12.2.4 eMIOS Disable Channel (UCDIS)

Address eMIOS base address + 0x0C

	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																
Reset:																

	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	UCDIS7	UCDIS6	UCDIS5	UCDIS4	UCDIS3	UCDIS2	UCDIS1	UCDIS0
W																
Reset:									0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 32-6. eMIOS Enable Channel Register (UCDIS)

Table 32-9. OUDIS Field Descriptions

Field	Descriptions
31–8	Reserved, should be cleared.
7–0 UCDISn	Enable Channel n bit. The UCDISn bit is used to disable each of the Unified Channels by stopping its respective clock. 0 Unified Channel n enabled 1 Unified Channel n disabled

32.12.2.5 eMIOS A Register (UCAn)

Depending on the mode of operation, internal registers A1 or A2, used for matches and captures, can be assigned to address UCAn. Both A1 and A2 are cleared by reset. Figure 32-10 summarizes the UCAn writing and reading accesses for all operation modes. For more information see Section 32.13.1.1, “UC Modes of Operation”.

Address UCn base address + 0x00

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

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	A[15:0]															
W																
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 32-7. eMIOS A Register (UCAn)

32.12.2.6 eMIOS B Register (UCBn)

Depending on the mode of operation, internal registers B1 or B2 can be assigned to address UCBn. Both B1 and B2 are cleared by reset. Table 32-10 summarizes the UCBn writing and reading accesses for all operation modes. For more information see section Section 32.13.1.1, “UC Modes of Operation”.

Address UCn base address + 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																
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	B[15:0]															
W																
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 32-8. eMIOS B register (UCBn)

Table 32-10. UCAn, UCBn and ALTAn values assignment

Operation Mode	Register access				
	write	read	write	read	alt read
GPIO	A1, A2	A1	B1,B2	B1	-
SAIC[1]	-	A2	B2	B2	-

Table 32-10. UCAn, UCBn and ALTAn values assignment

SAOC[1]	A2	A1	B2	B2	-
IPWM	-	A2	-	B1	-
IPM	-	A2	-	B1	-
DAOC	A2	A1	B2	B1	-
PEA	A1	A2	-	B1	-
PEC[1]	A1	A1	B1	B1	A2
QDEC[1]	A1	A1	B2	B2	-
WPTA	A1	A1	B1	B1	A2
MC[1]	A2	A1	B2	B2	-
OPWFM	A2	A1	B2	B1	-
OPWMC	A2	A1	B2	B1	-
OPWM	A2	A1	B2	B1	-
MCB[1]	A2	A1	B2	B2	-
OPWFMB	A2	A1	B2	B1	-
OPWMCB	A2	A1	B2	B1	-
OPWMB	A2	A1	B2	B1	-

1. In these modes, the register UCBn is not used, but B2 can be accessed.

32.12.2.7 eMIOS Counter Register (UCCNTn)

The UCCNTn register contains the value of the internal counter. When GPIO mode is selected or the channel is frozen, the UCCNTn register is read/write. For all others modes, the UCCNTn is a read-only register. When entering some operation modes, this register is automatically cleared (refer to Section 32.13.1.1, “UC Modes of Operation” for details).

Address UCn base address + 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[1]																
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	C[15:0]															
W[1]																
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or reserved

Figure 32-9. eMIOS Counter Register (UCCNTn)

1. In GPIO mode or Freeze action, this register is writable.

32.12.2.8 eMIOS Control Register (UCCRn)

The Control register gathers bits reflecting the status of the UC input/output signals and the overflow condition of the internal counter, as well as several read/write control bits.

Address UCn base address + 0x0C

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	FREN	ODIS	ODISSL[1:0]		UCPRE[1:0]		UCPREN	DMA	0	IF[3:0]				FCK	FEN	0
W																
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	BSL[1:0]		EDSEL	EDPOL	MODE[6:0]						
W			FORCMA	FORCMB												
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or reserved

Figure 32-10. eMIOS Control Register (UCCRn)

Table 32-11. UCCRn Field Descriptions

Field	Descriptions
31 FREN	Freeze Enable bit. The FREN bit, if set and validated by FRZ bit in MCR register, freezes all registers values when in debug mode, allowing the MCU to perform debug functions. 0 Normal operation 1 Freeze UC registers values
30 ODIS	Output Disable bit. The ODIS bit allows disabling the output pin when running any of the output modes with the exception of GPIO mode. 1 If the selected Output Disable Input signal is asserted, the output pin goes to EDPOL for OPWFMB and OPWMB modes and to the complement of EDPOL for other modes, but the Unified Channel continues to operate normally, i.e., it continues to produce FLAG and matches. When the selected Output Disable Input signal is negated, the output pin operates normally 0 The output pin operates normally

Table 32-11. UCCRn Field Descriptions (Continued)

<table>
<tr><td>29–28
ODISSL[1:0]</td><td>The ODISSL[1:0] bits select one of the four output disable input signals, as shown in Table 32-12.
Table 32-12. ODISSL Selection<table><tr><th>ODISSL[1:0]</th><th>input signal</th></tr><tr><td>00</td><td>Output Disable Input 0</td></tr><tr><td>01</td><td>Output Disable Input 1</td></tr><tr><td>10</td><td>Output Disable Input 2</td></tr><tr><td>11</td><td>Output Disable Input 3</td></tr></table></td></tr>
<tr><td>27–26
UCPRE[1:0]</td><td>Prescaler bits . The UCPRE[1:0] bits select the clock divider value for the internal prescaler of Unified Channel, as shown in Table 32-13.
Table 32-13. UC Internal Prescaler Clock Divider<table><tr><th>UCPRE[1:0]</th><th>Divide ratio</th></tr><tr><td>00</td><td>1</td></tr><tr><td>01</td><td>2</td></tr><tr><td>10</td><td>3</td></tr><tr><td>11</td><td>4</td></tr></table></td></tr>
<tr><td>25
UCPREN</td><td>Prescaler Enable bit. The UCPREN bit enables the prescaler counter.
0 Prescaler disabled (no clock) and prescaler counter is loaded with UCPRE[1:0] value
1 Prescaler enabled</td></tr>
<tr><td>24
DMA</td><td>Direct Memory Access bit . The DMA bit selects if the FLAG generation will be used as an interrupt or as a DMA request.
0 FLAG assigned to Interrupt request
1 FLAG assigned to DMA request</td></tr>
<tr><td>23</td><td>Reserved, should be cleared.</td></tr>
</table>

Table 32-11. UCCRn Field Descriptions (Continued)

<table>
<tr><td>22–19
IF[3:0]</td><td>Input Filter bitsThe IF[3:0] bits control the programmable input filter, selecting the minimum input pulse width that can pass through the filter, as shown in the forlowing table. . For output modes, these bits have no meaning.
Table 32-14. Input Filter Bits
<table><tr><th>IF[3:0][1]</th><th>Minimum input Pulse width [FLT_CLK periods]</th></tr><tr><td>0000</td><td>bypassed[2]</td></tr><tr><td>0001</td><td>02</td></tr><tr><td>0010</td><td>04</td></tr><tr><td>0100</td><td>08</td></tr><tr><td>1000</td><td>16</td></tr><tr><td>all others</td><td>reserved</td></tr></table>1. Filter latency is 3 clock edges.
2. The input signal is synchronized before arriving to the digital filter.</td></tr>
<tr><td>18
FCK</td><td>Filter Clock select bit . The FCK bit selects the clock source for the programmable input filter.
0 prescaled clock
1 main clock</td></tr>
<tr><td>17
FEN</td><td>FLAG Enable bit. The FEN bit allows the Unified Channel FLAG bit to generate an interrupt signal or a DMA request signal (The type of signal to be generated is defined by the DMA bit).
0 Disable (FLAG does not generate an interrupt or DMA request)
1 Enable (FlAG will generate an interrupt or DMA request)</td></tr>
<tr><td>16–14</td><td>Reserved, should be cleared.</td></tr>
<tr><td>13
FORCMA</td><td>Force Match A bitFor output modes, the FORCMA bit is equivalent to a successful comparison on comparator A (except that the FLAG bit is not set). This bit is cleared by reset and is always read as zero. This bit is valid for every output operation mode which uses comparator A, otherwise it has no effect.
0 Has no effect
1 Force a match at comparator A
For input modes, the FORCMA bit is not used and writing to it has no effect.</td></tr>
<tr><td>12
FORCMB</td><td>Force Match B bit. For output modes, the FORCMB bit is equivalent to a successful comparison on comparator B (except that the FLAG bit is not set). This bit is cleared by reset and is always read as zero. This bit is valid for every output operation mode which uses comparator B, otherwise it has no effect.
0 Has not effect
1 Force a match at comparator B
For input modes, the FORCMB bit is not used and writing to it has no effect.</td></tr>
<tr><td>11</td><td>Reserved, should be cleared.</td></tr>
</table>

Table 32-11. UCCRn Field Descriptions (Continued)

<table>
<tr><td>10–9
BSL[1:0]</td><td>Bus Select bitsThe BSL[1:0] bits are used to select either one of the counter buses or the internal counter to be used by the Unified Channel. Refer toTable 32-15 for details.

Table 32-15. BSL Bits
<table><tr><th>BSL[1:0]</th><th>selected bus</th></tr><tr><td>00</td><td>All channels: counter bus[A]</td></tr><tr><td>01</td><td>Channels 0 to 7: counter bus[B]</td></tr><tr><td>10</td><td>reserved</td></tr><tr><td>11</td><td>All channels: internal counter</td></tr></table></td></tr>
<tr><td>8
EDSEL</td><td>Edge Selection bit.
For input modes, the EDSEL bit selects whether the internal counter is triggered by both edges of a pulse or just by a single edge as defined by the EDPOL bit. When not shown in the mode of operation description, this bit has no effect.
0 Single edge triggering defined by the EDPOL bit
1 Both edges triggering
For GPIO in mode, the EDSEL bit selects if a FLAG can be generated.
0 A FLAG is generated as defined by the EDPOL bit
1 No FLAG is generated
For SAOC mode, the EDSEL bit selects the behavior of the output flip-flop at each match.
0 The EDPOL value is transferred to the output flip-flop
1 The output flip-flop is toggled</td></tr>
<tr><td>7
EDPOL</td><td>Edge Polarity bit.
For input modes (except QDEC mode), the EDPOL bit asserts which edge triggers either the internal counter or an input capture or a FLAG. When not shown in the mode of operation description, this bit has no effect.
0 Trigger on a falling edge
1 Trigger on a rising edge
For QDEC (MODE[0] cleared), the EDPOL bit selects the count direction according to direction signal (UCn input).
0 counts down when UCn is asserted
1 counts up when UCn is asserted
Note: UC[n-1] EDPOL bit selects which edge clocks the internal counter of UCn
0 Trigger on a falling edge
1 Trigger on a rising edge
For QDEC (MODE[0] set), the EDPOL bit selects the count direction according to the phase difference.
0 internal counter decrements if phase_A is ahead phase_B signal
1 internal counter increments if phase_A is ahead phase_B signal
Note: In order to operate properly, EDPOL bit must contain the same value in UCn and UC[n-1]
For output modes, the EDPOL bit is used to select the logic level on the output pin.
0 A match on comparator A clears the output flip-flop, while a match on comparator B sets it
1 A match on comparator A sets the output flip-flop, while a match on comparator B clears it</td></tr>
<tr><td>6–0
MODE[6:0]</td><td>The MODE[6:0] bits select the mode of operation of the Unified Channel, as shown in Table 32-16.</td></tr>
</table>

32.12.2.9 eMIOS Status Register (UCSRn)

Table 32-16. MODE Bits

MODE[6:0][1]	mode of operation
0000000	General purpose Input/Output mode (input)
0000001	General purpose Input/Output mode (output)
0000010	Single Action Input Capture
0000011	Single Action Output Compare
0000100	Input Pulse Width Measurement
0000101	Input Period Measurement
0000110	Double Action Output compare (with FLAG set on the second match)
0000111	Double Action Output compare (with FLAG set on both match)
0001000	Pulse/Edge Accumulation (continuous)
0001001	Pulse/Edge Accumulation (single shot)
0001010	Pulse/Edge Counting (continuous)
0001011	Pulse/Edge Counting (single shot)
0001100	Quadrature Decode (for count & direction encoders type)
0001101	Quadrature Decode (for phase_A & phase_B encoders type)
0001110	Windowed Programmable Time Accumulation
0001111	Reserved
001000b	Modulus Counter (Up counter)
0010010 0010011	Reserved
00101bb	Modulus Counter (Up/Down counter)
00110b0	Output Pulse Width and Frequency Modulation (immediate update)
00110b1	Output Pulse Width and Frequency Modulation (next period update)
00111b0	Center Aligned Output Pulse Width Modulation (with trail edge dead-time)
00111b1	Center Aligned Output Pulse Width Modulation (with lead edge dead-time)
01000b0	Output Pulse Width Modulation (immediate update)
01000b1	Output Pulse Width Modulation (next period update)
101000b	Modulus Counter Buffered (Up counter)
10101bb	Modulus Counter Buffered (Up/Down counter)
10110b0	Output Pulse Width and Frequency Modulation Buffered
10111b0	Center Aligned Output Pulse Width Modulation Buffered (with trail edge dead-time)
10111b1	Center Aligned Output Pulse Width Modulation Buffered (with lead edge dead-time)

Table 32-16. MODE Bits

MODE[6:0][1]	mode of operation (Continued)
11000b0	Output Pulse Width Modulation Buffered
others	Reserved

1. b = adjust parameters for the mode of operation. Refer to section Section 32.13.1.1, “UC Modes of Operation” for details.

Address UCn base address + 0x10

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	OVR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	OVRC															
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	OVFL	0	0	0	0	0	0	0	0	0	0	0	0	UCIN	UCOUT	FLAG
W	OVFLC															FLAGC
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or reserved

Figure 32-11. eMIOS Status Register (UCSRn)

Table 32-17. UCSRn Field Descriptions

Field	Descriptions
31 OVR OVRC	OVR — Overrun bit The OVR bit indicates that FLAG generation occurred when the FLAG bit was already set. 0 Overrun has not occurred 1Overrun has occurred OVRC — Overrun Clear bit The OVR bit can be cleared either by clearing the FLAG bit or by writing a 1 to the OVRC bit. 0 Do not change OVR bit 1 Clear OVR bit
30–16	Reserved, should be cleared

Table 32-17. UCSRn Field Descriptions (Continued)

15 OVFL OVRC	OVFL — Overflow bit The OVFL bit indicates that an overflow has occurred in the internal counter. OVFL must be cleared by software writing a 1 to the OVFLC bit. 0 No overflow 1 An overflow had occurred OVFLC — Overflow Clear bit The OVFL bit must be cleared by writing a 1 to the OVFLC. 0 Do not change OVFL bit 1 Clear OVFL bit
14–3	Reserved, should be cleared
2 UCIN	Unified Channel Input pin bit The UCIN bit reflects the input pin state after being filtered and synchronized.
1 UCOUT	Unified Channel Output pin bit The UCOUT bit reflects the output pin state.
0 FLAG FLAGC	FLAG — FLAG bit The FLAG bit is set when an input capture or a match event in the comparators occurred. 0 FLAG cleared 1 FLAG set event has occurred **Note:** mts_flag_out reflects the FLAG bit value. When DMA bit is set, the FLAG bit can be cleared by the DMA controller. FLAGC — FLAG Clear bit The FLAG bit must be cleared by writing a 1 to FLAGC. 0 Do not change FLAG bit 1 Clear FLAG bit

32.12.2.10 eMIOS Alternate A Register (ALTAn)

The ALTAn register provides an alternate read only address to access A2 channel registers in PEC and WPTA modes only. If UCAn register is used along with ALTAn, both A1 and A2 registers can be accessed in these modes. Please, see Section 32.13.1.1.8, “Pulse/Edge Counting (PEC) Mode and Section 32.13.1.1.10, “Windowed Programmable Time Accumulation (WPTA) Mode” for a more detailed description of the use of ALTAn register.

Address UCn base address + 0x14

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

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	ALTA[15:0]															
W																
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 32-12. eMIOS Alternate A register (ALTAn)

32.13 Functional Description

The eMIOS provides independent channels (UC) that can be configured and accessed by a host MCU. On the MAC72xx, two time bases can be shared by the channels through two counter buses and each Unified Channel can generate its own time base.

The eMIOS block is reset at positive edge of the clock (synchronous reset). All registers are cleared on reset.

32.13.1 Unified Channel (UC)

Figure 32-13 shows the Unified Channel block diagram. Each Unified Channel consists of:

- Counter bus selector, which selects the time base to be used by the channel for all timing functions
- A programmable clock prescaler
- Two double buffered data registers A and B that allow up to two input capture and/or output compare events to occur before software intervention is needed.
- Two comparators (equal only) A and B, which compares the selected counter bus with the value in the data registers
- Internal counter, which can be used as a local time base or to count input events
- Programmable input filter, which ensures that only valid pin transitions are received by channel
- Programmable input edge detector, which detects the rising, falling or either edges
- An output flip-flop, which holds the logic level to be applied to the output pin
- eMIOS Status and Control register
- An Output Disable Input selector, which selects the Output Disable Input signal that will be used as output disable
- Control state machine (FSM)

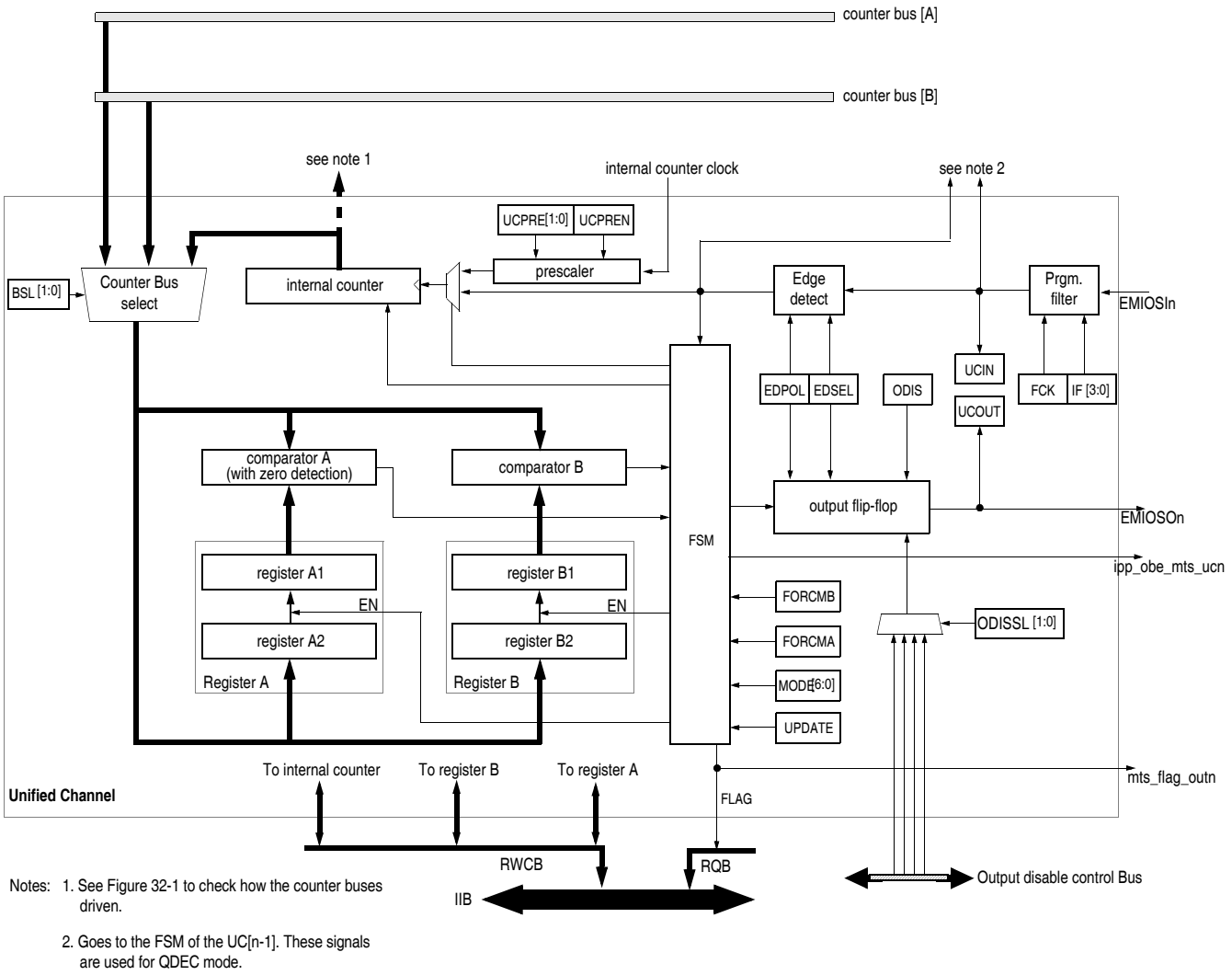


Figure 32-13. Unified Channel Block Diagram

32.13.1.1 UC Modes of Operation

The mode of operation of the Unified Channel is determined by the mode select bits MODE[6:0] in the UCCRn register (see Table 32-16 for details).

When entering an output mode (except for GPIO mode), the output flip-flop is set to the complement of the EDPOL bit in the UCCRn register.

As the internal counter UCCNTn continues to run in all modes (except for GPIO mode), it is possible to use this as a time base if the resource is not used in the current mode.

In order to provide smooth waveform generation even if A and B registers are changed on the fly, it is available the MCB, OPWFMB, OPWMB and OPWMCB modes. In these modes A and B registers are double buffered. They are presented in separated sections since there are several basic differences between these and the MC, OPWFM, OPWM and OPWMC modes, respectively.

32.13.1.1.1 General purpose Input/Output mode (GPIO) Mode

In GPIO mode, all input capture and output compare functions of the UC are disabled, the internal counter (UCCNTn register) is cleared and disabled. All control bits remain accessible. In order to prepare the UC for a new operation mode, writing to registers UCAn or UCBn stores the same value in registers A1/A2 or B1/B2, respectively.

MODE[0] bit selects between input (MODE[0] = 0) and output (MODE[0] = 1) modes.

It is required that when changing MODE[6:0], the application software goes to GPIO mode first in order to reset the UC's internal functions properly. Failure to do this could lead to invalid and unexpected output compare or input capture results or the FLAGs being set incorrectly.

In GPIO input mode, the FLAG generation is determined according to EDPOL and EDSEL bits and the input pin status can be determined by reading the UCIN bit.

In GPIO output mode, the Unified Channel is used as a single output port pin and the value of the EDPOL bit is permanently transferred to the output flip-flop.

32.13.1.1.2 Single Action Input Capture (SAIC) Mode

In SAIC mode, when a triggering event occurs on the input pin, the value on the selected time base is captured into register A2. At the same time, the FLAG bit is set to indicate that an input capture has occurred. Register UCAn returns the value of register A2

The input capture is triggered by a rising, falling or either edges in the input pin, as configured by EDPOL and EDSEL bits in UCCRn register.

Figure 32-14 shows how the Unified Channel can be used for input capture.

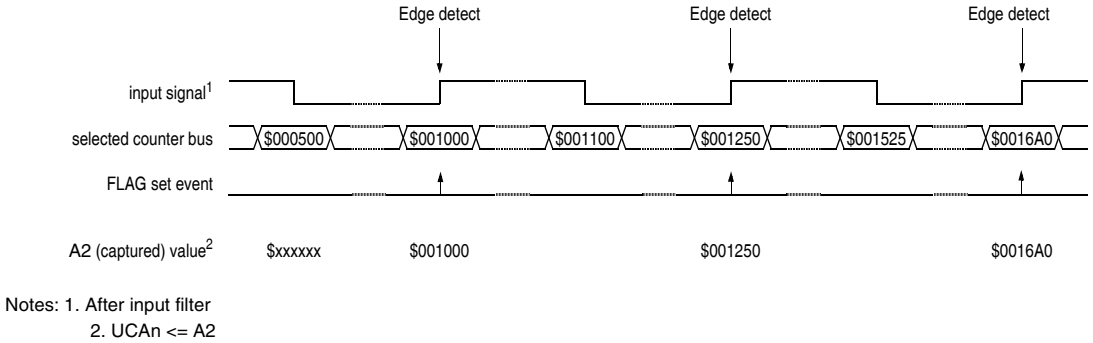


Figure 32-14. Single Action Input Capture example

32.13.1.1.3 Single Action Output Compare (SAOC) Mode

In SAOC mode a match value is loaded in register A2 and then transferred to register A1 to be compared with the selected time base. When a match occurs, the EDSEL bit selects if the output flip-flop is toggled or if the value in EDPOL is transferred to it. At the same time, the FLAG bit is set to indicate that the output compare match has occurred. Writing to register UCAn stores the value in register A2 and reading to register UCAn returns the value of register A1.

An output compare match can be simulated in software by setting the FORCMA bit in UCCRn register. In this case, the FLAG bit is not set.

Figure 32-15 and Figure 32-16 show how the Unified Channel can be used to perform a single output compare with EDPOL value being transferred to the output flip-flop and toggling the output flip-flop at each match, respectively.

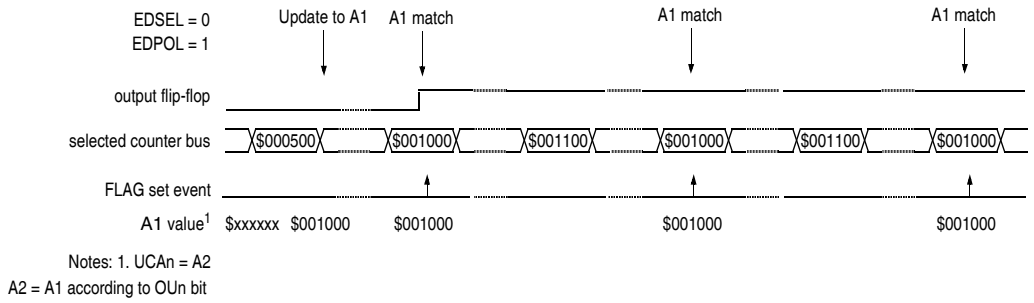


Figure 32-15. SAOC example with EDPOL value being transferred to the output flip-flop

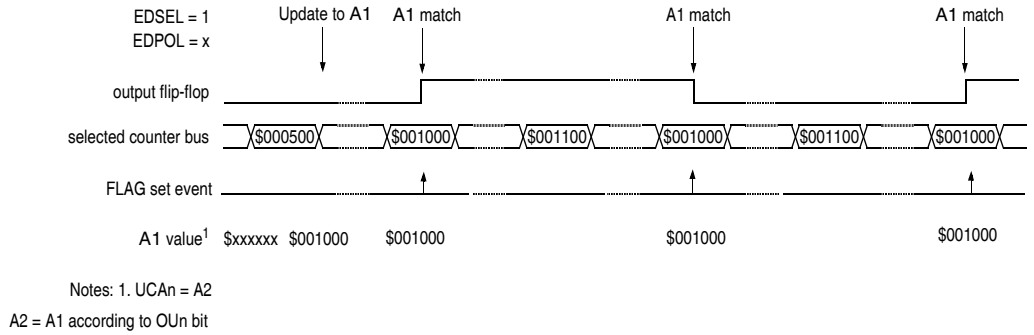


Figure 32-16. SAOC example toggling the output flip-flop

32.13.1.1.4 Input Pulse Width Measurement (IPWM) Mode

The IPWM mode allows the measurement of the width of a positive or negative pulse by capturing the leading edge on register B1 and the trailing edge on register A2. Successive captures are done on consecutive edges of opposite polarity. The leading edge sensitivity (i.e., pulse polarity) is selected by EDPOL bit in the UCCRn register. Registers UCAn and UCBn return the values in register A2 and B1, respectively.

The capture function of register A2 remains disabled until the first leading edge triggers the first input capture on register B2. When this leading edge is detected, the count value of the selected time base is latched into register B2; the FLAG bit is not set. When the trailing edge is detected, the count value of the selected time base is latched into register A2 and, at the same time, the FLAG bit is set and the content of register B2 is transferred to register B1 and to register A1.

If subsequent input capture events occur while the corresponding FLAG bit is set, registers A2, B1 and A1 will be updated with the latest captured values and the FLAG will remain set. Registers UCAn and UCBn return the value in registers A2 and B1, respectively.

In order to guarantee coherent access, reading UCAn forces B1 be updated with the content of register A1. At the same time transfers between B2 and B1 are disabled until the next read of UCBn register. Reading UCBn register forces B1 be updated with A1 register content and re-enables transfers from B2 to B1, to take effect at the next trailing edge capture. Transfers from B2 to A1 are not blocked at any time.

The input pulse width is calculated by subtracting the value in B1 from A2.

Figure 32-17 shows how the Unified Channel can be used for input pulse width measurement.

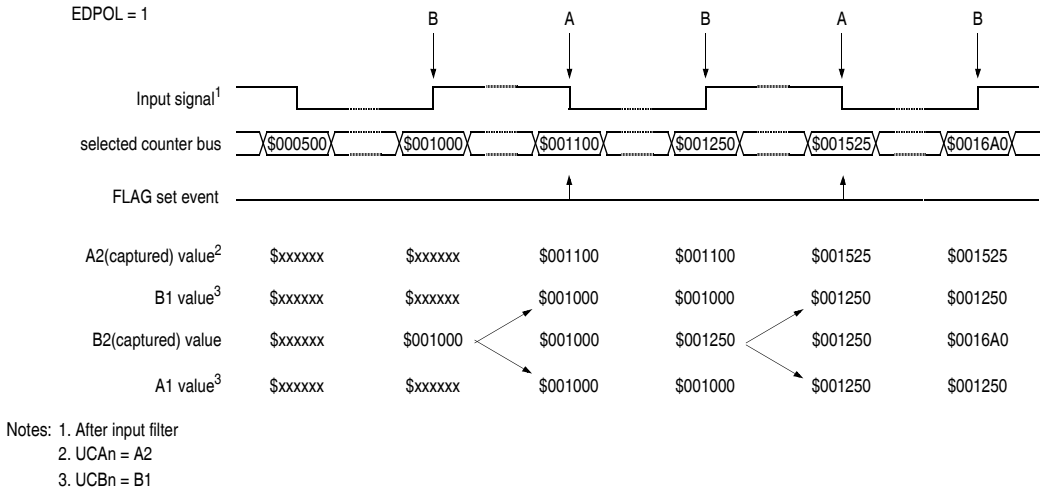


Figure 32-17. Input Pulse Width Measurement example

Figure 32-18 shows the A1 and B1 updates when UCAn and UCBn register reads occur. Note that A1 register has always coherent data related to A2 register. Note also that when UCAn read is performed B1 register is loaded with A1 register content. This guarantee that the data in register B1 has always the coherent data related to the last UCAn read. The B1 register updates remains locked until UCBn read occurs. If UCAn read is performed B1 is updated with A1 register content even if B1 update is locked by a previous UCAn read operation.

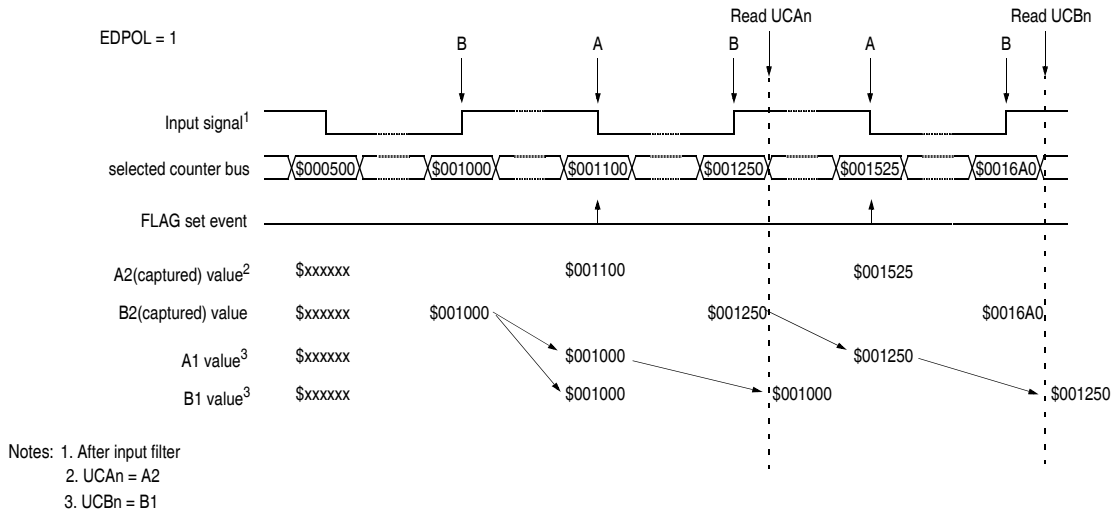


Figure 32-18. B1 and A1 updates at UCAn and UCBn reads

Reading UCAn followed by UCBn always provide coherent data. If not coherent data is required for any reason, the sequence of reads should be inverted, therefore UCBn should be read prior to UCAn register. Note that even in this case B1 register updates will be blocked after UCAn read, thus a second UCBn is required in order to release B1 register updates.

32.13.1.1.5 Input Period Measurement (IPM) Mode

The IPM mode allows the measurement of the period of an input signal by capturing two consecutive rising edges or two consecutive falling edges. Successive input captures are done on consecutive edges of the same polarity. The edge polarity is defined by the EDPOL bit in the UCCRn register.

When the first edge of selected polarity is detected, the selected time base is latched into the registers A2 and B2, and the data previously held in register B2 is transferred to register B1. On this first capture the FLAG line is not set, and the values in registers B1 is meaningless. On the second and subsequent captures, the FLAG line is set and data in register B2 is transferred to register B1.

When the second edge of the same polarity is detected, the counter bus value is latched into registers A2 and B2, the data previously held in register B2 is transferred to data register B1 and to register A1. The FLAG bit is set to indicate the start and end points of a complete period have been captured. This sequence of events is repeated for each subsequent capture. Registers UCAn and UCBn return the values in register A2 and B1, respectively.

In order to allow coherent data, reading UCAn forces A1 content be transferred to B1 register and disables transfers between B2 and B1. These transfers are disabled until the next read of the UCBn register. Reading UCBn register forces A1 content to be transferred to B1 and re-enables transfers from B2 to B1, to take effect at the next edge capture.

The input pulse period is calculated by subtracting the value in B1 from A2.

Figure 32-19 shows how the Unified Channel can be used for input period measurement.

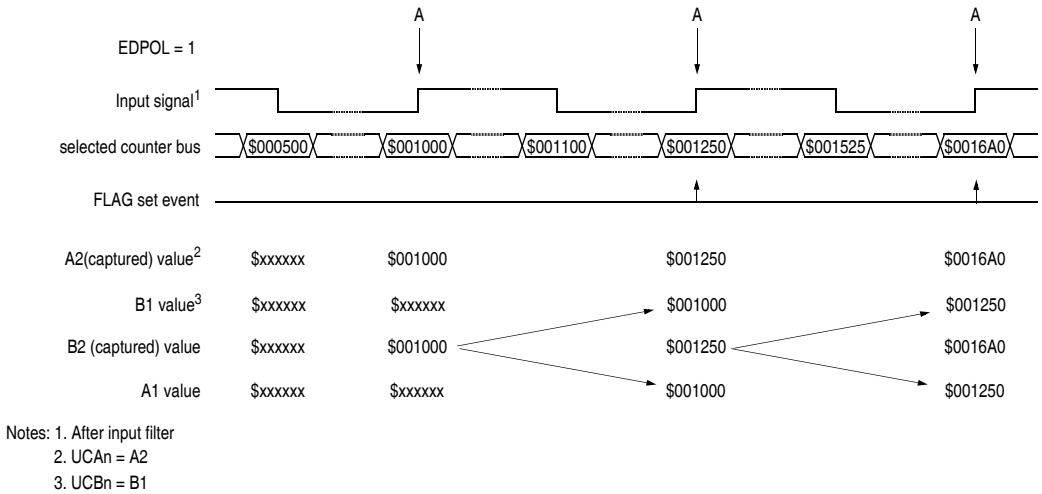


Figure 32-19. Input Period Measurement example

Figure 32-20 describes the A1 and B1 register updates when UCAn and UCBn read operations are performed. When UCAn read occurs the content of A1 is transferred to B1 thus providing coherent data

in A2 and B1 registers. Transfers from B2 to B1 are then blocked until UCBn is read. After UCBn is read, register A1 content is transferred to register B1 and the transfers from B2 to B1 are re-enabled to occur at the transfer edges, which is the leading edge in the Figure 32-20 example.

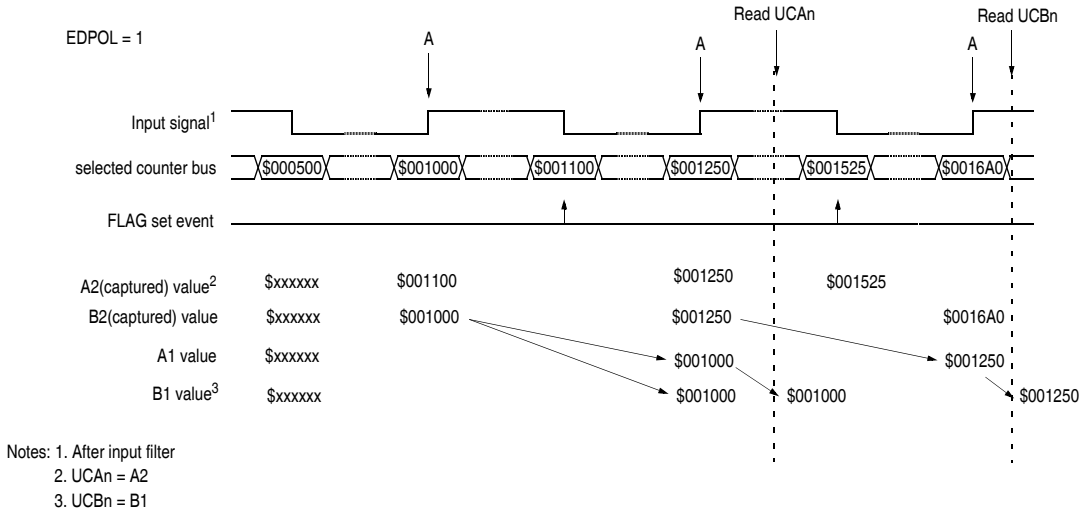


Figure 32-20. A1 and B1 updates at UCAn and UCBn reads

32.13.1.1.6 Double Action Output Compare (DAOC) Mode

In the DAOC mode the leading and trailing edges of the variable pulse width output are generated by matches occurring on comparators A and B, respectively.

When the DAOC mode is first selected (coming from GPIO mode) both comparators are disabled. Comparators A and B are enabled by updating registers A1 and B1 respectively and remain enabled until a match occurs on that comparator, when it is disabled again. In order to update registers A1 and B1, a write to A2 and B2 must occur and the OUDISn bit must be cleared.

The output flip-flop is set to the value of EDPOL when a match occurs on comparator A and to the complement of EDPOL when a match occurs on comparator B.

MODE[0] controls if the FLAG is set on both matches or just on the second match (see Table 32-16 for details).

If subsequent enabled output compares occur on registers A1 and B1, pulses will continue to be generated, regardless of the state of the FLAG bit.

At any time, the FORCMA and FORCMB bits allow the software to force the output flip-flop to the level corresponding to a comparison event in comparator A or B, respectively. Note that the FLAG bit is not affected by these forced operations.

NOTE

If both registers (A1 and B1) are loaded with the same value, the Unified Channel behaves as if a single match on comparator B had occurred, i.e., the output pin will be set to the complement of EDPOL bit and the FLAG bit is set.

Figure 32-21 and Figure 32-22 show how the Unified Channel can be used to generate a single output pulse with FLAG bit being set on the second match or on both matches, respectively.

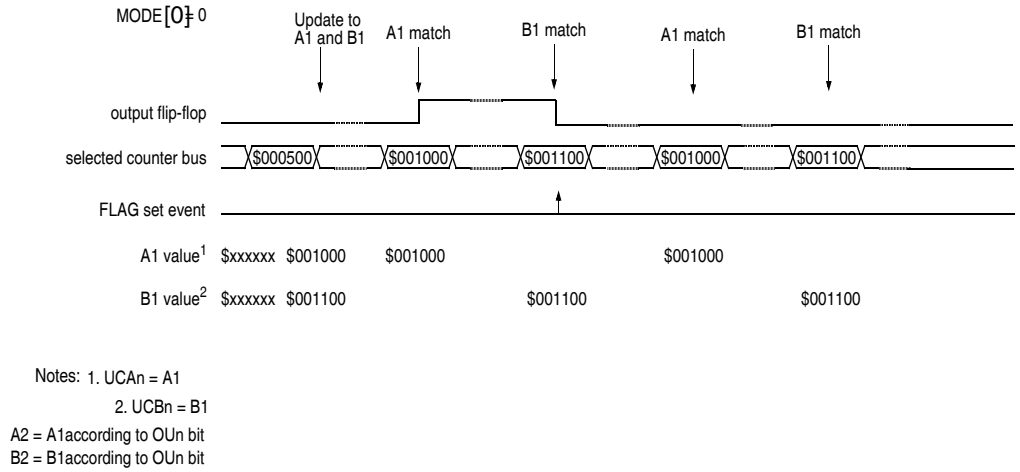


Figure 32-21. Double Action Output Compare with FLAG set on the second match

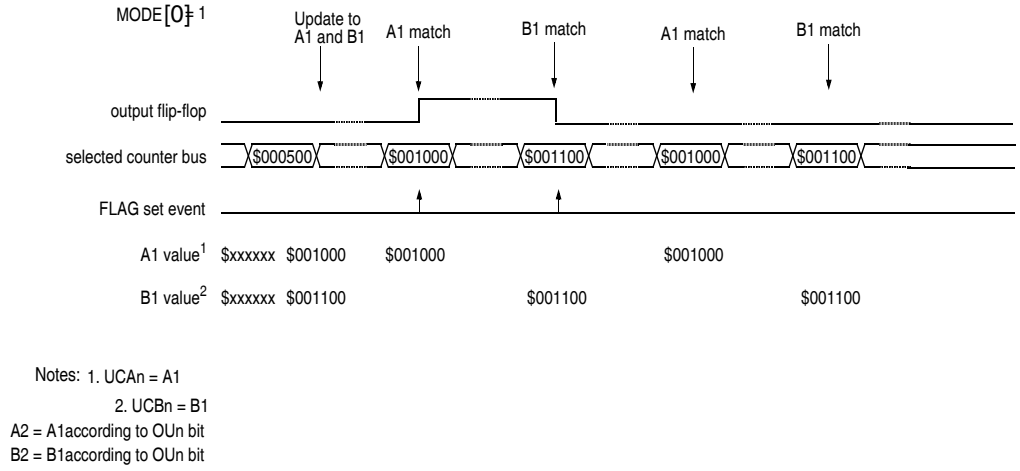


Figure 32-22. Double Action Output Compare with FLAG set on both matches

32.13.1.1.7 Pulse/Edge Accumulation (PEA) Mode

The PEA mode returns the time taken to detect a desired number of input events. MODE[0] bit selects between continuos or single shot operation.

After writing to register A1, the internal counter is cleared on the first input event, ready to start counting input events and the selected timebase is latched into register B2. On the match between the internal counter and register A1, a counter bus capture is triggered to register A2 and B2. The data previously held in register B2 is transferred to register B1 and the FLAG bit is set to indicate that an event has occurred. The desired time interval can be determined subtracting register B1 from A2. Registers UCAn and UCBn return the values in register A2 and B1, respectively.

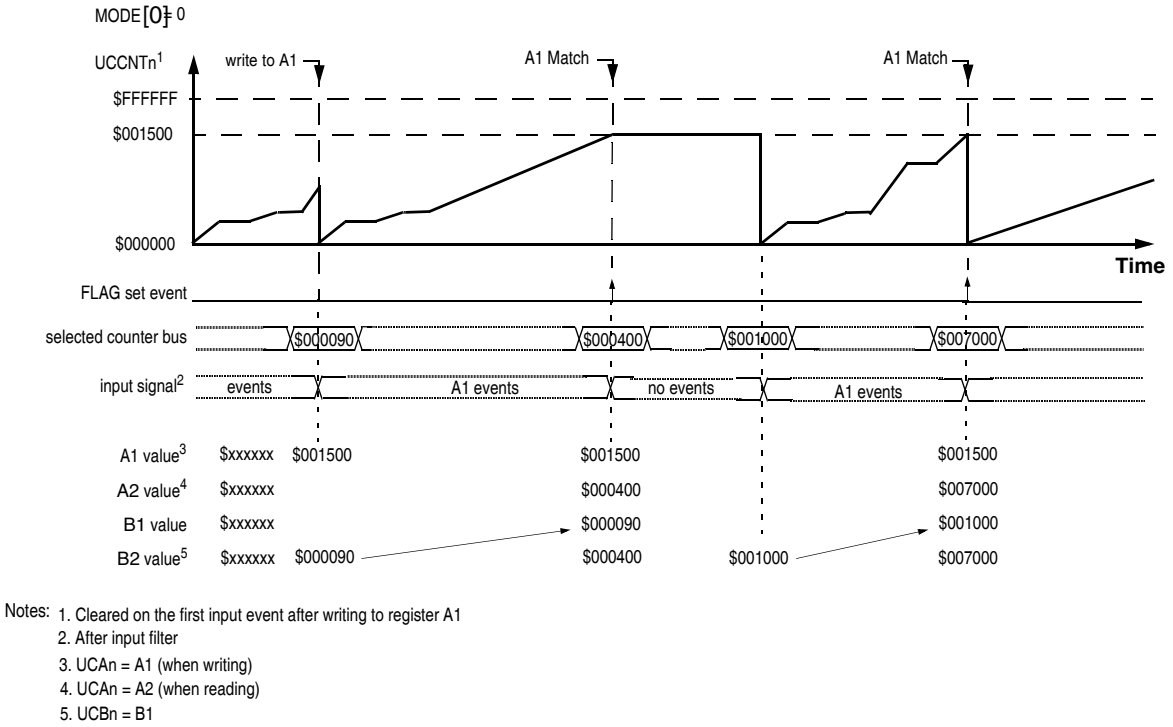
In order to guarantee coherent access, reading UCAn disables transfers between B2 and B1. These transfers are disabled until the next read of the UCBn register. Reading the UCBn register re-enables transfers from B2 to B1, to take effect at the next transfer event, as described above.[1]

Triggering of the counter clock (input event) is done by a rising or falling edge or both edges on the input pin. The polarity of the triggering edge is selected by the EDSEL and EDPOL bits in UCCRn register.

For continuos operation mode (MODE[0] cleared), the counter is cleared on the next input event after a FLAG generation and continues to operate as described above.

For single shot operation (MODE[0] set), the counter is not cleared or incremented after a FLAG generation, until a new writing operation to register A is performed.

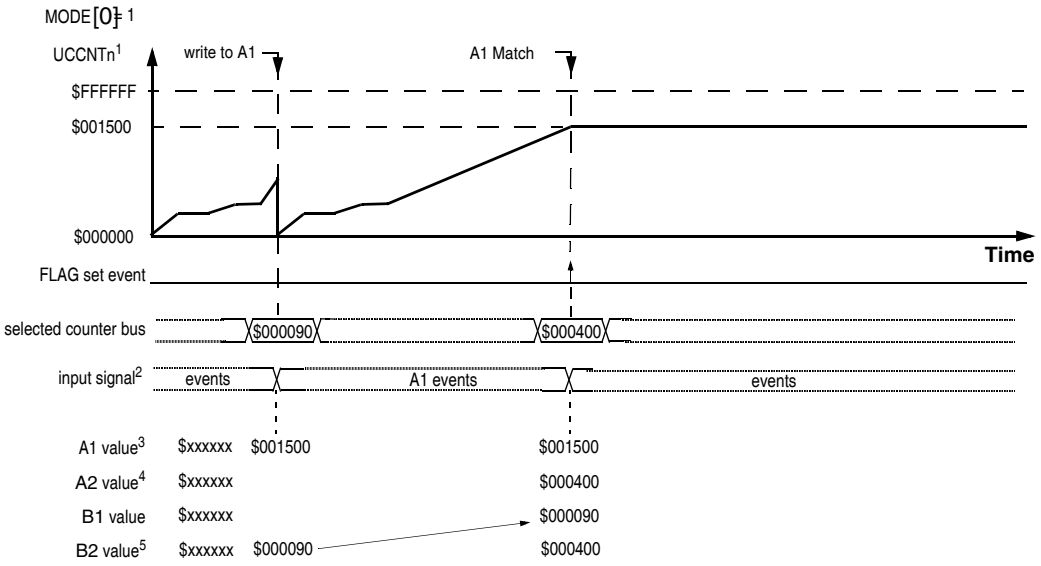
Figure 32-23 and Figure 32-24 show how the Unified Channel can be used for continuos and single shot pulse/edge accumulation mode.



Notes: 1. Cleared on the first input event after writing to register A1
2. After input filter
3. UCAn = A1 (when writing)
4. UCAn = A2 (when reading)
5. UCBn = B1

Figure 32-23. Pulse/Edge Accumulation continuous mode example

1. If B1 was not updated due to B2 to B1 transfer being disabled after reading register UCAn, further UCAn and UCBn reads will not return coherent data until a new bus capture is triggered to registers A2 and B2. This capture event is indicated by the channel FLAG being asserted. If enabled, the FLAG also generates an interrupt.



Notes: 1. Cleared on the first input event after writing to register A1
2. After input filter
3. UCAn = A1 (when writing)
4. UCAn = A2 (when reading)
5. UCBn = B1

Figure 32-24. Pulse/Edge Accumulation single-shot mode example

32.13.1.1.8 Pulse/Edge Counting (PEC) Mode

The PEC mode returns the amount of pulses or edges detected on the input for a desired time window. MODE[0] bit selects between continuos or single shot operation.

Triggering of the internal counter is done by a rising or falling edge or both edges on the input signal. The polarity and the triggering edge is selected by EDSEL and EDPOL bits in UCCRn register.

Register A1 holds the start time and register B1 holds the stop time for the time window. After writing to register A1, when a match occur between comparator A and the selected timebase, the internal counter is cleared and it is ready to start counting input events. When the time base matches comparator B, the internal counter is disabled and its content is transferred to register A2. At the same time the FLAG bit is set. Reading registers MSTCNTn or A2 returns the amount of detected pulses.

For continuos operation (MODE[0] cleared), the next match between comparator A and the selected time base clears the internal counter and counting is enabled again. In order to guarantee coherent measurements when reading UCCNTn after the flag is set, the software must check if the time base value is out of the time interval defined by registers A1 and B1. Alternatively register A2 always holds the latest available measurement providing coherent data at any time after the first FLAG had occurred. This register is addressed by the alternate address ALTAn.

For single shot operation (MODE[0] set), the next match between comparator A and the selected time base has no effect, until a new write to register A is performed. The UCCNTn content is also transferred to register A2 when a match in the B comparator occurs.

Figure 32-25 and Figure 32-26 show how the Unified Channel can be used for continuos or single shot pulse/edge counting mode.

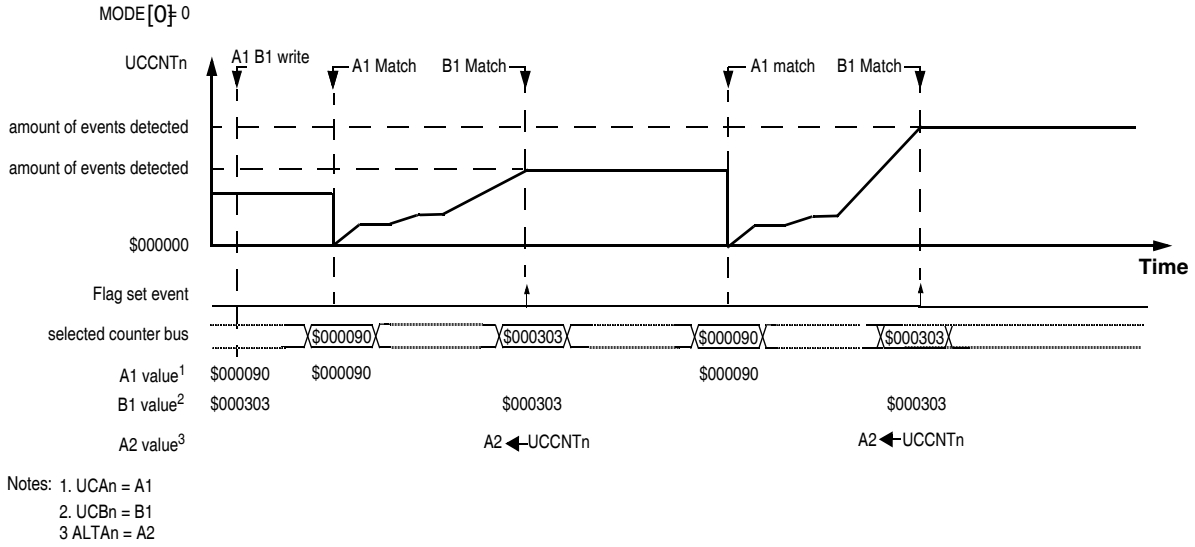


Figure 32-25. Pulse/Edge Counting continuous mode example

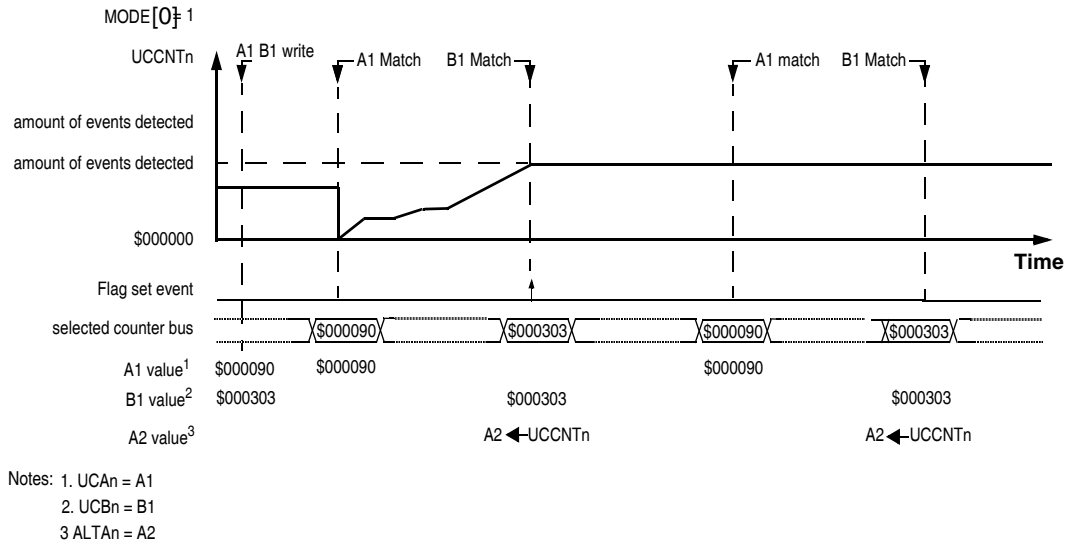


Figure 32-26. Pulse/Edge Counting single-shot mode example

32.13.1.1.9 Quadrature Decode (QDEC) Mode

Quadrature decode mode uses UCn operating in QDEC mode and the input programmable filter (IPF) from UC[n-1]. Note that UC[n-1] can be configured, at the same time, to an operation mode that does not use I/O pins, such as MC mode (modulus counter). The connection among the UCs is circular, i.e., when UC[0] is running in QDEC mode, the input programmable filter from UC[23] is being used.

This mode generates a FLAG every time the internal counter matches A1 register. The internal counter is automatically selected and is not cleared when entering this mode.

MODE[0] bit selects which type of encoder will be used: *count & direction* encoder or *phase_A & phase_B* encoders.

When operating with *count & direction* encoder (MODE[0] cleared), UCn input pin must be connected to the *direction* signal and UC[n-1] input pin must be connected to the *count* signal of the quadrature encoder. UCn EDPOL bit selects count direction according to *direction* signal and UC[n-1] EDPOL bit selects if the internal counter is clocked by the rising or falling edge of the *count* signal.

When operating with *phase_A & phase_B* encoder (MODE[0] set), UCn input pin must be connected to the *phase_A* signal and UC[n-1] input pin must be connected to the *phase_B* signal of the quadrature encoder. EDPOL bit selects the count direction according to the phase difference between *phase_A & phase_B* signals.

Figure 32-27 and Figure 32-28 show two Unified Channels configured to quadrature decode mode for *count & direction* encoder and *phase_A & phase_B* encoders, respectively.

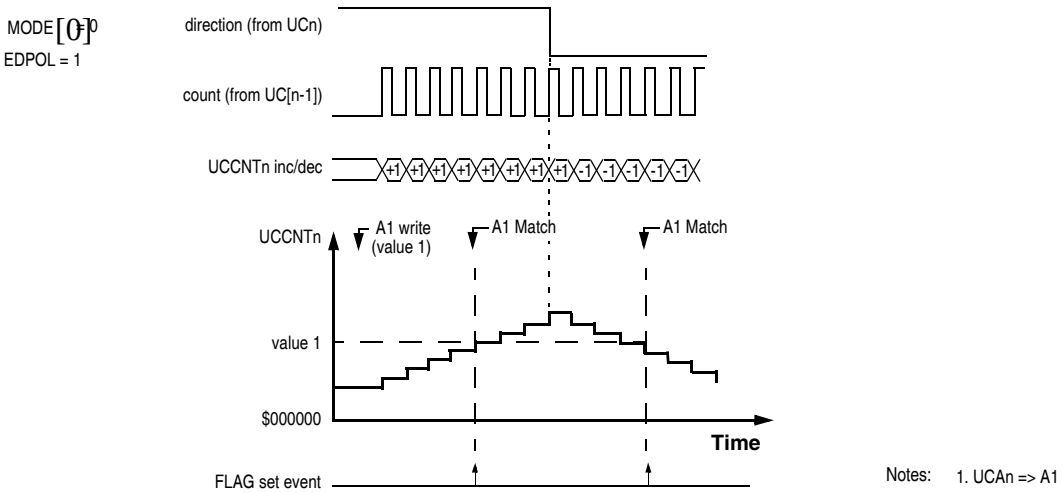


Figure 32-27. Quadrature Decode mode example with *count & direction* encoder

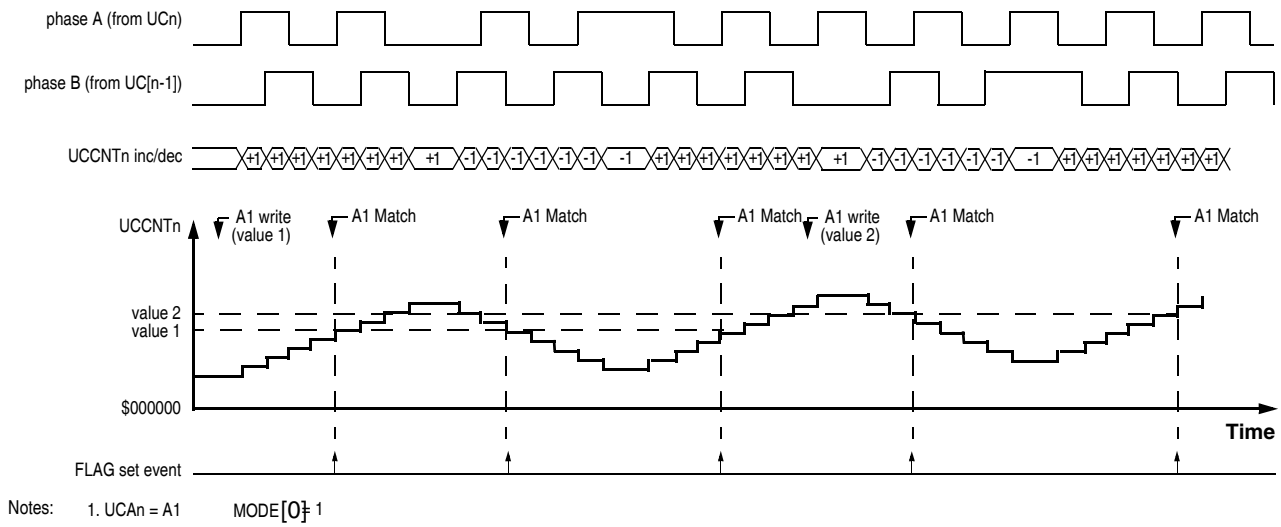


Figure 32-28. Quadrature Decode mode example with *phase_a & phase_B* encoder

32.13.1.1.10 Windowed Programmable Time Accumulation (WPTA) Mode

The WPTA mode accumulates the sum of the total high time or low time of an input signal over a programmable interval (time window).

The prescaler bits UCPRE[1:0] in UCCRn register define the increment rate of the internal counter.

Register A1 holds the start time and register B1 holds the stop time of the programmable time interval. When a match occurs between register A and the selected timebase, the internal counter is cleared and it is ready to start counting. The internal counter is used as a time accumulator, i.e., it counts up when the input signal has the same polarity of EDPOL bit in UCCRn register and does not count otherwise. When a match occurs in comparator B, the internal counter is disabled regardless of the input signal polarity and the FLAG bit is set. At the same time the content of UCCNTn is transferred to register A2. Reading registers UCCNTn or A2 returns the high or low time of the input signal,

Note that UCCNTn is stable only outside the time window defined from A1 to B1 matches, otherwise its contents reflects a count in progress and not the final value. Alternatively to UCCNTn register A2 returns the latest available measurement. Since this register is updated only at comparator B matches it always contains stable and up-to-date data. In this mode this register is accessible through the alternate register address ALTAn.

Figure 32-29 shows how the Unified Channel can be used to accumulate high time.

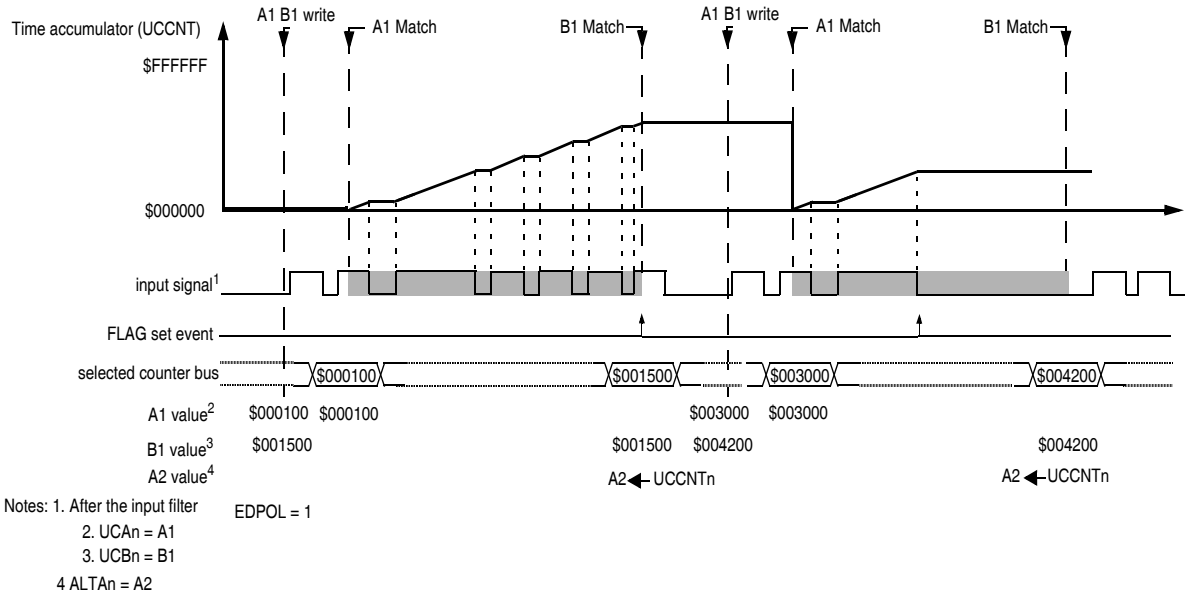


Figure 32-29. Windowed Programmable Time Accumulation example

32.13.1.1.11 Modulus Counter (MC) Mode

The MC mode can be used to provide a time base for a counter bus or as a general purpose timer.

MODE[0] bit selects internal or external clock source when cleared or set, respectively. When external clock is selected, the input signal pin is used as the source and the triggering polarity edge is selected by the EDPOL and EDSEL in the UCCRn register.

The internal counter counts up from the current value until it matches the value in register A1. Register B1 is cleared and is not acessible to the MCU. MODE[2] bit selects up mode or up/down mode, when cleared or set, respectivelly.

When in up count mode, a match between the internal counter and register A1 sets the FLAG and clears the internal counter.

When in up/down count mode, a match between the internal counter and register A1 sets the FLAG and changes the counter direction from increment to decrement. A match between register B1 and the internal counter changes the counter direction from decrement to increment and sets the FLAG only if MODE[1] bit is set.

Figure 32-30 and Figure 32-31 show how the Unified Channel can be used as modulus counter in up mode and up/down mode, respectively.

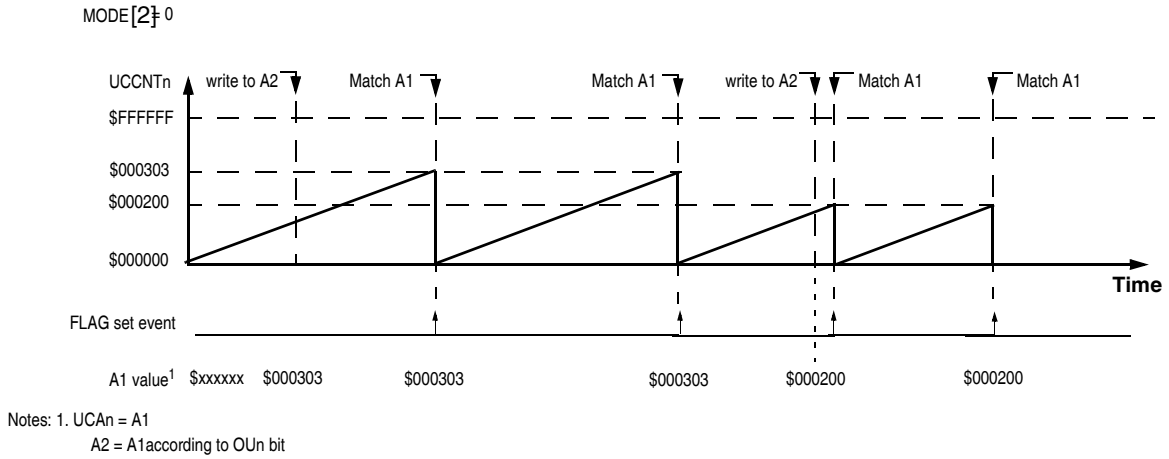


Figure 32-30. Modulus Counter up mode example

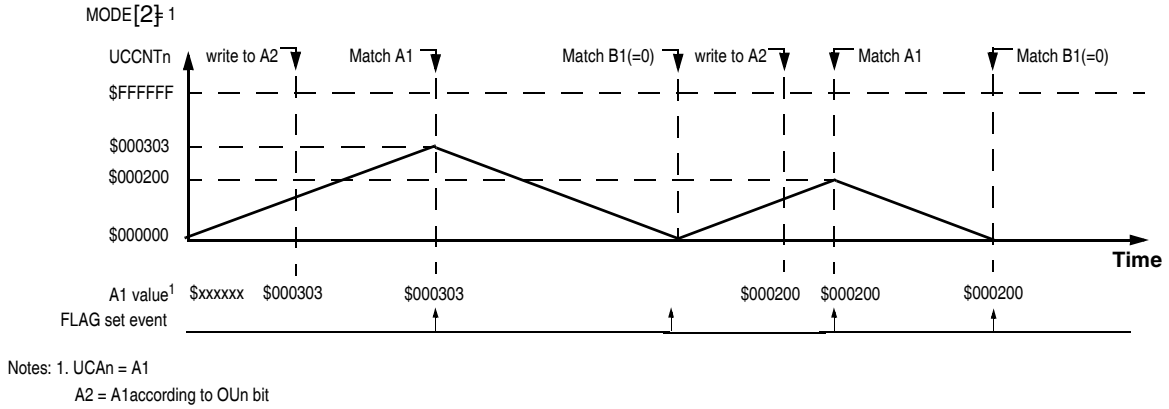


Figure 32-31. Modulus Counter up/down mode example

32.13.1.1.12 Modulus Counter Buffered (MCB) Mode

The MCB mode provides a time base which can be shared with other channels through the internal counter buses. Register A1 is double buffered thus allowing smooth transitions between cycles when changing A2

register value on the fly. A1 register is updated at the cycle boundary, which is defined as when the internal counter reaches the value one. Note that the internal counter values are within a range from one up to register A1 value in MCB mode.

MODE[0] bit selects internal clock source if set to zero or external, if set to one. When external clock is selected the input channel pin is used as the channel clock source. The active edge of this clock is defined by EDPOL and EDSEL bits in the UCCRn channel register.

When entering in MCB mode, if up counter is selected by MODE[2]=0, the internal counter starts counting from its current value to up direction until A1 match occurs. On the next system clock cycle after A1 match occurs the internal counter is set to one. If up/down counter is selected by setting MODE[2]=1, the counter changes direction at A1 match and counts down until it reaches the value one. After it have reached one it is set to count in up direction again. Register B1 is set to one at mode entering and cannot be changed while this mode is selected. B1 register is used to generate a match in order to set the internal counter in up-count direction if up/down mode is selected.

Note that differently from the MC mode, the MCB mode counts between one and A1 register value. The counter cycle period is equal to A1 value in up counter mode. If in up/down counter mode the period is defined by the expression: (2*A1)-2.

Figure 32-32 describes the counter cycle for several A1 values. Register A1 is loaded with A2 register value at the cycle boundary. Thus any value written to A2 register within cycle (n) will be updated to A1 at the next cycle boundary and therefore will be used on cycle (n+1). The cycle boundary between cycle (n) and cycle (n+1) is defined as the first system clock cycle of cycle (n+1). Note that the flags are generated as soon as A1 match had occurred.

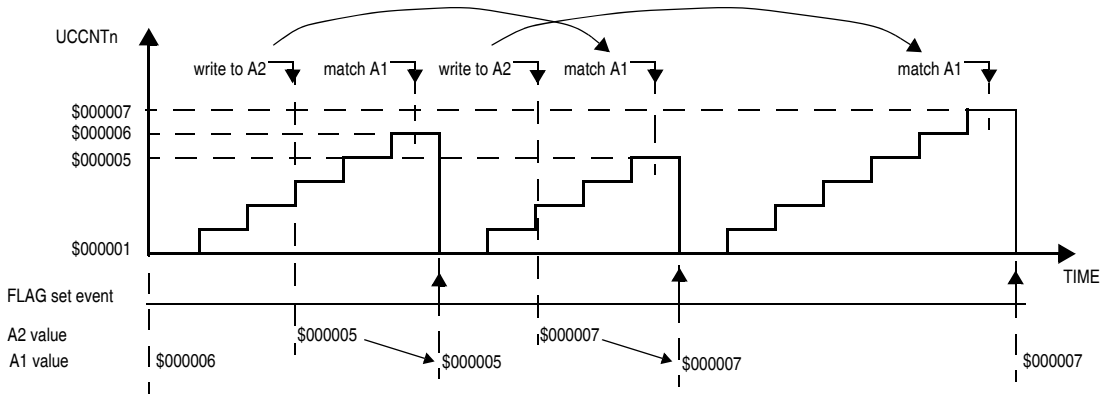


Figure 32-32. Modulus Counter Buffered (MCB) Up Count mode

Figure 32-33 describes the MCB in up/down counter mode. A1 register is updated at the cycle boundary. If A2 is written in cycle (n), this new value will be used in cycle (n+1) for A1 match.

Flags are generated only at A1 match if MODE[1] is 0. If MODE[1] is set to 1 flags are also generated at the cycle boundary.

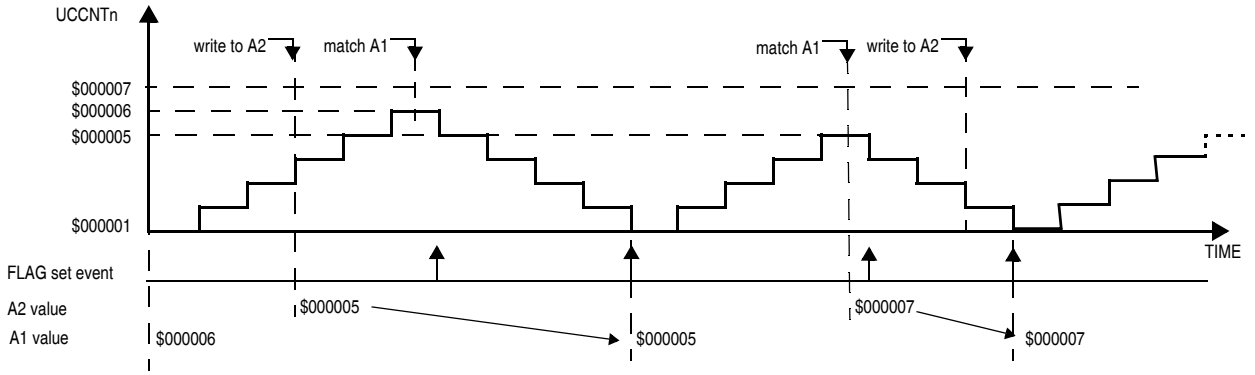


Figure 32-33. Modulus Counter Buffered (MCB) Up/Down Mode

Figure 32-34 describes in more detail the A1 register update process in up counter mode. The A1 load signal is generated based on the detection of the internal counter reaching one and has the duration of one system clock cycle. Note that during the load pulse A1 still holds its previous value. It is actually updated only at the second system clock cycle.

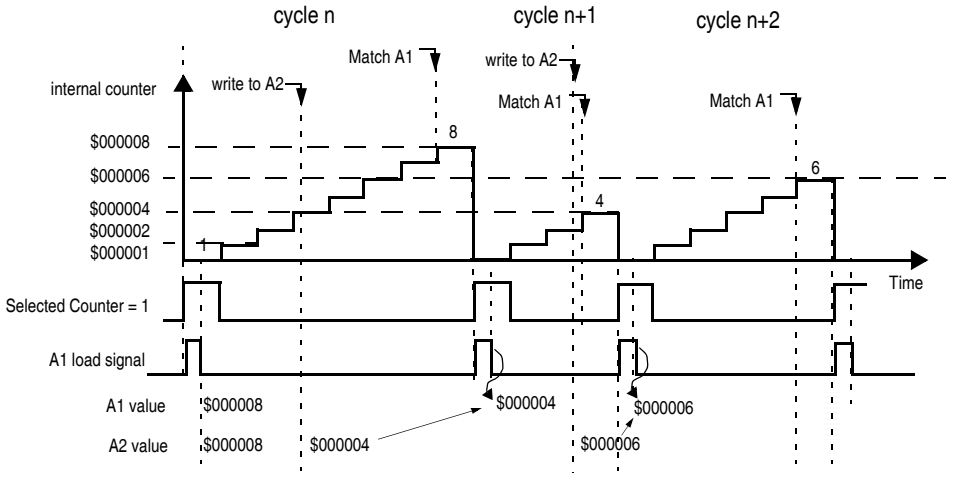


Figure 32-34. MCB Mode A1 Register Update in Up Counter Mode

Figure 32-35 describes the A1 register update in up/down counter mode. Note that A2 can be written at any time within cycle (n) in order to be used in cycle (n+1). Thus A1 receives this new value at the next cycle boundary. Note that the update disable bits OUDISn can be used to disable the update of A1 register.

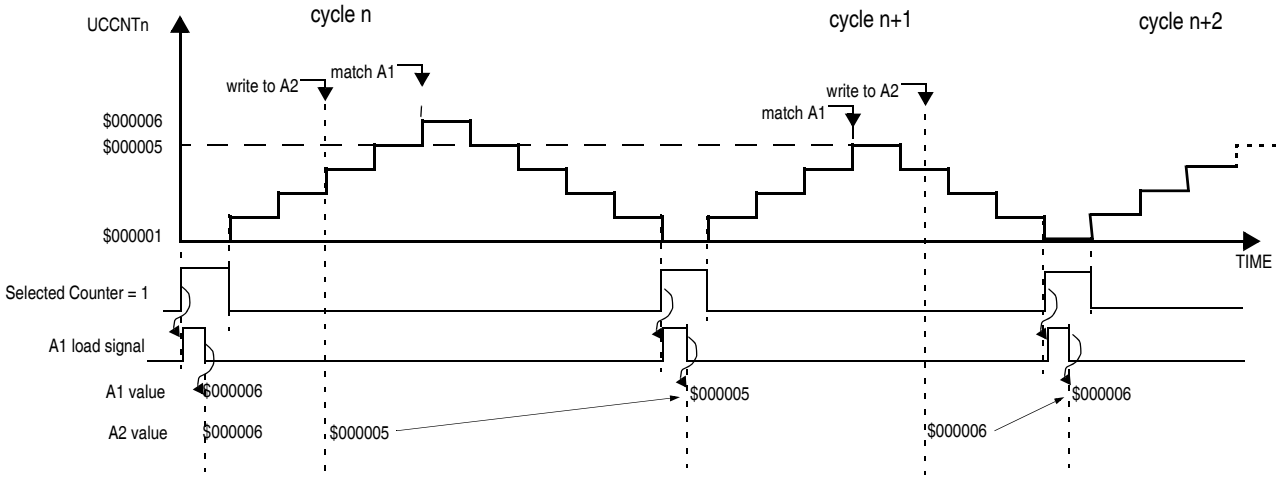


Figure 32-35. MCB Mode A1 Register Update in Up/Down Counter Mode

32.13.1.1.13 Pulse Width and Frequency Modulation (OPWFM) Mode

In this mode, duty cycle of output signal is the value defined in register A1 plus one and the period is the value defined in register B1 plus one. MODE[0] bit controls the transfer from register B2 to B1, which can be done either immediately (MODE[0] cleared), providing the fastest change in the duty cycle, or at every match of register A1 (MODE[0] set).

The internal counter is automatically selected as a time base, therefore the BSL[1:0] bits in register UCCRn has no meaning. When a match on comparator A occurs, the output flip-flop is set to the value of the EDPOL bit. When a match occurs on comparator B, the output flip-flop is set to the complement of the EDPOL bit and the internal counter is cleared.

FLAG can be generated at match B, when MODE[1] is cleared, or in both matches, when MODE[1] is set.

At any time, the FORCMA and FORCMB bits allow the software to force the output flip-flop to the level corresponding to a match on A or B respectively. Also, FORCMB clears the internal counter. Note that the FLAG bit is not set by the FORCMA or FORCMB operations.

If subsequent comparisons occur on comparators A and B, the PWFM pulses continue to be output, regardless of the state of the FLAG bit.

In order to achieve 100% duty cycle, both registers A1 and B1 must be set to the same value. When a simultaneous match occurs on comparators A and B, the output flip-flop is set at every period to the value of EDPOL bit. 0% duty cycle is possible by writing 0x000000 to register A. When a match occurs, the output flip-flop is set at every period to the complement of EDPOL bit. The transfer from register B2 to B1 is still controlled by MODE[0] bit.

NOTE

Writing 0x000000 to A1 and B1 produces a duty cycle of 0%.

Figure 32-36 shows the Unified Channel running in OPFWM mode with immediate register update and Figure 32-37 shows the Unified Channel running in OPFWM mode with next period update PFWM mode.

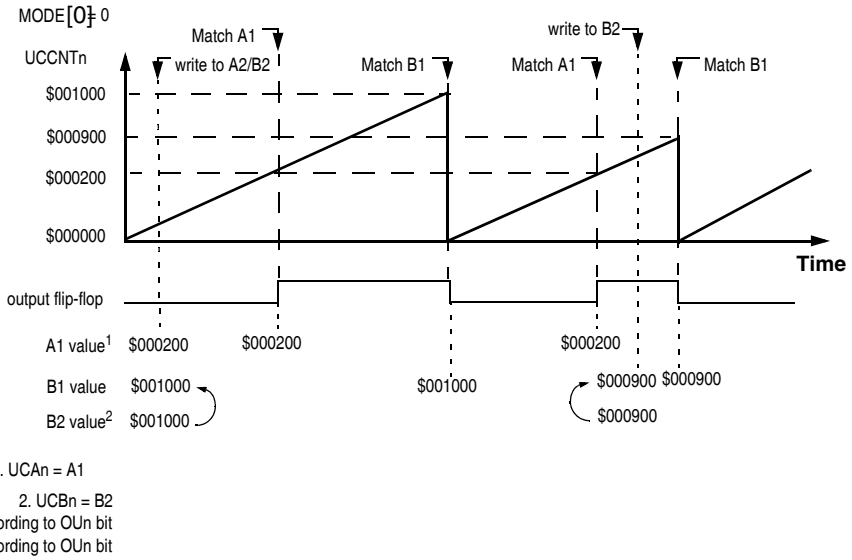


Figure 32-36. OPWFM with immediate update

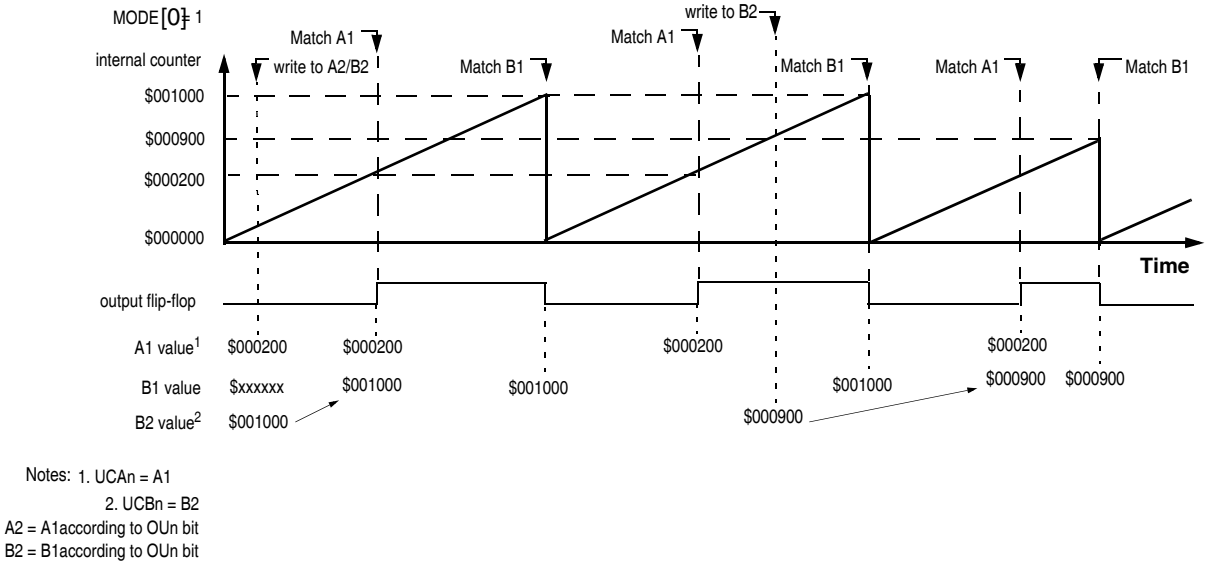


Figure 32-37. OPWFM with next period update

32.13.1.1.14 Pulse Width and Frequency Modulation Buffered (OPWFMB) Mode

This mode provides waveforms with variable duty cycle and frequency. The internal channel counter is automatically selected as the time base when this mode is selected. A1 register indicates the duty cycle and B1 register the frequency. Both A1 and B1 registers are double buffered to allow smooth signal generation when changing the registers values on the fly. 0% and 100% duty cycles are supported.

In order to provide smooth and consistent channel operation this mode differs substantially from the OPWFM mode. The main differences reside in the A1 and B1 registers update, on the delay from the A1 match to the output pin transition and on the range of the internal counter values which starts from 1 up to B1 register value.

When a match on comparator A occurs the output register is set to the value of EDPOL. When a match on comparator B occurs the output register is set to the complement of EDPOL. B1 match also causes the internal counter to transition to 1, thus re-starting the counter cycle.

Figure 32-38 describes the operation of the OPWFMB mode regarding output pin transitions and A1/B1 registers match events. Note that the output pin transition occurs when the A1 or B1 match signal is deasserted which is indicated by the A1 match negedge detection signal. If register A1 is set to 0x000004 the output pin transitions 4 counter periods after the cycle had started, plus one system clock cycle. Note that in the example shown in Figure 32-38 the internal counter prescaler is set to two.

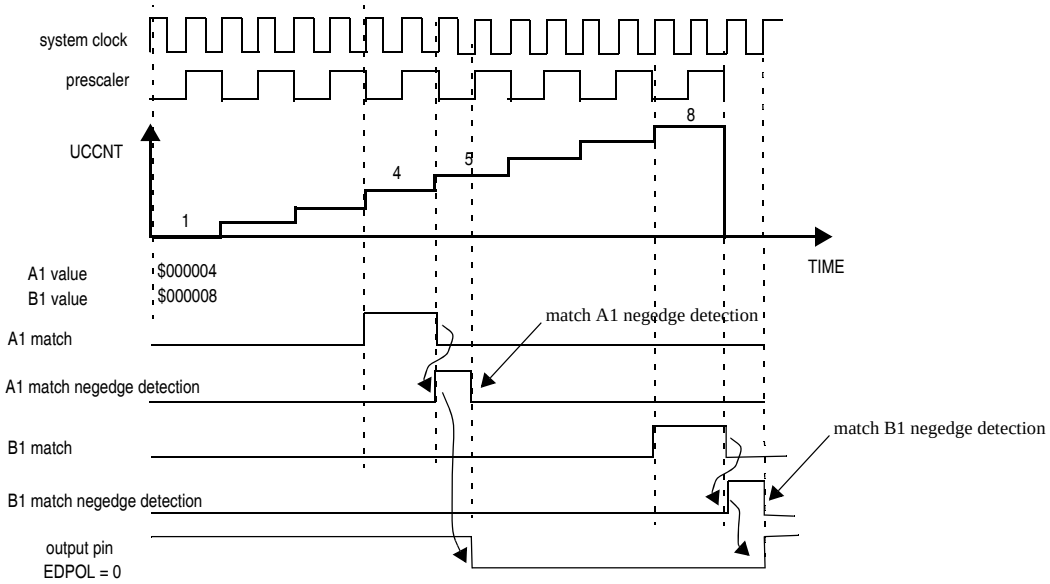


Figure 32-38. OPWFMB A1 and B1 match to Output Register Delay

Figure 32-39 describes the generated output signal if A1 is set to 0. Since the counter does not reach zero in this mode, the channel internal logic infers a match as if A1=1 with the difference that in this case, the posedge of the match signal is used to trigger the output pin transition instead of the posedge used when A1=1. Note that A1 posedge match signal from cycle (n+1) occurs at the same time as B1 negedge match signal from cycle (n). This allows to use the A1 posedge match to mask the B1 negedge match when they occur at the same time. The result is that no transition occurs on the output flip-flop and a 0% duty cycle is generated.

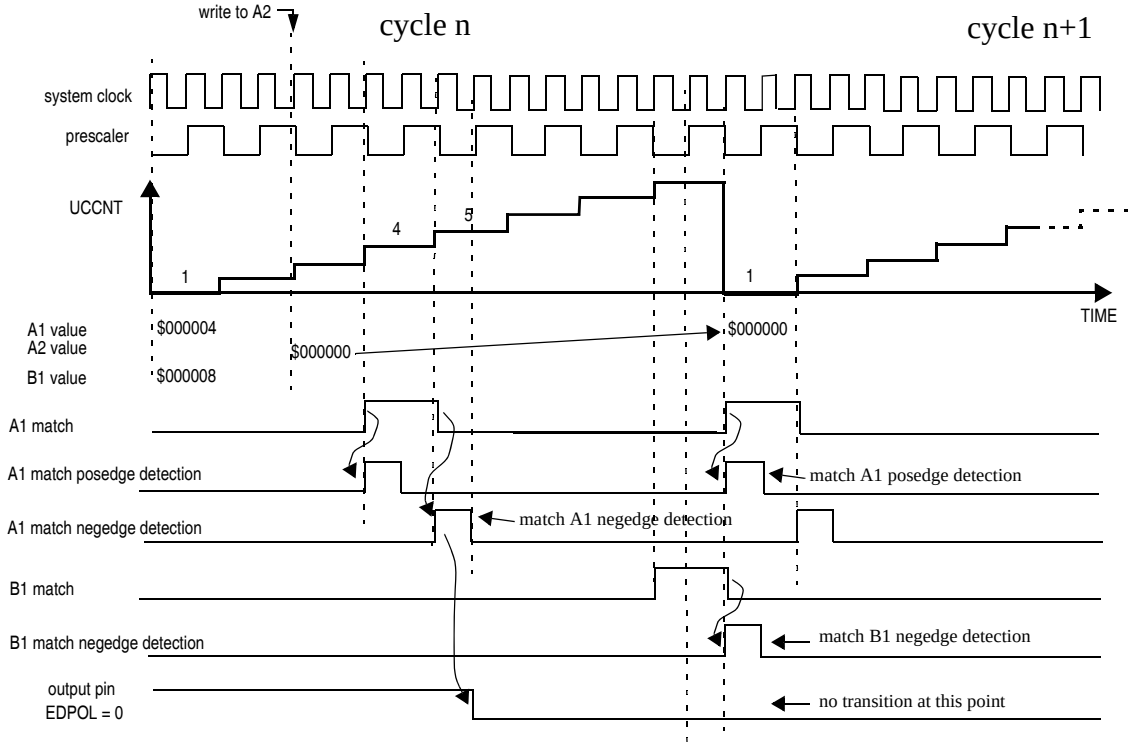


Figure 32-39. OPWFMB Mode with A1 = 0 (0% duty cycle)

Figure 32-40 describes the timing for the A1 and B1 registers load. The A1 and B1 load use the same signal which is generated based on the selected counter reaching the value one, or UCCNTn = 1. This event is defined as the cycle boundary. The load signal pulse has the duration of one system clock cycle and occurs at the first system clock period of every counter cycle. If A2 and B2 are written within cycle (n) their values are loaded into A1 and B1, respectively, at the first clock of cycle (n+1). The update disable bits OUDISn can be used to control the update of these registers, thus allowing to delay the A1 and B1 registers update for synchronization purposes.

During the load pulse A1 still holds its old value, which is updated only at the following system clock cycle. During the A1 load pulse an internal by-pass allows to use the A2 value instead of the A1 value for matches if A2 is either 0 or 1, thus allowing to generate matches even when A1 register is being loaded. This approach allows a uniform channel operation for any A2 value, including 1 and 0.

In Figure 32-40 it is assumed that the channel and global prescalers are set to one, meaning that the channel internal counter transition at every system clock cycle. FLAGs can be generated only on B1 matches when MODE[1] is cleared, or on both A1 and B1 matches when MODE[1] is set. Since B1 FLAG occurs at the cycle boundary, this flag can be used to indicate that A2 or B2 data written on cycle (n) were loaded to A1 or B1, respectively, thus generating matches in cycle (n+1).

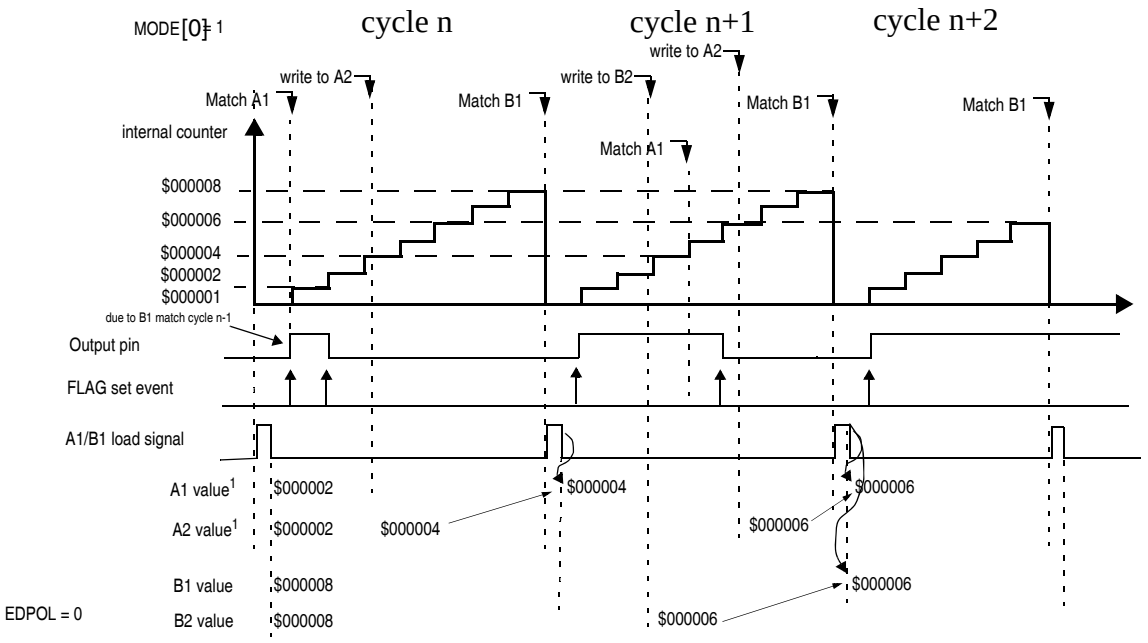


Figure 32-40. OPWFMB A1 and B1 Registers Update and Flags

Figure 32-41 describes the operation of the Output Disable feature in OPWFMB mode. Differently from the OPWFM mode, the output disable forces the channel output flip-flop to EDPOL bit value. This functionality targets applications that use active high signals and a high to low transition at A1 match. In this case EDPOL should be set to 0.

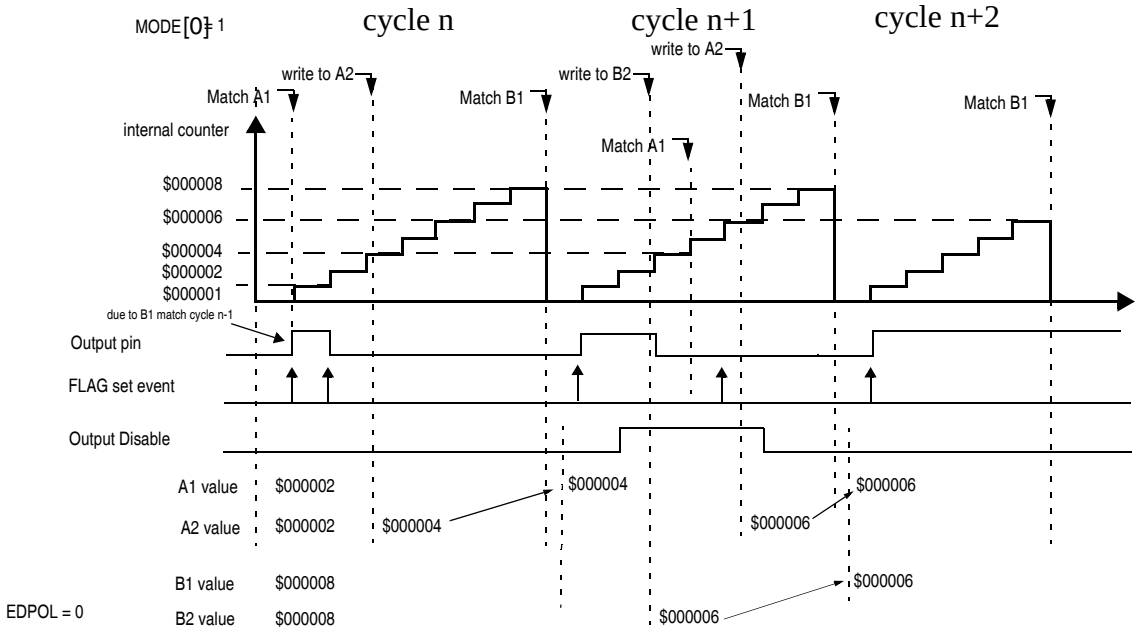


Figure 32-41. OPWFMB Mode with Active Output Disable

Note that the output disable has a synchronous operation, meaning that the assertion of the Output Disable input pin causes the channel output flip-flop to transition to EDPOL at the next system clock cycle. If the Output Disable input is deasserted the output pin transition at the following A1 or B1 match.

In Figure 32-41 it is assumed that the Output Disable input is enabled and selected for the Channel. Please, refer to Section 32.12.2.8, “eMIOS Control Register (UCCRn)” for a detailed description about the ODIS and ODISSL bits, respectively enable and selection of the Output Disable inputs.

The FORCMA and FORCMB bits allow the software to force the output flip-flop to the level corresponding to a match on comparators A or B respectively. Similar to a B1 match FORCMB sets the internal counter to 0x000001. The FLAG bit is not set by the FORCMA or FORCMB bits being asserted.

Figure 32-42 describes the generation of 100% and 0% duty cycle signals. It is assumed EDPOL =0 and the resultant prescaler value is 1. Initially A1=0x000008 and B1=0x000008. In this case, B1 match has precedence over A1 match, thus the output flip-flop is set to the complement of EDPOL bit. This cycle corresponds to a 100% duty cycle signal. The same output signal can be generated for any A1 value greater or equal to B1.

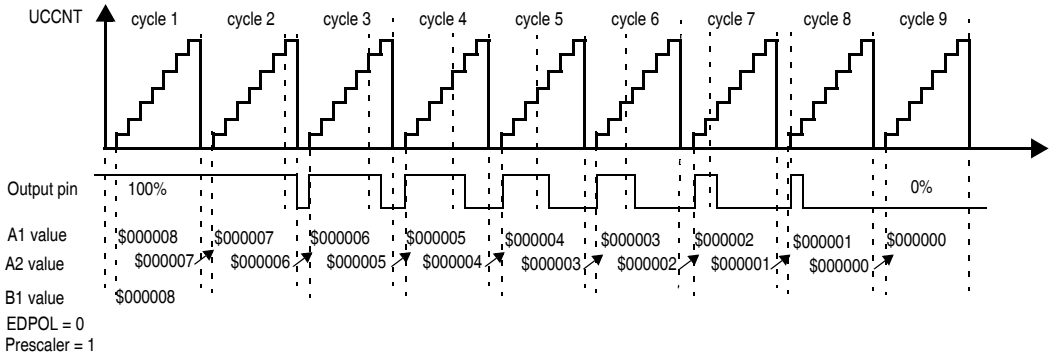


Figure 32-42. OPWFMB Mode from 100% to 0% Duty Cycle

A 0% duty cycle signal is generated if A1=0 as shown in Figure 32-42 cycle 9. In this case B1=0x000008 match from cycle 8 occurs at the same time as the A1=0x000000 match from cycle 9. Please, refer to Figure 32-39 for a description of the A1 and B1 match generation. In this case A1 match has precedence over B1 match and the output signal transitions to EDPOL.

32.13.1.1.15 Center Aligned Output Pulse Width Modulation with Dead-Time (OPWMC) Mode

This operation mode generates a center aligned PWM with dead time insertion in the leading or trailing edge. The selected counter bus must be running an up/down time base, as shown in Figure 32-31. BSL[1:0] bits select the time base. Register A1 contains the ideal duty cycle for the PWM signal and is compared with the selected time base. Register B1 contains the dead time value and is compared with the internal counter. For a leading edge dead time insertion, the output PWM duty cycle is equal to the difference between register A1 and register B1, and for a trailing edge dead time insertion, the output PWM duty cycle is equal to the sum of register A1 and register B1. Mode[0] bit selects between trailing and leading dead time insertion, respectively.

NOTE

The internal counter may be running in the internal prescaler ratio, while the selected time base may be running in a different prescaler ratio. The output signal may produce an unexpected output if the dead time interval is greater than the duty cycle of the PWM signal.

When operating with leading edge dead time insertion, the first match between A1 and the selected time base clears the internal counter and switches the selected time base to the internal counter. When a match occurs between register B1 and the selected time base, the output flip-flop is set to the value of the EDPOL bit and the time base is switched to the selected counter bus. In the next match between register A1 and the selected time base, the output flip-flop is set to the complement of the EDPOL bit. This sequence repeats continuously.

When operating with trailing edge dead time insertion, the first match between A1 and the selected time base sets the output flip-flop to the value of the EDPOL bit. In the next match between register A1 and the selected time base, the internal counter is cleared and the selected time base is switched to the internal counter. When a match occurs between register B1 and the selected time base, the output flip-flop is set to the complement of the EDPOL bit and the time base is switched to the selected counter bus. This sequence repeats continuously.

FLAG can be generated in the trailing edge of the output PWM signal when MODE[1] is cleared, or in both edges, when MODE[1] is set.

At any time, the FORCMA or FORCMB bits are equivalent to a successful comparison on comparator A or B with the exception that the FLAG bit is not set.

NOTE

When in freeze mode, the FORCMA or FORCMB bits only allow the software to force the output flip-flop to the level corresponding of a match on A or B respectively.

If subsequent matches occur on comparators A and B, the PWM pulses continue to be generated, regardless of the state of the FLAG bit.

In order to achieve a duty cycle of 100%, both registers A1 and B1 must be set to the same value. When a simultaneous match occurs between the selected time base and registers A1 and B1, the output flip-flop is set at every period to the value of EDPOL bit and the selected time base switches to the selected counter bus, allowing a new cycle to begin at any time, as previously described. 0% duty cycle is possible by writing 0x000000 to register A. When a match occurs, the output flip-flop is set at every period to the complement of EDPOL bit and the selected time base switches to the selected counter bus, allowing a new cycle to begin at any time, as previously described. In both cases, FLAG is generated regardless of MODE[1] bit.

NOTE

If A1 and B1 are set to the 0x000000, a 0% duty cycle waveform is produced.

Figure 32-43 and Figure 32-44 show the Unified Channel running in OPWMC with leading and trailing dead time, respectively.

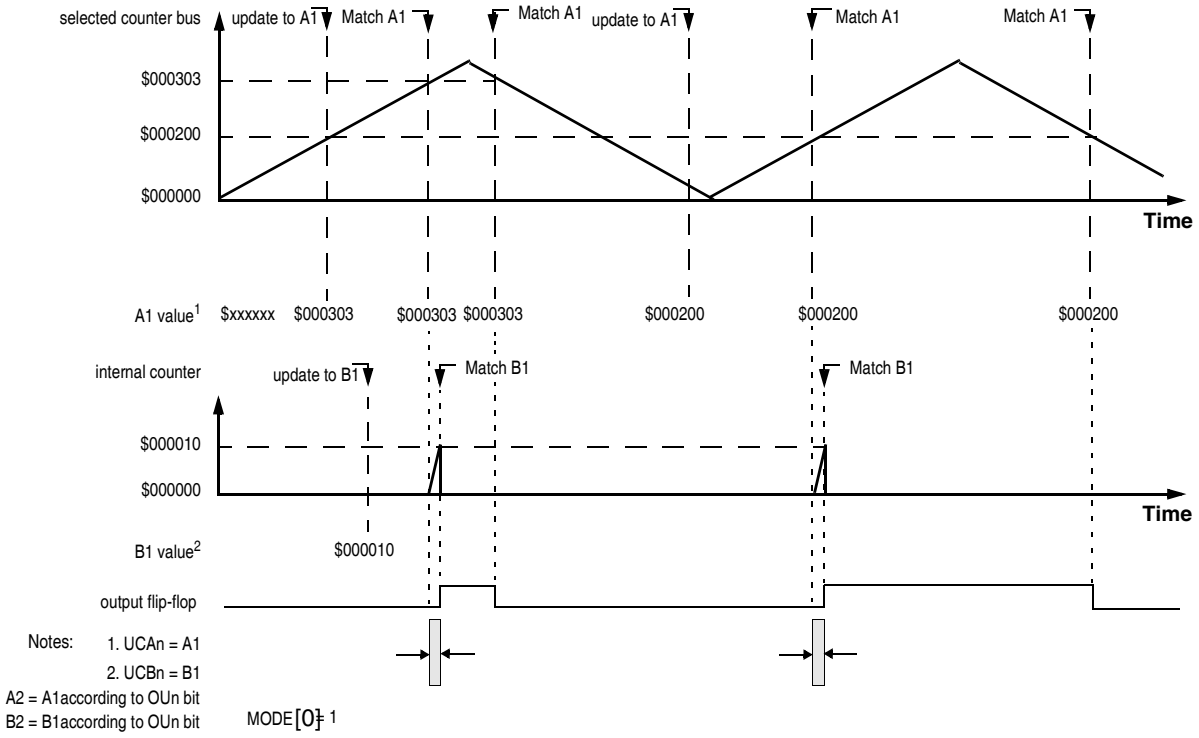


Figure 32-43. Output PWMC with leading dead time insertion

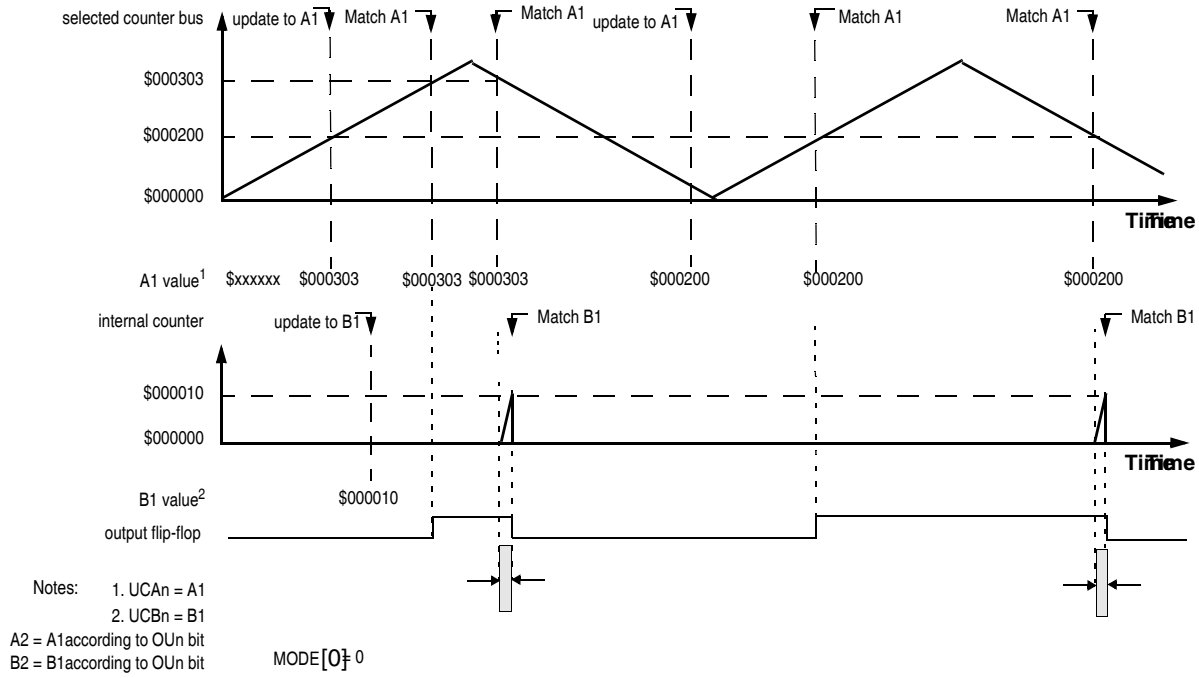


Figure 32-44. Output PWMC with trailing dead time insertion

32.13.1.1.16 Center Aligned Output PWM Buffered with Dead-Time (OPWMCB) Mode

This operation mode generates a center aligned PWM with dead time insertion to the leading or trailing edge. A1 and B1 registers are double buffered to allow smooth output signal generation when changing A2 or B2 registers values on the fly.

The selected counter bus must be running in up/down counter mode, as shown in Figure 32-33. The time base selected for a channel configured to OPWMCB mode should be a channel configured to MCB mode. BSL[1:0] bits select the time base. Register A1 contains the ideal duty cycle for the PWM signal and is compared with the selected time base. Register B1 contains the dead time value and is compared against the internal counter. For a leading edge dead time insertion, the output PWM duty cycle is equal to the difference between register A1 and register B1, and for a trailing edge dead time insertion, the output PWM duty cycle is equal to the sum of register A1 and register B1. Mode[0] bit selects between trailing and leading dead time insertion, respectively.

NOTE

The internal prescaler of the OPWMCB channel should be set to the same value of the MCB channel prescaler. These prescalers should also be synchronized. In this case A1 and B1 registers represents the same timing scale for duty cycle and dead time insertion.

Figure 32-45 describes the load of A1 and B1 registers which occurs when the selected counter bus reaches the value one. This counter value defines the cycle boundary. Note that values written to A2 or B2 within cycle (n) are loaded into A1 or B1 registers, respectively and used to generate matches in cycle (n+1).

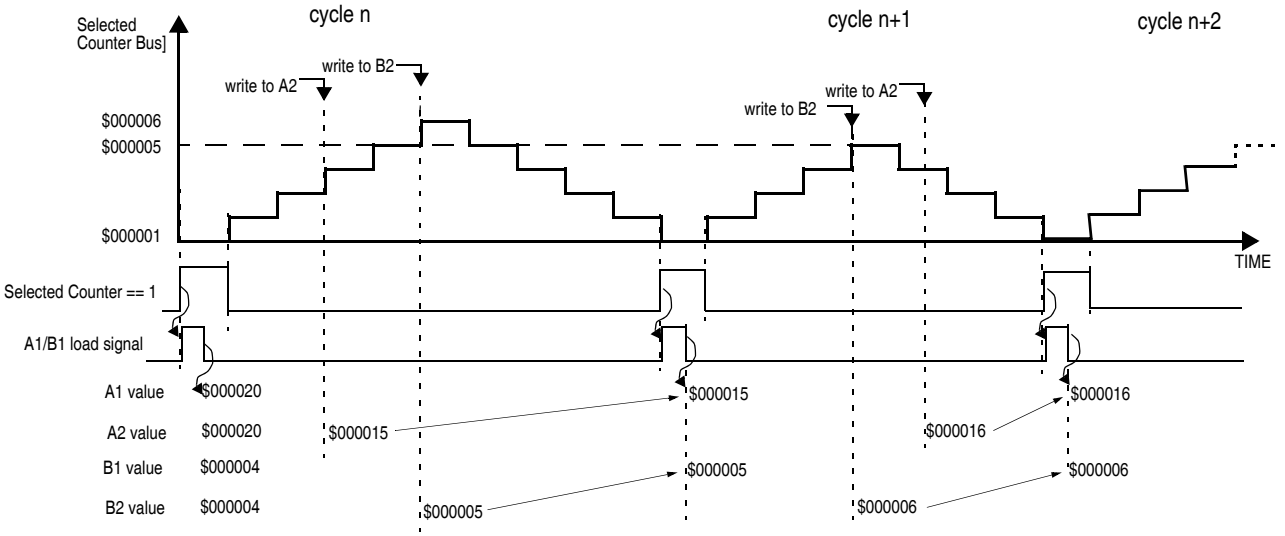


Figure 32-45. OPWMCB A1 and B1 registers load

The OUDISn bit can be used to disable the A1 and B1 updates, thus allowing to synchronize the load on these registers with the load of A1 or B1 registers in others channels. Note that using the update disable bit A1 and B1 registers can be updated at the same counter cycle thus allowing to change both registers at the same time.

In this mode A1 matches always sets the internal counter to 0x000001. When operating with leading edge dead time insertion the first A1 match sets the internal counter to 0x000001. When a match occurs between register B1 and the internal time base, the output flip-flop is set to the value of the EDPOL bit. In the following match between register A1 and the selected time base, the output flip-flop is set to the complement of the EDPOL bit. This sequence repeats continuously. Figure 32-46 shows two cycles of a Center Aligned PWM signal. Note that both A1 and B1 register values are changing within the same cycle which allows to vary at the same time the duty cycle and dead time values.

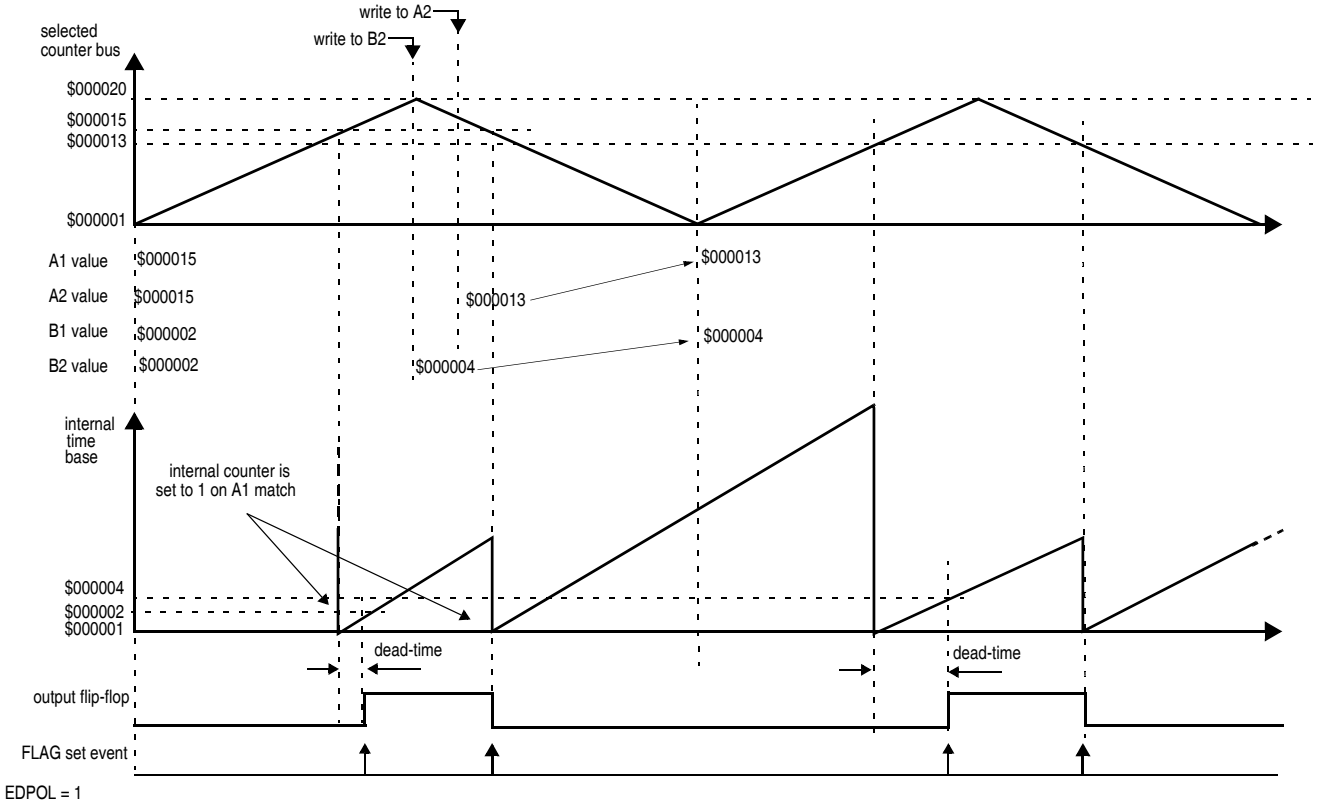


Figure 32-46. Output PWMCB with Lead Dead Time Insertion

When operating with trailing edge dead time insertion, the first match between A1 and the selected time base sets the output flip-flop to the value of the EDPOL bit and sets the internal counter to 0x000001. In the second match between register A1 and the selected time base, the internal counter is set to 0x000001 and B1 matches are enabled. When the match between register B1 and the selected time base occurs the output flip-flop is set to the complement of the EDPOL bit. This sequence repeats continuously.

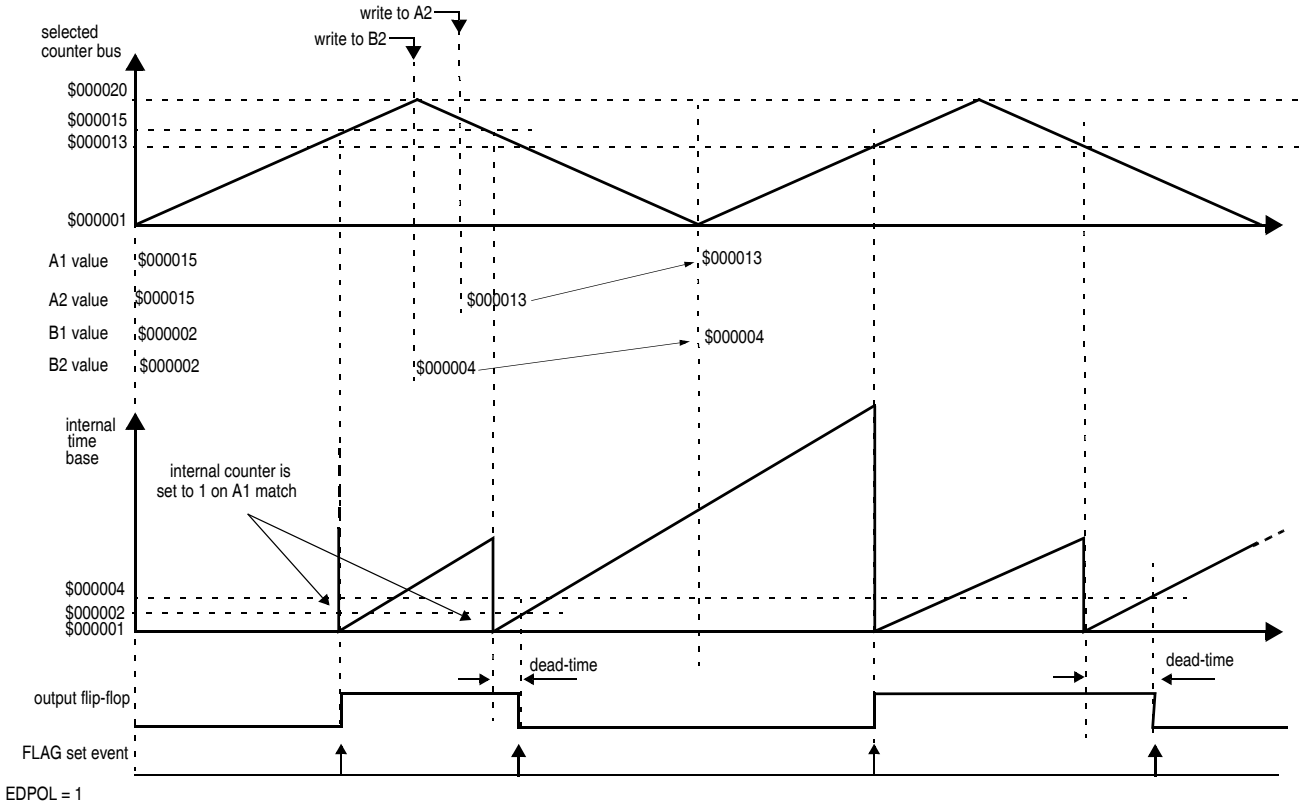


Figure 32-47. Output PWMCB with Trail Dead Time Insertion

FLAG can be generated in the trailing edge of the output PWM signal when MODE[1] is cleared, or in both edges, when MODE[1] is set. If subsequent matches occur on comparators A and B, the PWM pulses continue to be generated, regardless of the state of the FLAG bit.

NOTE

In OPWMCB mode FORCMA and FORCMB do not have the same behavior as a regular match. Instead they force the output flip-flop to constant value which depends upon the selected dead time insertion mode, lead or trail and the value of the EDPOL bit.

FORCMA has different behaviors depending upon the selected dead time insertion mode, lead or trail. In lead dead time insertion FORCMA force a transition in the output flip-flop to the opposite of EDPOL. In trail dead time insertion the output flip-flop is forced to the value of EDPOL bit.

If FORCMB bit is set, the output flip-flop value depends upon the selected dead time insertion mode. In lead dead time insertion FORCMB forces the output flip-flop to transition to EDPOL bit value. In trail dead time insertion the output flip-flop is forced to the opposite of EDPOL bit value.

NOTE

FORCMA bit set does not set the internal time-base to 0x000001 as a regular A1 match.

The FLAG bit is not set either in case of a FORCMA or FORCMB or even if both forces are issued at the same time.

NOTE

FORCMA and FORCMB have the same behavior even in Freeze or normal mode regarding the output pin transition.

When *FORCMA* is issued along with *FORCMB* the output flip-flop is set to the opposite of EDPOL bit value. This is equivalent of saying that.*FORCMA* has precedence over *FORCMB* when lead dead time insertion is selected and *FORCMB* has precedence over *FORCMA* when trail dead time insertion is selected.

Duty cycle from 0% to 100% can be generated by setting appropriate values to A1 and B1 registers relatively to the period of the external time base. Setting A1=1 generates a 100% duty cycle waveform. If A1 > maximum value of the selected counter bus period, then a 0% duty cycle is produced. Assuming EDPOL is set to one and OPWMCB mode with trail dead time insertion, 100% duty cycle signals can be generated if B1 occurs at or after the cycle boundary (external counter = 1).

NOTE

A special case occurs when A1 is set to (external counter bus period)/2, which is the maximum value of the external counter. In this case the output flip-flop is constantly set to the EDPOL bit value.

The internal channel logic prevents matches from one cycle to propagate to the next cycle. In trail dead time insertion B1 match from cycle (n) could eventually cross the cycle boundary and occur in cycle (n+1). In this case B1 match is masked out and does not cause the output flip-flop to transition. Therefore matches in cycle(n+1) are not affected by the late B1 matches from cycle(n).

Figure 32-48 shows a 100% duty cycle output signal generated by setting A1=4 and B1=3. In this case the trailing edge is positioned at the boundary of cycle n+1, which is actually considered to belong to cycle n+2 and therefore does not cause the output flip-flip to transition.

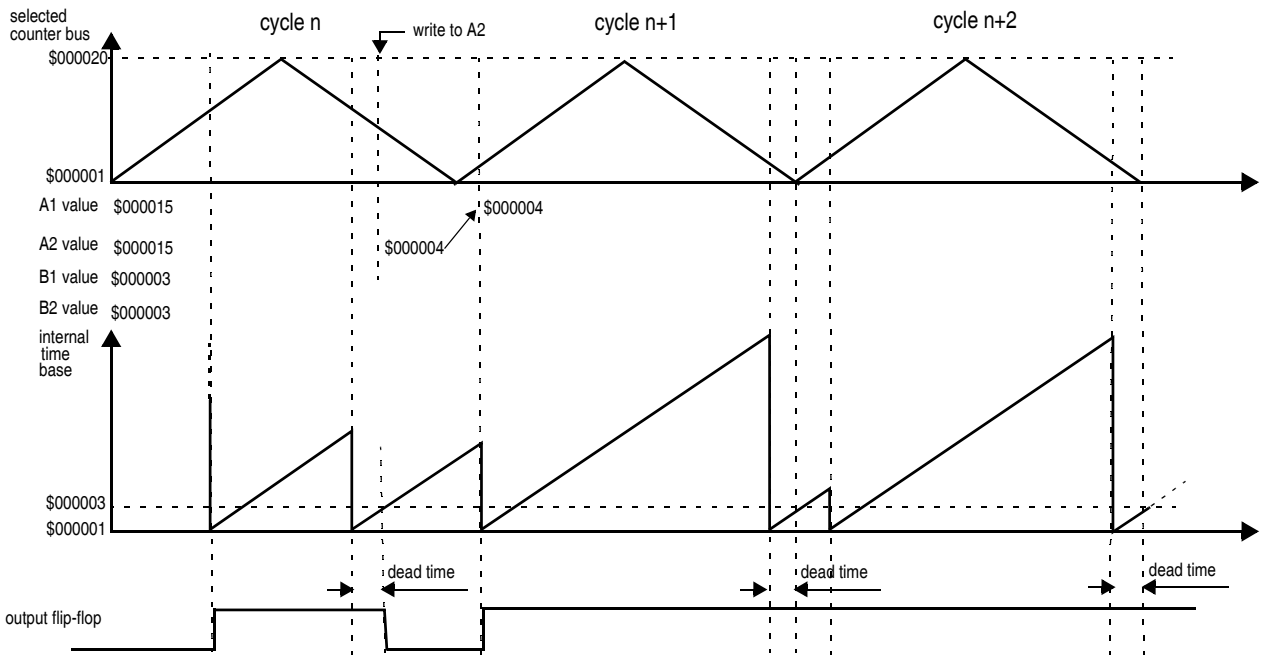


Figure 32-48. OPWMCB with 100% Duty Cycle (A1=4 and B1=3)

The output disable input, if enabled, cause the output flip-flop to transition to EDPOL inverted. This feature allows to force the channel output pin to a safety state from the application stand point. The internal channel matches continue to occur even in this case, thus generating Flags. As soon as the output disable is deasserted the channel output pin is again controlled by A1 and B1 matches. Note that this process is synchronous, meaning that the output channel pin transitions only on system clock edges.

It is important to notice that, like in OPWMB and OPWFMB modes, the match signal used to set or clear the channel output flip-flop is generated on the deassertion of the channel combinational comparator output signal which compares the selected time base with A1 or B1 register values. Please, refer to Figure 32-38 which describes the delay from matches to output flip-flop transition in OPWFMB mode. The operation of OPWMCB mode is similar to OPWFMB regarding matches and output pin transition.

32.13.1.1.17 Output Pulse Width Modulation (OPWM) Mode

Registers A1 and B1 define the leading and trailing edges of the PWM output pulse, respectively. MODE[0] bit controls the transfer from register B2 to B1, which can be done either immediately (MODE[0] cleared), providing the fastest change in the duty cycle, or at every match of register A1 (MODE[0] set).

The value loaded in register A1 is compared with the value on the selected time base. When a match on comparator A occurs, the output flip-flop is set to the value of the EDPOL bit. When a match occurs on comparator B, the output flip-flop is set to the complement of the EDPOL bit.

FLAG can be generated at match B, when MODE[1] is cleared, or in both matches, when MODE[1] is set.

At any time, the FORCMA and FORCMB bits allow the software to force the output flip-flop to the level corresponding to a match on A or B respectively. Note that FLAG bit is not set by the FORCMA and FORCMB operations.

If subsequent matches occur on comparators A and B, the PWM pulses continue to be generated, regardless of the state of the FLAG bit.

In order to achieve 100% duty cycle, both registers A1 and B1 must be set to the same value. When a simultaneous match on comparators A and B occur, the output flip-flop is set at every period to the value of EDPOL bit. 0% duty cycle is possible by writing 0x000000 to register A. When a match occurs, the output flip-flop is set at every period to the complement of EDPOL bit. The transfer from register B2 to B1 is still controlled by MODE[0] bit.

NOTE

If A1 and B1 are set to the 0x000000, a 0% duty cycle waveform is produced.

Figure 32-49 and Figure 32-50 show the Unified Channel running in OPWM with immediate update and next period update, respectivelly.

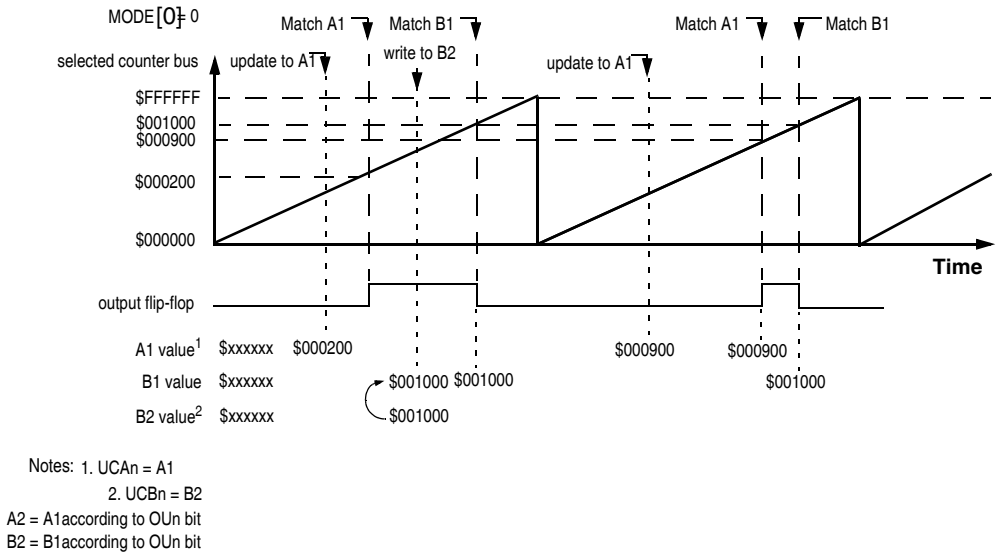


Figure 32-49. Output PWM with immediate update

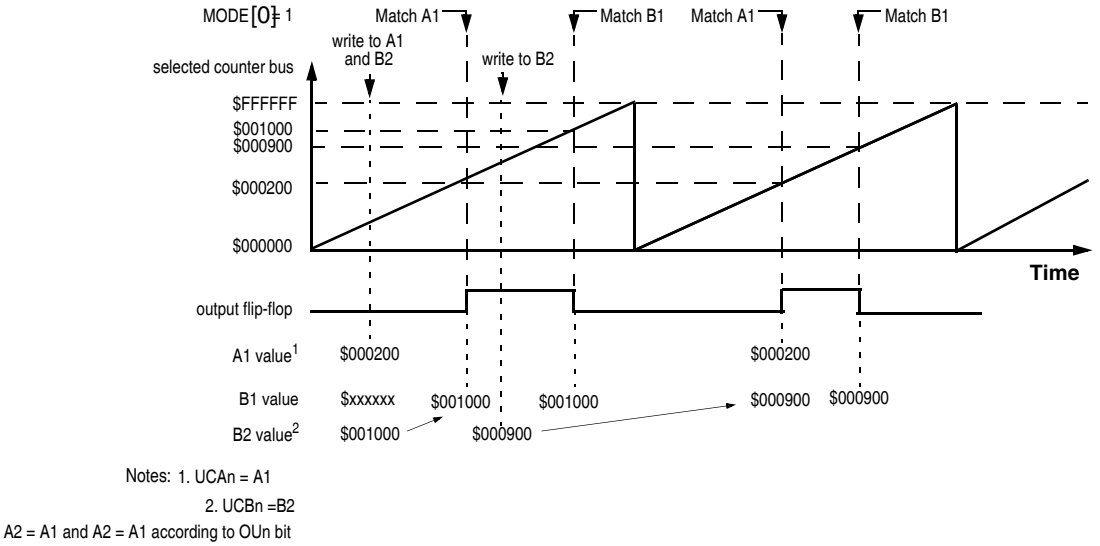


Figure 32-50. Output PWM with next period update

32.13.1.1.18 Pulse Width Modulation Buffered (OPWMB) Mode

OPWMB mode is used to generate pulses with programmable leading and trailing edge placement. An external counter is selected from one of the counter buses. A1 register value defines the first edge and B1 the second edge. The output signal polarity is defined by the EDPOL bit. If EDPOL is zero, a negative edge occurs when A1 matches the selected counter bus; and a positive edge occurs when B1 matches the selected counter bus.

The A1 and B1 registers are double buffered and updated from A2 and B2, respectively, at the cycle boundary. The load operation is similar to the OPWFMB mode. Please, refer to Figure 32-40 for more information about A1 and B1 registers update.

FLAG can be generated at B1 matches, when MODE[1] is cleared, or in both, A1 and B1 matches, when MODE[1] is set. If subsequent matches occur on comparators A and B, the PWM pulses continue to be generated, regardless of the state of the FLAG bit.

FORCMA and FORCMB bits allow the software to force the output flip-flop to the level corresponding to a match on A1 or B1 respectively. FLAG bit is not set by the FORCMA and FORCMB operations.

Following are described some rules applicable to the OPWMB mode:

- B1 matches have precedence over A1 matches if they occur at the same time within the same counter cycle
- A1=0 match from cycle(n) has precedence over B1 match from cycle(n-1)
- A1 matches are masked out if they occur after B1 match within the same cycle
- Any value written to A2 or B2 on cycle(n) is loaded to A1 and B1 registers at the following cycle boundary (assuming OUDISn is not asserted). Thus the new values will be used for A1 and B1 matches in cycle(n+1)

Figure 32-51 describes the operation of the OPWMB mode regarding A1 and B1 matches and the transition of the channel output pin. In this example EDPOL is set to zero.

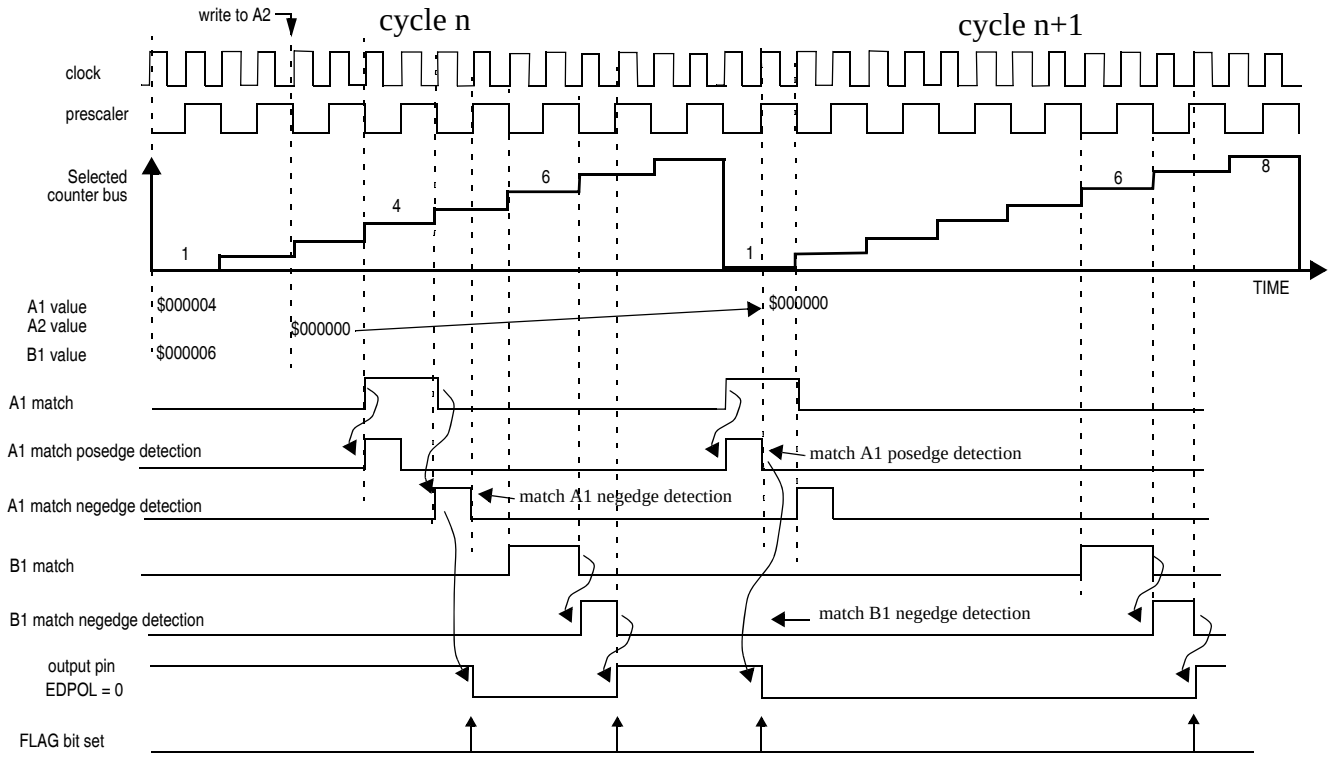


Figure 32-51. OPWMB Mode Matches and Flags

Note that the output pin transitions are based on the negedges of the A1 and B1 match signals. Figure 32-51 shows in cycle(n+1) the value of A1 register being set to zero. In this case the match posedge is used instead of the negedge to transition the output flip-flop.

Figure 32-52 describes the channel operation for 0% duty cycle. Note that the A1 match posedge signal occurs at the same time as the B1=8 negedge signal. In this case A1 match has precedence over B1 match, causing the output pin to remain at EDPOL bit value, thus generating a 0% duty cycle signal.

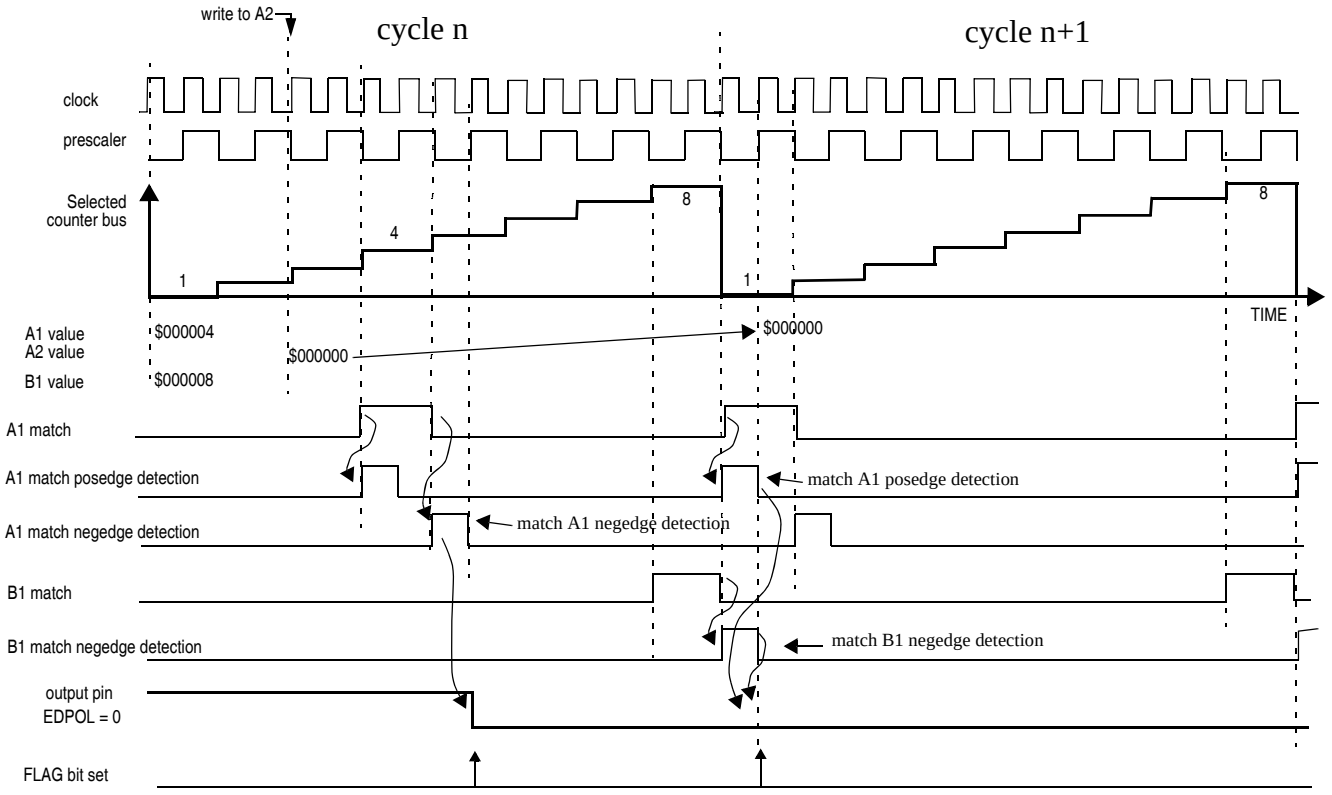


Figure 32-52. OPWMB Mode with 0% Duty Cycle

Figure 32-53 describes the operation of the OPWMB mode with the Output Disable signal being asserted. The output disable forces a transition in the output pin to the EDPOL bit value. After deasserted, the output disable allows the output pin to transition at the following A1 or B1 match. Note that the output disable does not modify the Flag bit behavior. Note that there is one system clock delay between the assertion of the output disable signal and the transition of the output pin to EDPOL.

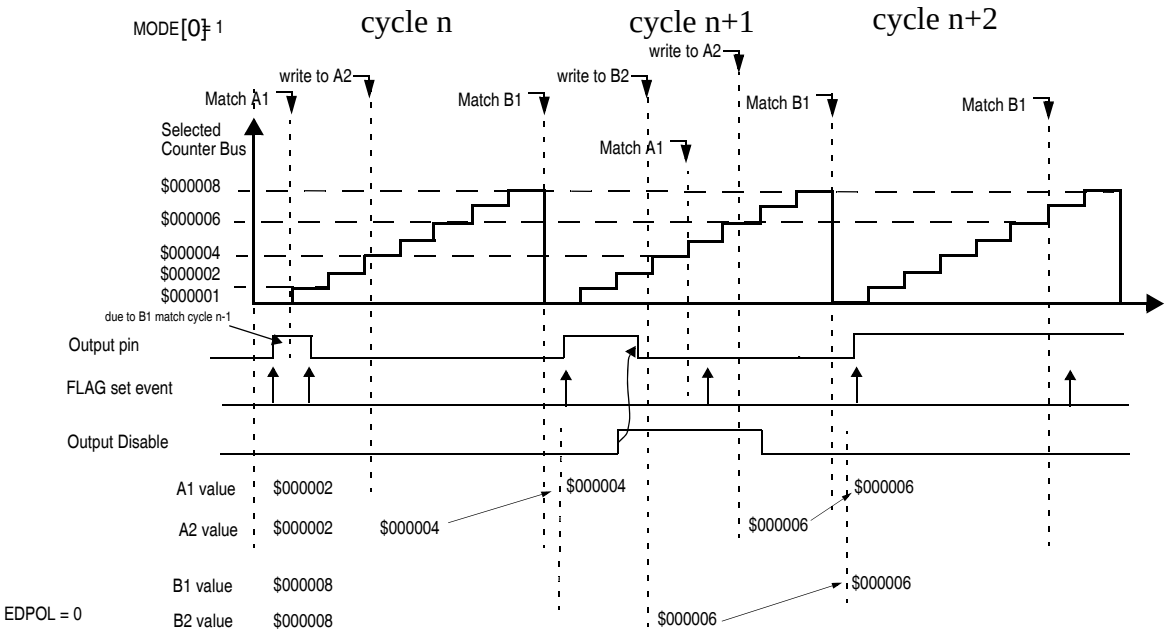


Figure 32-53. OPWMB Mode with Active Output Disable

Figure 32-54 shows a waveform changing from 100% to 0% duty cycle. EDPOL in this case is zero. In this example B1 is programmed to the same value as the period of the external selected time base.

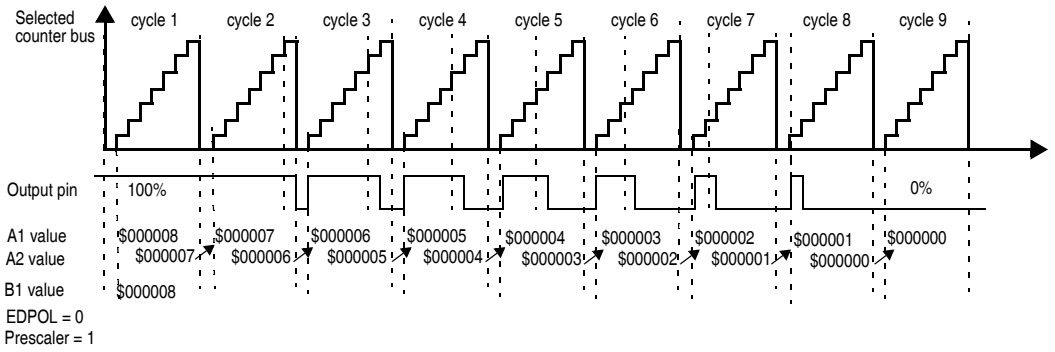


Figure 32-54. OPWMB Mode from 100% to 0% Duty Cycle

In Figure 32-54 if B1 is set to a value lower than 0x000008 it is not possible to achieve 0% duty cycle by only changing A1 register value. Since B1 matches have precedence over A1 matches the output pin transitions to the opposite of EDPOL bit at B1 match. Note also that if B1 is set to 0x000009, for instance, B1 match does not occur, thus a 0% duty cycle signal is generated.

32.13.1.2 Input Programmable Filter (IPF)

The IPF ensures that only valid input pin transitions are received by the Unified Channel edge detector. A block diagram of the IPF is shown in Figure 32-55.

The IPF is a 5-bit programmable up counter that is incremented by the selected clock source, according to bits IF[3:0] in UCCRn register.

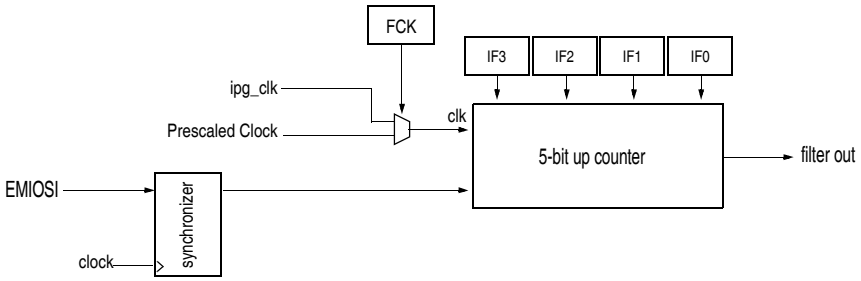


Figure 32-55. Input Programmable Filter submodule diagram

The input signal is synchronized by system clock. When a state change occurs in this signal, the 5-bit counter starts counting up. As long as the new state is stable on the pin, the counter remains incrementing. If a counter overflows occurs, the new pin value is validated. In this case, it is transmitted as a pulse edge to the edge detector. If the opposite edge appears on the pin before validation (overflow), the counter is reset. At the next pin transition, the counter starts counting again. Any pulse that is shorter than a full range of the masked counter is regarded as a glitch and it is not passed on to the edge detector. A timing diagram of the input filter is shown in Figure 32-56.

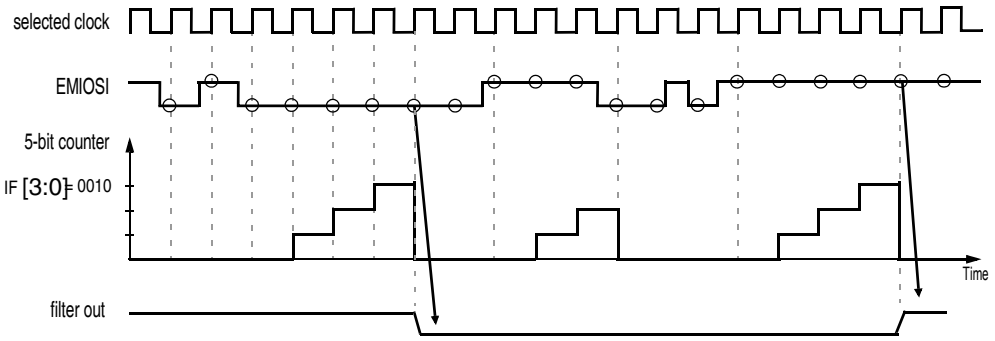


Figure 32-56. Input Programmable filter example

32.13.1.3 Clock Prescaler (CP)

The CP divides the GCP output signal to generate a clock enable for the internal counter of the Unified Channels. It is a programmable 2-bit down counter. The GCP output signal is prescaled by the value defined in Table 32-7 according to the UCPRE[1:0] bits in UCCRn register. The output is clocked every time the counter reaches zero. Counting is enabled by setting the UCPREN bit in the UCCRn. The counter can be stopped at any time by clearing this bit, thereby stopping the internal counter in the Unified Channel.

32.13.1.4 Effect of Freeze on the Unified Channel

When in debug mode, FRZ bit in the MCR register and the FREN bit in the UCCRn are both set, the internal counter and Unified Channel capture and compare functions are halted. The UC is frozen in its current state.

During freeze, all registers are accessible. When the Unified Channel is operating in an output mode, the force match functions remain available, allowing the software to force the output to the desired level.

Note that for input modes, any input events that may occur while the channel is frozen are ignored.

When exiting debug mode or freeze enable bit is cleared (FRZ in the MCR or FREN in the UCCRn register) the channel actions resume.

32.13.2 IP Bus Interface Unit (BIU)

The BIU provides the interface between the Internal interface bus (IIB) and the Peripheral Bus, allowing communication among all submodules and this IP interface.

The BIU allows 8, 16 and 32 bits access. They are performed over a 32-bit data bus in a single cycle clock.

32.13.2.1 Effect of Freeze on the BIU

When the FRZ bit in the MCR register is set and the module is in debug mode, the operation of BIU is not affected.

32.13.3 Global Clock Prescaler Submodule (GCP)

The GCP divides the system clock to generate a clock for the CPs of the Unified Channels. It is a programmable 8-bit up counter. The main clock signal is prescaled by the value defined in Figure 32-7 according to the GPRE[7:0] bits in MCR register. The output is clocked every time the counter overflows. Counting is enabled by setting the GPREN bit in the MCR. The counter can be stopped at any time by clearing this bit, thereby stopping the internal counter in all the Unified Channels.

32.13.3.1 Effect of Freeze on the GCP

When the FRZ bit in the MCR register is set and the module is in debug mode, the operation of GCP submodule is not affected, i.e., there is no freeze function in this submodule.

32.14 Initialization/Application Information

On resetting the eMIOS all of the Unified Channels enter GPIO input mode.

32.14.1 Considerations

Before changing an operating mode, the UC must be programmed to GPIO mode and UCAn and UCBn registers must be updated with the correct values for the next operating mode. Then the UCCRn register can be written with the new operating mode. If a UC is changed from one mode to another without performing this procedure, the first operation cycle of the selected time base can be random, i.e., matches can occur in random time if the contents of UCAn or UCBn were not updated with the correct value before the time base matches the previous contents of UCAn or UCBn.

When interrupts are enabled, the software must clear the FLAG bits before exiting the interrupt service routine.

32.14.2 Application Information

Correlated output signals can be generated by all output operation modes. Bits OUDISn can be used to control the update of these output signals.

In order to guarantee that the internal counters of correlated channels are incremented in the same clock cycle, the internal prescalers must be set up before enabling the global prescaler. If the internal prescalers are set after enabling the global prescaler, the internal counters may increment in the same ratio, but at a different clock cycle.

It is recommended to drive Output Disable Input signals with the mts_flag_out signals of some UCs running in SAIC mode. When an output disable condition happens, the software interrupt routine must service the output channels before servicing the channels running SAIC. This procedure avoid glitches in the output pins.

32.14.2.1 Time Base Generation

For MC, OPWFM and OPWM with internal clock source operation modes, the internal counter rate can be modified by configuring the clock pre scaler ratio. Figure 32-57 shows an example of a time base with pre scaler ratio equal to one. When the pre scaler is greater than one, the counter is immediately cleared on a match and then incremented in the next pre scaled clock edge, except when running in OPWFM mode or MC mode with internal clock source, in which case the counter will skip the next pre scaled clock edge and continue incrementing on subsequent edges, as shown in Figure 32-58.

NOTE

MCB, OPWFMB and OPWMB modes have a different behavior

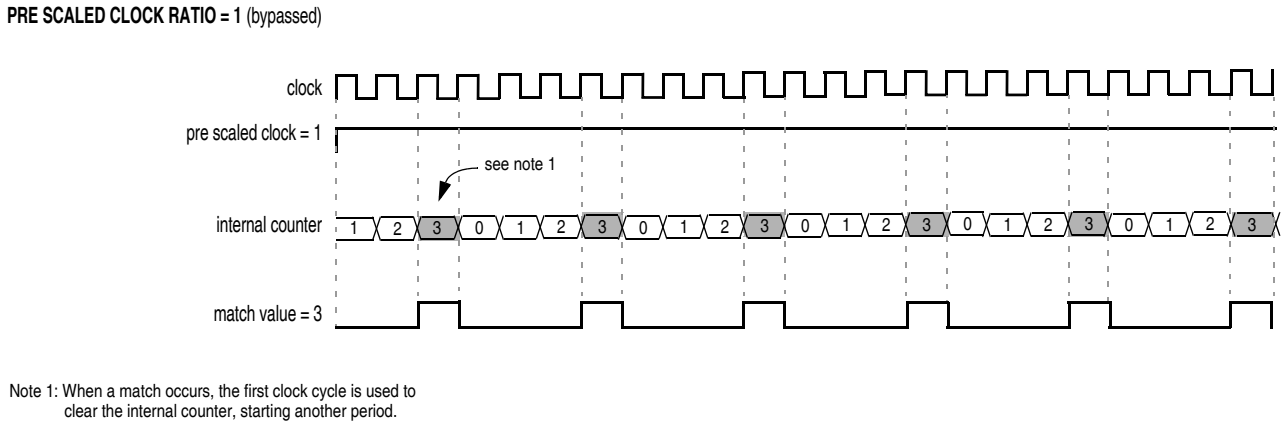
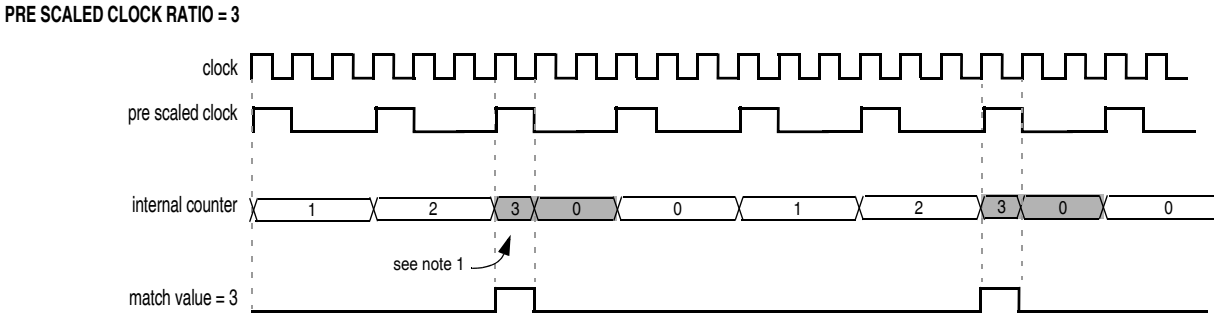


Figure 32-57. Time base period when running in the fastest pre scaler ratio



Note 1: When a match occurs, the first clock cycle is used to clear the internal counter, and only after a second edge of pre scaled clock the counter will start counting.

Figure 32-58. Time base period when running with a pre scaler ratio greater than 1

32.14.2.2 Coherent Accesses

For IPWM, IPM and PEA modes, it is highly recommended that the software waits for a new FLAG set event before start reading UCAn and UCBn registers to get a new measurement. The FLAG indicates that new data has been captured and it is the only way to assure data coherency.

The FLAG set event can be detected by polling the FLAG bit or by enabling the interrupt or DMA request generation.

Reading the UCAn register again in the same period of the last read of UCBn register may lead to incoherent results. This will occur if the last read of UCBn register occurred after a disabled B2 to B1 transfer.

Chapter 33
A/D Converter (ATD)

33.1 Introduction to the ATD on MAC7200

The MAC7200 family of devices has a single Analog to Digital Converter module (ATD_A), offering up to 16 analog input channels. The ATD module is connected to the eDMA controller in order to help access the conversion results and to define the conversions to be performed. Two DMA channels are connected to the ATD: One channel is used to move the conversion result out of the modules result register, and the other channel provides a conversion command word. The conversion command word is used to define parameters such as the mode of conversion, the channel to be converted and the length of the conversion. By defining a number of conversion words in the MCUs system memory, it is possible to build up a predefined sequence of conversions which will be executed without the intervention of the CPU. It is also possible for the CPU to write the conversion command word and to read the conversion results directly on the module without the need to use the DMA controller.

The ATD module includes the ability to trigger a conversion sequence based on either an external signal, or by one of two internal signal lines, SYSTRG0 or SYSTRG1.

In order to use an external trigger source to prompt a conversion, any one of the analog channels for the ATD can be used as the source for this off-chip trigger. The channel to be used must first be defined in the module External Trigger Channel register. This incoming trigger signal will be synchronized to the system clock which will introduce a delay of two system clock cycles before the conversion can be triggered.

The internal trigger signal lines are connected to the devices Programmable Interrupt Timer (PIT) which provides two dedicated programmable 16-bit timers to trigger the Analog to Digital converter. The counter associated with the SYSTRG[x] lines can be programed with the desired conversion trigger duration. This counter will count down from this pre-loaded value to zero at the rate defined by the system frequency. Once at zero the trigger signal will be sent to the ATD and the counter will be reloaded.

The ATD module can be disabled by writing to the MDIS bit in the modules ATDMODE register. Disabling the module will turn off the clock and shut down the analog circuits, although most of the module registers remain available to be accessed by the core across the peripheral bus. The MDIS bit is intended to be used when the module is not required in the application. By default the ATD is disabled after reset (MDIS=1), so prior to use, the MDIS bit must be cleared.

See Section 33.8, “The DMADC1032 Module” for detailed information about the control and operation of the Analog to Digital Converter module.

33.2 ATD Features

- Up to 16 analog input channels.
- 12-bit resolution 9-bit accuracy.

- 2µS minimum conversion time.
- Internal sample and hold circuitry.
- Pre-measurement discharge of internal Sample and Hold circuit possible for all channels.
- Programmable input sample time for various source impedances.
- Queued conversion sequences supported by DMA controller.
- Analog inputs configurable as external sample triggers.
- On-chip timer triggers for sampling.

33.3 ATD Implementation

There are several scenarios for using the ATD on the MAC72xx:

1. A single channel is sampled multiple times, and the results averaged to produce a single reading
2. Multiple channels are continuously sampled (once per channel)
3. The processor switches between combinations of #1 and #2 based on external or internal stimulus.

To support synchronized simultaneous sampling on the 32-channel MAC72xx device, we will have a two internal triggers (**ETRIG**) for the ATD. Each ATD will have the following trigger options:

- No trigger
- Trigger from ATD Channel *xx*
- Trigger from ETRIG0
- Trigger from ETRIG1

NOTE

What we are referring to above is the sample trigger, not the DMA trigger.

33.4 ATD External Pins

Table 33-1. ATD External Pins

Signal	Description
VRH	Connected directly to the VRH pad. Used for external reference voltage (high).
VRL	Connected directly to the VRL pad. Used for external reference voltage (low).
VDDA	Connected directly to the VDDA pad. Used to power the ATD.
VSSA	Connected directly to the VSSA pad. Used to power the ATD.
REFBYPC	Connected directly to the REFBYPC pad. Used to filter noise from the VRH/VRL reference voltage to improve conversion results.
AN00 - AN15	Connected directly to the PE[0:15] pads. Used as ATD channels or external sample triggers. The GPIO functionality on these pads is handled inside the PIM block.

33.5 ATD Bus Aborts

The ATD module supports Peripheral Bus bus aborts, and enforces the following memory map:

Table 33-2. ATD Bus Aborts

Abort	Allowed
	$0000-$0007
$0008-$000b	
	$000c-$0013
	$0014-$0017 (READ only)
$0018-$3fff	

Supervisor Access: Unused.

33.6 ATD Differences from MAC71xx

- ATD runs directly from system (fast) clock (not programmable)
- Fixed MUCts01643: Write on bus abort still writes the register
- Added support for up to 32 channels on a single ATD
  - Changed ETRIGCH field in the ATDETRIGCH register from 4 bits to 5 bits
  - Changed CWCH field in the ATDCW register from 4 bits to 5 bits
  - Changed RRCH field in the ATDRR register from 4 bits to 5 bits
- Increased resolution for the ATD clock prescaler
  - Changed PRES field in the ATDPRE register from 7 bits to 8 bits
- Added Pre-discharge functionality
  - Added the PRED bit in the ATDMODE register
- Changed from 10±1 bit@7µs to 12±3 bit@2µs accuracy
  - Added the ATDCAL register for calibration
  - Added the CWAR bit in the ATDCW register
  - Added the RRAR bit in the ATDRR register
- Added support for 8,10,11 and 12 bit conversions
  - Replaced the CW8 bit with the CWNF[3:2] bits in the ATDCW register
  - Replaced the RR8 bit with the RRNF[3:2] bits in the ATDRR register
- Removed sample amplifier bypass functionality
  - Removed the CWSB bit in the ATDCW register
  - Removed the RRSB bit in the ATDRR register
- Added Warp feature to increase ATD accuracy
  - Added the WARP bit in the ATDMODE register
- Added Command queue not full interrupt
  - Added the CONFIE bit in the ATDINT register
  - Added the CQNF bit in the ATDFLAG register
- Changed interrupt clear in the ATDFLAG register from read register to write ’1’

- DMA request fo r command word is asserted immediately after module is turned on

33.7 ATD Application Usage

33.7.1 Enabling the ATD

Before the ATD can be used, it must be explicitly enabled by clearing the **MDIS** bit.

33.7.2 Setting up the ATD

In order to use the basic functionality of the ATD, it is only necessary to clear the **MDIS** bit. Since all analog inputs on the MAC72xx are routed directly through the PIM to the ATD, it is not necessary to set up the PIM in order to use the ATD to sample channels. In this case, you can use a pin both as an analog input for the ATD and as digital General Purpose Input. These pins can not be used as General Purpose Outputs.

33.7.3 Using external triggers

If you want to use the external triggering capability of the ATD, you should use the following steps:

1. Configure the appropriate pin(s) to be in Peripheral Mode, in the **CONFIG** register(s) in the PIM
2. Clear the **MDIS** bit in the ATD
3. Write the **ATDETRIGCH** register in the ATD to set the desired channel number
4. Write ‘11’ to the **TRIGSEL** bits in the **ATDTRIGCTL** register in the ATD

33.7.4 Using the system triggers

If the ATD should start conversions synchronous to SYSTRIG0 or SYSTRIG1 the following settings must be done in the PIT:

1. PIT: Clear MDIS bit
2. PIT: Specify timer reload value
3. PIT: Enable timer

The timer reload value for step PIT2) can be calculated with the following formula:

$$\frac{\text{TriggerPeriod}}{\text{ClockPeriod}} - 1 = \text{TimerReloadValue} \qquad \textit{Eqn. 33-1}$$

33.7.4.1 Example — Using System Triggers

The system clock is66.67Mhz ($T_{SYSCLOCK}$ = 15ns), which gives us a bus clock of 40Mhz ($T_{BUSCLOCK}$ = 30ns). The desired trigger period is 30µs.

$$\frac{\text{TriggerPeriod}}{\text{ClockPeriod}} - 1 = \frac{30\text{us}}{30\text{ns}} - 1 = 999 = 0\text{x}000003\text{e}7 \qquad \textit{Eqn. 33-2}$$

NOTE

All PIT timers, except the RTI, run off of the Peripheral Bus Clock, which is 1/2 the frequency of the System Clock.

If SYSTRIG0 should trigger a conversion, write the reload value to the timer load value register for timer 10 and enable this timer.

If SYSTRIG1 should trigger a conversion, write the reload value to the timer load value register for timer 9 and enable this timer.

In the ATD you must perform the following steps in order to use a system trigger (it is assumed that all other registers in the ATD are already configured):

1. ATD: Clear MDIS bit
2. ATD: Write $06 to the ATDTRIGCTL register. This selects rising edge sensitivity for SYSTRIG0.

or

3. ATD: Write $0C to the ATDTRIGCTL register. This selects rising edge sensitivity for SYSTRIG1.
4. ATD: Write a command word to the ATDCW register that has the bits CWCM set to ’10’ (wait for trigger)

After the write of the command word to the ATDCW register is performed, the ATD will start the conversion when the trigger is asserted.

NOTE

Be aware that a new command word must be provided before the next trigger is asserted. Otherwise the ETO-flag in the ATDFLAG register will be set.

33.8 The DMADC1032 Module

33.8.1 Overview

The DMADC1032 is a 32-channel multiplexed input analog-to-digital converter (ATD) with a resolution of 12 bits. Please refer to *Device Electrical Specifications* for ATD electrical parameters.

The ATD module can be configured to enable the conversion results to be moved from the result register to system memory without the intervention of the core by using a DMA channel. ATD conversions are controlled using a command word which determines aspects of the conversion such as the channel to be converted or any trigger source used to start the conversion. A DMA channel can also be used to provide a sequence of conversion commands enabling the device to queue up conversion sequences.

Conversions can be started either under the direct control of the ATD module, or by a trigger source. The trigger source can be external to the device, using one of the analog channel pins as its input. It can also be one of the two device-internal programmable timers.

33.8.2 Block Diagram

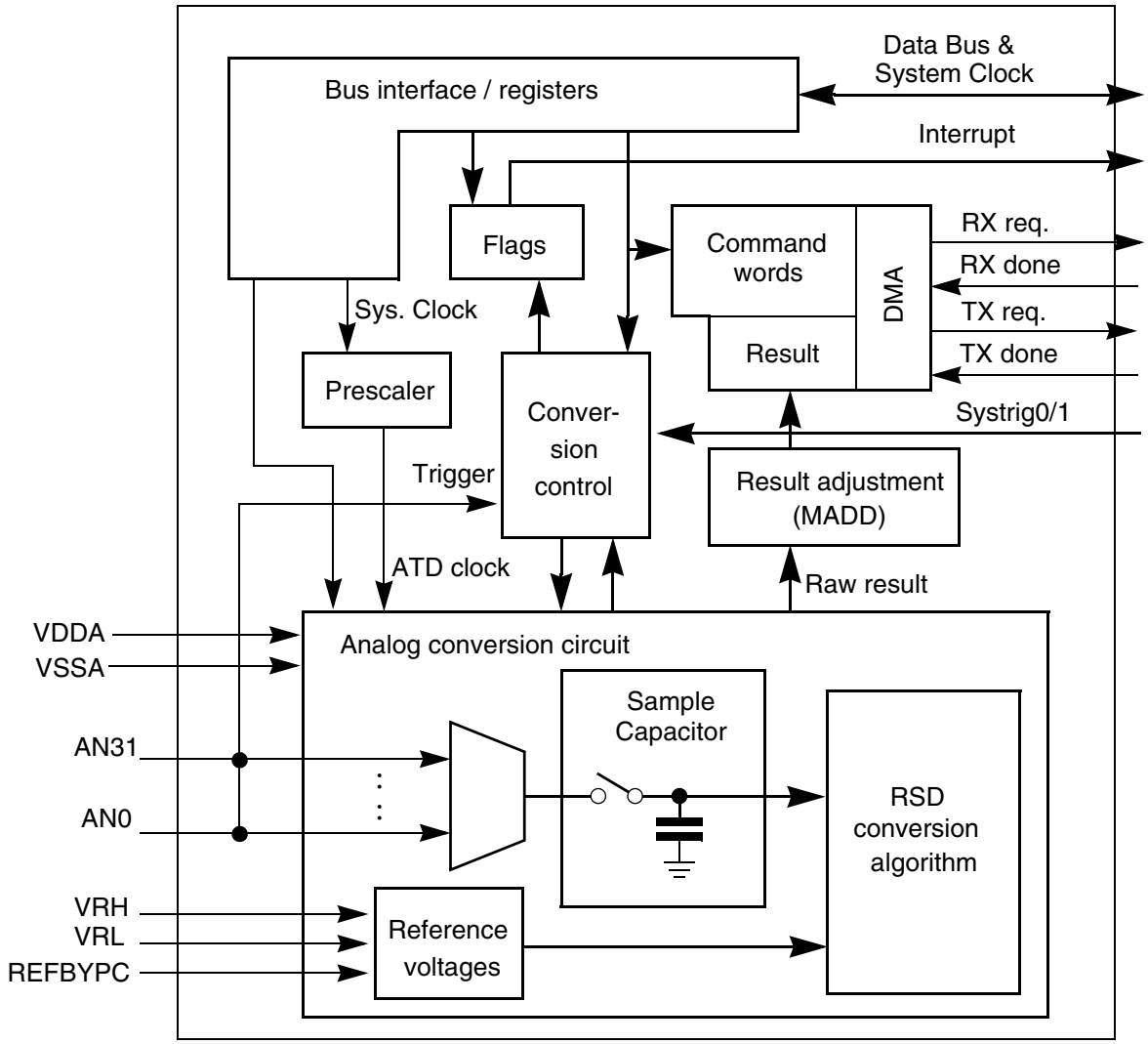


Figure 33-1. DMADC1032 Block Diagram

33.8.3 Features of DMADC1032

- DMA interface
- Command words for conversion are stored in on-chip or external memory
- 12-bit ATD with 9-bit accuracy @10.67MHz ATD clock
- ≤ 2 µsec Single Conversion Time
- Detection of a broken sensor pin (Predischarge feature)
- Programmable Sample Time
- Left/Right Justified, Signed/Unsigned 8/10/11/12 Bit Result Data
- Conversion Completion Interrupt Generation
- Analog Input Multiplexer for 32 Analog Input Channels

- Programmable channel sampling order
- Several sampling modes supported
  - — Continuous Conversion Mode
  - — Convert and Pause
  - — Convert on Trigger
- Flexible Trigger Control
  - — Configurable external trigger functionality on any ATD channel
  - — 2 selectable on-chip triggers

33.8.4 Modes of Operation

The ATD has different modes of operation:

- Normal mode
- Doze mode for medium power saving
- Debug mode for error tracking (no power saving included)
- Disabled mode for maximum power saving

For further details about the ATD behavior in the various operation modes please Section 33.11.4, “Low Power / Operating Modes”.

33.9 Signal Description

33.9.1 Overview

All I/O signals connected to the external SoC pins are summarized in Table 33-3 and described in more detail in the next sub-sections.

Table 33-3. DMADC1032 Signals

Signal Name	Direction	Description	Reset state
VDDA	Input	Secondary power supply	—
VSSA	Input	Ground for secondary power supply	—
VRH	Input	High reference voltage for ATD conversions	—
VRL	Input	Low reference voltage for ATD conversions	—
VREFC	Input	External buffer capacitor for reference voltage	—
AN00	Input	Channel 0: single-ended analog input	—
AN01	Input	Channel 1: single-ended analog input	—
AN02	Input	Channel 2: single-ended analog input	—
AN03	Input	Channel 3: single-ended analog input	—
AN04	Input	Channel 4: single-ended analog input	—
AN05	Input	Channel 5: single-ended analog input	—
AN06	Input	Channel 6: single-ended analog input	—

Table 33-3. DMADC1032 Signals

Signal Name	Direction	Description	Reset state
AN07	Input	Channel 7: single-ended analog input	—
AN08	Input	Channel 8: single-ended analog input	—
AN09	Input	Channel 9: single-ended analog input	—
AN10	Input	Channel 10: single-ended analog input	—
AN11	Input	Channel 11: single-ended analog input	—
AN12	Input	Channel 12: single-ended analog input	—
AN13	Input	Channel 13: single-ended analog input	—
AN14	Input	Channel 14: single-ended analog input	—
AN15	Input	Channel 15: single-ended analog input	—
AN16	Input	Channel 16: single-ended analog input	—
AN17	Input	Channel 17: single-ended analog input	—
AN18	Input	Channel 18: single-ended analog input	—
AN19	Input	Channel 19: single-ended analog input	—
AN20	Input	Channel 20: single-ended analog input	—
AN21	Input	Channel 21: single-ended analog input	—
AN22	Input	Channel 22: single-ended analog input	—
AN23	Input	Channel 23: single-ended analog input	—
AN24	Input	Channel 24: single-ended analog input	—
AN25	Input	Channel 25: single-ended analog input	—
AN26	Input	Channel 26: single-ended analog input	—
AN27	Input	Channel 27: single-ended analog input	—
AN28	Input	Channel 28: single-ended analog input	—
AN29	Input	Channel 29: single-ended analog input	—
AN30	Input	Channel 30: single-ended analog input	—
AN31	Input	Channel 31: single-ended analog input	—

33.9.2 Signal Descriptions

In Figure 33-1 a block diagram of the ATD is shown. The signals on the left side of the diagram are SoC external signals. The signals on the right side are SoC internal signals.

33.9.2.1 VDDA

Power supply for analog conversion circuit

33.9.2.2 VSSA

Ground for VDDA

33.9.2.3 VRH

High reference voltage for ATD conversions.

33.9.2.4 VRL

Low reference voltage for ATD conversions.

33.9.2.5 REFBYPC

Positive terminal for external bypass capacitor for reference voltage VREF.

33.9.2.6 AN*n*

These pins serve as the analog input channels. All pins are single ended inputs only. AN0 .. AN31 can also be configured as an external trigger source for the ATD conversion. Please refer to Section Chapter 4, “Signal Description” for availability of the pins.

33.9.2.7 SYSTRIG*n*

Chip-internal triggers which can start conversions

33.9.2.8 RX req / RX done

DMA request and acknowledge signal for command word receiving

33.9.2.9 TX reg / TX done

DMA request and acknowledge signal for result transmitting

33.9.2.10 SYSTRIG0/1, Interrupt, IP Bus Clock, System Clock

These signals are chip internal signals only. They are shown to give a better overview of the ATD structure.

33.10 Memory Map and Register Definition

33.10.1 Module Memory Map

This section describes the register memory map of the ATD, including the function of each register. Table 33-4 gives an overview of all available DMADC1032 registers.

Table 33-4. Module Memory Map

Address	Use	Size	Access	Mode[1]
Base + 0x00	ATD Trigger Control Register (ATDTRIGCTL)	8	R/W	A
Base + 0x01	ATD External Trigger Channel Register (ATDETRIGCH)	8	R/W	A

Table 33-4. Module Memory Map (Continued)

Base + 0x02	ATD Prescaler Register (ATDPRE)	8	R/W	A
Base + 0x03	ATD Operating Modes Register (ATDMODE)	8	R/W	A
Base + 0x04	ATD Calibration Register (ATDCAL)	8/16/32	R/W	A
Base + 0x08	ATD Predischarge Time Select Register (ATDPTS)	8	R/W	A
Base + 0x09 .. 0x0D	Reserved	—	—	—
Base + 0x0E	ATD Interrupt Register (ATDINT)	8	R/W	A
Base + 0x0F	ATD Flag Register (ATDFLAG)	8	R/W	D/A
Base + 0x10	ATD Command Word Register (ATDCW)	16/32	R/W	A
Base + 0x14	ATD Result Register (ATDRR)	8/16/32	R	A

1. U = User Mode, S = Supervisor Mode, D = Debug Mode, A = All (No restrictions), V = DFV Mode

NOTE

Register Address = Base Address + Address Offset, where the Base Address is specified in Chapter 9, “Device Memory Map”.

33.10.2 Register Descriptions

This section describes (in address order) all of the DMADC1032 registers and their individual bits. A detailed register description starts with a figure that contains all bits of the register.

It consists of:

- **Register address:** Describes the location of the register in the memory map
- **Bit:** The exact bit number for the bit or field within the register
- **R:** The value returned for the bit when a read operation on the register is executed
- **W:** Describes whether or not the bit is writable
- **Reset:** The value of the bit after reset

Those bits that are shaded out are read-only, meaning that writes to that bit will have no effect. If you want to maintain maximum future compatibility, all unused bits should be masked out during any read/write operations.

33.10.2.1 ATD Trigger Control Register (ATDTRIGCTL)

This register defines how the ATD uses both external and on-chip triggers. On those devices where the on-chip triggers are available, this register can be used to configure which trigger (if any) is used to start a conversion.

Address Base + 0x00 Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	TRIGSEL		TRIGP	TRIGLE
W								
Reset	0	0	0	0	0	0	0	0

Figure 33-2. ATD Trigger Control Register (ATDTRIGCTL)

Table 33-5. ATDTRIGCTL Field Descriptions

Field	Description
7–4	Reserved, should be cleared.
3–2 TRIGSEL	Trigger source select. Determines the trigger source. An external trigger will always be synchronized internally to the system clock, resulting in a delay of 4 system clock cycles between the trigger and a conversion start. When using SYSTRIG0 or SYSTRIG1, there is always a delay of 2 system clock cycles between the trigger and a conversion start. 00 No external trigger used. 01 Use SYSTRIG0 (on-chip trigger) as trigger. 10 Use SYSTRIG1 (on-chip trigger) as trigger. 11 Use analog input channel (see ETRIGCH) as trigger. Note: Avoid sampling a channel that is currently set as trigger pin. Otherwise, there will be an increased current consumption on that pin.
1 TRIGP	Trigger polarity. 0 Trigger is low level/falling edge sensitive. 1 Trigger is high level/rising edge sensitive.
0 TRIGLE	Trigger is level/edge sensitive. Determines whether the trigger is edge or level sensitive. When the conversion mode for the current command word is set to “wait for trigger”, the conversion starts only if the right trigger is asserted. 0 Trigger is edge sensitive 1 Trigger is level sensitive

Table 33-6 shows the trigger sensitivity according to the TRIGP and TRIGLE bit.

Table 33-6. Trigger Sensitivity Selection Table

TRIGP	TRIGLE	Trigger Sensitivity
0	0	falling edge sensitive
0	1	low level sensitive
1	0	rising edge sensitive
1	1	high level sensitive

NOTE

Writing to this register will stop the current conversion and discard the command word in ATDCW. A DMA-request for fetching a command word will be asserted. The DMA-request for saving a result will remain as it is. A new conversion will start after a command word has been written to the command register.

33.10.2.2 ATD External Trigger Channel Register (ATDETRIGCH)

This register defines which of the 16 analog input channels will be used for the external trigger. The external triggering must be enabled in ATDTRIGCTL.

Address Base + 0x01 Access: User read/write

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

Figure 33-3. ATD External Trigger Channel Register (ATDETRIGCH)

Table 33-7. ATDETRIGCH Field Descriptions

Field	Description
7–5	Reserved, should be cleared.
4–0 ETRIGCH	External trigger channel. Defines which analog input channel will be the input trigger source. 0b00000 Channel 0 0b00001 Channel 1 0b00010 Channel 2 0b00011 Channel 3 . . . . . . 0b11110 Channel 30 0b11111 Channel 31 Note: An external trigger will always be synchronized internally to the system clock, resulting in a delay of 4 system clock cycles between the trigger and a conversion start.

NOTE

Writing to this register will stop the current conversion and discard the command word in ATDCW. A DMA-request for fetching a command word will be asserted. The DMA-request for saving a result will remain as it is. A new conversion will start after a command word has been written to the command register.

NOTE

Changing the value of this register can cause an unintended trigger which will set the ETO flag. (e.g. when ‘rising edge sensitivity’ is selected), the previous selected channel was low and the new selected channel is high this will be seen as a rising edge.

33.10.2.3 ATD Prescaler Register (ATDPRE)

This register defines the clock divider for the clock used by the RSD-unit. As this unit requires a clock between 0.5 MHz and 12 MHz (ATD clock), the system clock must be divided so that it falls within this range. The clock divider can be set between 2 and 256.

Address Base + 0x02 Access: User read/write

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

Figure 33-4. ATD Prescaler Register (ATDPRE)

Table 33-8. ATDPRE Field Descriptions

Field	Description
7–0 PRES	Defines the division ratio of the system clock and therefore the rate of the ATD clock. PRES can range from 0 to 255 offering a division of the system clock from divide by 2 to divide by 256.

The following formula shows how the ATD clock is derived from the system clock:

$$f_{ATDClock} = \frac{f_{SystemClock}}{(PRES + 1)} \qquad \textit{Eqn. 33-3}$$

Example 33-1.

The system clock is assumed to be 48 MHz. As the ATD clock must be 12 MHz and below the clock prescaler is calculated to be 48.0 MHz/12.0 MHz = 4 resulting to 12 Mhz for ATD clock. The binary value for a prescaler of 4 according to Table 33-9 is 0b0000_0011.

Example 33-2.

The system clock is assumed to be 50 MHz. As the ATD clock must be 12 MHz and below the clock prescaler is calculated to be 50 MHz/12 MHz = 4.17. In this case the next possible number for a prescaler is 5 resulting to 10.0 MHz for ATD clock. The binary value for a prescaler of 5 is 0b0000_0100.

Table 33-9 lists the various clock divider values.

Table 33-9. Selection Table for System Clock Divider

PRES [7:0]	sys clk divide ratio	PRES [7:0]	sys clk divide ratio
0b0000_0000	not programmable	0b1111_1010	251
0b0000_0001	2	0b1111_1011	252
0b0000_0010	3	0b1111_1100	253
0b0000_0011	4	0b1111_1101	254
0b0000_0100	5	0b1111_1110	255
...	...	0b1111_1111	256

NOTE

A divide ratio of 1 (prescaler = 0b00000000) can not be set. When writing 0x00 to ATDPRE the register will automatically be set to 0x01 (divide ratio = 2).

NOTE

Executing conversions with a clock frequency above 12 MHz or below 0.5 MHz will lead to unreliable conversion results.

NOTE

Writing to this register will stop the current conversion and discard the command word in ATDCW. A DMA-request for fetching a command word will be asserted. The DMA-request for saving a result will remain as it is. A new conversion will start after a command word has been written to the command register.

33.10.2.4 ATD Operating Modes Register (ATDMODE)

This register provides control bits for the ATD operating modes. For details about the meaning of the operating modes see Section 33.11.4, “Low Power / Operating Modes.

Address Base + 0x03 Access: User read/write

	7	6	5	4	3	2	1	0
R	MDIS	DOZE	DEBUG		PRED	WARP	0	0
W								
Reset	1	0	0	0	0	1	0	0

Figure 33-5. ATD Operating Modes Register (ATDMODE)

Table 33-10. ATDMODE Field Descriptions

Field	Description
7 MDIS	Module disable. Causes all ATD clocks to halt and the analog conversion circuit to shut down. This mode offers maximum power saving. For temporarily disabling the ATD, MDIS can be used but Doze mode is recommended. 0 Module enabled 1 Module disabled Note: Enabling and disabling of the ATD via the MDIS bit is intended to be used primarily during the startup of the device. Care should be taken to ensure that the ATD is in an idle state, with no conversions pending, before setting or clearing MDIS.
6 DOZE	Doze mode support. Causes the analog conversion circuit to shut down but leaves the registers accessible. The mode is intended for power saving. 0 Do not support Doze mode 1 Support Doze mode

Table 33-10. ATDMODE Field Descriptions (Continued)

Field	Description
5–4 DEBUG	Debug mode support. Allows the user to enable Debug features. This is useful for debugging real-time systems, where continuation of the conversion process after the MCU has been halted could result in abnormal system behavior. The ATD can react in different ways when Debug mode is entered: 00 Continue conversion (Debug mode disabled for the ATD). 01 Reserved for future use. 10 Finish current conversion, then freeze. 11 Freeze immediately.
3 PRED	Pre-discharge. This bit allows to pre-discharge the sample capacitor before the actual conversion starts. The voltage after pre-discharging is V_{RL}. Enabling this feature will require a higher conversion time. Section 33.11.6, “Conversion Timing and See 33.11.7 Conversions Using Predischarge for details. 0 Pre-discharge to V_{RL} is disabled 1 Pre-discharge to V_{RL} is enabled
2 WARP	Warp transfer curve. This bit allows to slightly warp the ATD transfer curve. This will change the raw result value of the conversions. This bit can be used to increase the ATD accuracy (See 33.12.2 ATD Calibration / Result Adjustment for details). 0 Warp disabled 1 Warp enabled
1–0	Reserved, should be cleared.

NOTE

Writing to this register will stop the current conversion and discard the command word in ATDCW. A DMA-request for fetching a command word will be asserted. The DMA-request for saving a result will remain as it is. A new conversion will start after a command word has been written to the command register.

33.10.2.5 ATD Calibration Register (ATDCAL)

This register contains the calibration constants used to adjust the ATD conversion results to reduce the total unadjusted error. Section 33.12.2, “ATD Calibration / Result Adjustment,” for details about the calibration scheme.

Address Base + 0x4 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	GCC														
W																
Reset	0	1	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	OCC													
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 33-6. 32ATD Calibration Register (ATDCAL)

Table 33-11. ATDCAL Field Descriptions

Field	Descriptions
31	Reserved, should be cleared.
30–16 GCC	Gain Calibration Constant. This register contains the Gain Calibration Constant used to adjust the ATD conversion results. The constant is calculated from the Gain value: • GCC = Gain x 16384. The GCC range is 0 <= 32767 corresponding to the Gain range 0 <= gain < 2. Example: Gain is assumed to be 1.024. In this case GCC = 1.024 x 16384 = 16777.2. The value for GCC must be truncated to 16777. In binary format, this is 0b100_0001_1000_1001.
15–14	Reserved, should be cleared.
13–0 OCC	Offset Calibration Constant. This register contains the Offset Calibration Constant used to adjust conversion results. It is calculated from the Offset value: • Offset >= 0: OCC = Offset x 4 • Offset < 0: OCC = Offset x 4 + 16384 The OCC range is 0 <= OCC <= 8191 corresponding to Offset range -2048 <= Offset <= 2047.75 in steps of 0.25. Example (positive Offset): Offset is assumed to be 3.8. In this case OCC = 3.8 x 4 = 15.2. The value for OCC must be truncated to 15. In binary format, this is 0b00_0000_0000_1111. Example (negative Offset): Offset is assumed to be -3.8. In this case OCC = -3.8 x 4 + 16384 = -15.2 +16384 = 16368.8. The value for OCC must be truncated to 16368. In binary format, this is 0b11_1111_1111_0000. The following table gives some examples for OCC values vs. Offset value:

OCC[13:0]	Offset (decimal)
0b00_0000_0000_0000	0
0b00_0000_0000_0001	0.25
0b00_0000_0000_0010	0.5
0b01_1111_1111_1111	2047.75
0b10_0000_0000_0000	-2048
0b10_0000_0000_0001	-2047.75
0b11_1111_1111_1111	-0.25

NOTE

Write accesses to this register are not recommended while a conversion is running as the conversion result might get corrupted.

33.10.2.6 ATD Predischarge Time Select Register (ATDPTS)

This register defines the duration of the predischarge phase. During the predischarge phase the sample capacitor and the parasitric capacitiy of the current analog channel will be discharged. The predischarge

time is used to detect a broken sensor connection. For further details about the predischarge feature please see Section 33.11.7, “Conversions Using Predischarge”.

Address Base + 0x08 Access: User read/write

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

Figure 33-7. ATD Predischarge Time Select Register (ATDPTS)

Table 33-12. ATDPTS Field Descriptions

Field	Description
7–4	Reserved, should be cleared.
3–0 TIME	Defines the time during which the sample capacitor will be discharged. During this time also the parasitic capacities of the current ATD channel will be discharged. The time can be selected between 2 ATD clock cycles and 16 ATD clock cycles (TIME = 0x01..0x0F). TIME = 0x00 also means 2 ATD clock cycles discharge time. The ATD channel which will also be predischarged is defined by the CWCH bits of the current command word. The following scenarios are possible: • When the current command word selects any channel and the feature ‘special channel’ is not set then the channel selected by the CWCH bits will be discharged. • When the current command word selects one of the ATD reference voltage channels only the sample capacitor is discharged. The predischarge feature can be enabled via the ATDMODE.PRED bit. Please see Section 33.10.2.4, “ATD Operating Modes Register (ATDMODE)”. If the feature is disabled the bit values in this register are ignored. For further details about the Predischarge feature please refer to Section 33.11.7, “Conversions Using Predischarge”.

33.10.2.7 ATD Interrupt Register (ATDINT)

This register contains the interrupt enable bits corresponding to the flags in the ATDFLAG register. If a flag is set and the interrupt enable bit for this flag is also set, an interrupt will be asserted to the system interrupt controller.

Address Base + 0x0E Access: User read/write

	7	6	5	4	3	2	1	0
R	0	CQNFIE	CQFIE	CQEIE	CRLIE	ETOIE	CPIE	CCIE
W								
Reset	0	1	0	0	0	0	0	0

Figure 33-8. ATD Interrupt Register (ATDINT)

Table 33-13. ATDINT Field Descriptions

Field	Description
7	Reserved, should be cleared.
6 CQNFIE	Command queue not full interrupt enable. Enables/disables the interrupt for CQNF (command queue not full). 0 CQNF interrupt is disabled. If CQNF is ‘1’ no interrupt will be generated 1 CQNF interrupt is enabled. If CQNF is ‘1’ an interrupt will be generated
5 CQFIE	Command queue full interrupt enable. Enables/disables the interrupt for CQF (command queue full). 0 CQF interrupt is disabled. If CQF is ‘1’ no interrupt will be generated 1 CQF interrupt is enabled. If CQF is ‘1’ an interrupt will be generated
4 CQEIE	Command queue empty interrupt enable. Enables/disables the interrupt for CQE (command queue empty). 0 CQE interrupt is disabled. If CQE is ‘1’ no interrupt will be generated 1 CQE interrupt is enabled. If CQE is ‘1’ an interrupt will be generated
3 CRLIE	Conversion result lost interrupt enable. Enables/disables the interrupt for CRL (conversion result lost). 0 CRL interrupt is disabled. If CRL is ‘1’ no interrupt will be generated 1 CRL interrupt is enabled. If CRL is ‘1’ an interrupt will be generated
2 ETOIE	External trigger overrun interrupt enable. Enables/disables the interrupt for ETO (external trigger overrun). 0 ETO interrupt is disabled. If ETO is ‘1’ no interrupt will be generated 1 ETO interrupt is enabled. If ETO is ‘1’ an interrupt will be generated
1 CPIE	Conversion paused interrupt enable. Enables/disables the interrupt for CP (conversion paused). 0 CP interrupt is disabled. If CP is ‘1’ no interrupt will be generated 1 CP interrupt is enabled. If CP is ‘1’ an interrupt will be generated
0 CCIE	Conversion complete interrupt enable. Enables/disables the interrupt for CC (conversion complete). 0 CC interrupt is disabled. If CC is ‘1’ no interrupt will be generated 1 CC interrupt is enabled. If CC is ‘1’ an interrupt will be generated

33.10.2.8 ATD Flag Register (ATDFLAG)

This register contains the interrupt flags for the ATD module. When an event occurs (e.g. a conversion finished), the appropriate interrupt flag bits is set. If the interrupt enable bit corresponding to the flag is also set, an interrupt will be asserted to the system interrupt controller. See 33.10.2.7 ATD Interrupt Register (ATDINT) for the interrupt enable bits.

A flag can be cleared by writing a ‘1’ to it. Writing a ‘0’ has no effect. During a low-power mode, the flags can not be cleared. For further details about low-power modes please see Section 33.11.4, “Low Power / Operating Modes”.

Address Base + 0x0F — Access: User read/write

	7	6	5	4	3	2	1	0
R	0	CQNF	CQF	CQE	CRL	ETO	CP	CC
W								
Reset	0	1	0	0	0	0	0	0

Figure 33-9. ATD Flag Register (ATDFLAG)

Table 33-14. ATDFLAG Field Descriptions

Field	Description
7	Reserved, should be cleared.
6 CQNF	Command queue not full flag. Indicates that the queue for command words is not full and can therefore store another command word. The conversion, currently being executed, if any, is not affected by a write. 0 Command queue is full 1 Command queue is not full
5 CQF	Command queue full flag. Indicates that the queue for command words is full. Note that when CQF is set, any write access to the command word register (ATDCW) will overwrite the command that was in the command register before. The conversion currently executed is not affected by a write. 0 Command queue is not full 1 Command queue is full
4 CQE	Command queue empty flag. Indicates that the queue for command words is empty. More command words can be stored. It also indicates that no conversion is currently running. 0 Command queue not empty 1 Command queue is empty
3 CRL	Conversion result lost flag. Indicates that a conversion result was overwritten before it could be read. 0 Conversion results have always been stored/read in time 1 A conversion result was overwritten Note: CRL is useful to detect possible problems with the availability of the DMA channels. Therefore it is useful to enable this interrupt (CRLIE = 1) during the software debugging phase.
2 ETO	External trigger overrun flag. Will indicate that additional edges at the trigger input have been recognized but no triggers were expected as the ATD is currently executing a conversion. If level-sensitive triggering is set, ETO will never be set. 0 No additional edges have been detected 1 One or more additional edges have been detected. If ETOIE is ‘1’ an interrupt will be generated. Note: Additional edges can not be detected while module is in low-power mode or disabled.
1 CP	Conversion paused flag. Indicates that a conversion with CWCM = 01 (convert then pause) has finished. When such a conversion finishes no new command word is executed until the MCU writes a new command word to the ATDCW register. 0 No conversion finished with CWCM = “convert then pause” 1 A conversion with CWCM = “convert then pause” finished
0 CC	Conversion complete flag. Indicates the end of a conversion. When a conversion finishes, this flag will be set (and remain set) if the CWGI bit of the command word is also set. 0 No conversion with CWGI = 1 finished 1 A conversion with CWGI = 1 finished

The CQNF, CQF and CQE flags are dynamic, which means that their values can change without clearing them as they represent current states. The flags CRL, ETO, CP and CC remain set after they got set once. They must be cleared via software.

NOTE

The flags can not be cleared during a low-power mode. Fur further details about low-power modes please see Section 33.11.4, “Low Power / Operating Modes”.

33.10.2.9 ATD Command Word Register (ATDCW)

This register contains the command word for the next conversion to be executed after the current one.

Each write access to this register will be regarded as a new command word. Therefore it is not recommended to replace a 16-bit write access to ATDCW[31:16] with two 8-bit write accesses to ATDCW[31:24] and ATDCW[23:16].

As long as a command word has not become the current command word it will be overwritten by additional writes to the ATDCW register (See 33.11.5.1 Command Queue).

NOTE

Writing to this register during the Doze mode is not recommended. The write access will change the register value but it will not be regarded as a new command word. Therefore it will not start a new conversion after waking up from Doze mode.

Address Base + 0x10 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	CWAR	CWSC	CWGI	CWCM		CWSL			CWNF				CWCH			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	C	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 33-10. ATD Command Word Register (ATDCW)

Table 33-15. ATDCW Field Descriptions

Field	Descriptions
31 CWAR	Command word adjust result. To reduce the total unadjusted error the conversion result can be adjusted using the gain calibration constant and the offset calibration constant in the ATDCAL register. 0 Use raw conversion result 1 Adjust the conversion result
30 CWSC	Command word special channel selected. Selects between special channels and single-ended conversions. Special channels are internal reference voltages. 0 Sample channel defined by CWCH 1 Sample special channels See Table 33-18. Channel Selection Table for details about the channel selection.
29 CWGI	Command word generate interrupt. Defines whether the CC flag (ATDFLAG) should be set after a conversion finishes. 0 Do not set CC flag 1 Set CC flag
28–27 CWCM	Command word conversion mode. Defines the way channels are sampled. There are four different ways, which are summarized in table Table 33-16. A more detailed description of the four different mode is given later in this section.

Table 33-15. ATDCW Field Descriptions (Continued)

<table>
<tr><th>Field</th><th>Descriptions</th></tr>
<tr><td>26–25
CWSL</td><td>Command word sample length. Selects the length of the sample time measured in ATD clock cycles.
00 Sample time of 2 ATD clock periods
01 Sample time of 8 ATD clock periods
10 Sample time of 64 ATD clock periods
11 Sample time of 128 ATD clock periods</td></tr>
<tr><td>24–21
CWNF</td><td>Command word numeric format. Determines the numeric format in which the conversion result is stored. The following tables provide settings for the various available formats.
<table>
<tr><th>CWNF[1:0]</th><th>Conversion Result Numeric Format (Sign/Alignment)</th></tr>
<tr><td>00</td><td>Right justified and unsigned</td></tr>
<tr><td>01</td><td>Right justified and signed</td></tr>
<tr><td>10</td><td>Left justified and unsigned</td></tr>
<tr><td>11</td><td>Left justified and signed</td></tr>
</table>
<table>
<tr><th>CWNF[3:2]</th><th>Conversion Result Numeric Format (Resolution)</th></tr>
<tr><td>00</td><td>8 Bit</td></tr>
<tr><td>01</td><td>10 Bit</td></tr>
<tr><td>10</td><td>11 Bit</td></tr>
<tr><td>11</td><td>12 Bit</td></tr>
</table></td></tr>
<tr><td>20–16
CWCH</td><td>Command word channel. Determines the input channel that will be sampled. Table 33-18 gives a detailed overview how to configure CWCH for the desired channel.</td></tr>
<tr><td>15–0</td><td>Reserved, should be cleared.</td></tr>
</table>

Table 33-16. Conversion Mode Selection Table

CWCM[1:0]	Conversion mode
00	Conversion reset
01	Convert then pause
10	Wait for trigger
11	Convert continuously

- Conversion reset:

  This conversion mode is intended to abort a running conversion. The conversion is aborted immediately after writing the command word. All bits in the command word beside the CWCM bits are ignored.

  If the aborted conversion was either “convert continuously” or “wait for trigger” the ATD will assert a DMA request for a new command. If the aborted conversion was “convert then pause” the

ATD will not assert a DMA request for a new command. The DMA request for saving the conversion result remains unaffected in both cases.

To start a new conversion, the MCU must write a new command word to the command register (see Section 33.12.3.3, “Example 3: Interrupted Conversion Sequence”).

- Convert then pause:

  If a command word is currently being executed with CWCM = “convert then pause” the ATD will stop converting after the conversion finished and will set the ATDFLAG.CP bit. No DMA request for fetching a command word will be asserted. Only a DMA request for saving the result in the memory will be asserted.

  To start a new conversion the MCU must write a new command word (not “conversion reset”) to the command register (Section 33.12.3.2, “Example 2: A Simple Conversion Sequence (Convert then Pause)”).

- Wait for trigger:

  If a command word is currently being executed with CWCM = “wait for trigger”, the ATD will start converting only after a valid trigger has been asserted. For setting the right trigger source/behavior refer to Section 33.10.2.1, “ATD Trigger Control Register (ATDTRIGCTL) and Section 33.10.2.2, “ATD External Trigger Channel Register (ATDETRIGCH).

  See Section 33.12.3.4, “Example 4: Edge-triggered Conversion” and Section 33.12.3.5, “Example 5: Level-triggered Conversion” for examples about triggered conversions.

- Convert continuously:

  If a command word is currently being executed where CWCM = “convert continuously”, the ATD will start a new conversion after the current conversion has finished and a new command word is already available in ATDCW.

  If there is no new command word available, the ATD will try to receive one by keeping a DMA request asserted. After a new command word has been received from the DMA or the MCU, the ATD will start a new conversion according to the new command word (see Section 33.12.3.6, “Example 6: Queue Running Idle”).

The following figure shows how right-justified unsigned conversion will look in the RRCR. In the table, ‘Bit[x]’ reflect the bits determined by the successive approximation.

RRCR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
12 Bit	0	0	0	0	Bit11 MSB	Bit10	Bit9	Bit8	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0 LSB
11 Bit	0	0	0	0	0	Bit11 MSB	Bit10	Bit9	Bit8	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1 LSB
10 Bit	0	0	0	0	0	0	Bit11 MSB	Bit10	Bit9	Bit8	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2 LSB
8 Bit	0	0	0	0	0	0	0	0	Bit11 MSB	Bit10	Bit9	Bit8	Bit7	Bit6	Bit5	Bit4 LSB

Figure 33-11. RRCR for Right Justified Unsigned

The following figure shows how an right-justified signed conversion will look in the RRCR. When using “signed” arithmetic, the MSB of the conversion result must be copied to the bits that are not influenced by

a conversion (e.g. 15 to 8 for 8-Bit). The MSB always shows the inverted value compared to the unsigned numeric format.

RRCR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
12 Bit	$\overline{\text{Bit11}}$ MSB	$\overline{\text{Bit11}}$ MSB	$\overline{\text{Bit11}}$ MSB	$\overline{\text{Bit11}}$ MSB	$\overline{\text{Bit11}}$ MSB	Bit10	Bit9	Bit8	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0 LSB
11 Bit	$\overline{\text{Bit11}}$ MSB	$\overline{\text{Bit11}}$ MSB	$\overline{\text{Bit11}}$ MSB	$\overline{\text{Bit11}}$ MSB	$\overline{\text{Bit11}}$ MSB	$\overline{\text{Bit11}}$ MSB	Bit10	Bit9	Bit8	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1 LSB
10 Bit	$\overline{\text{Bit11}}$ MSB	$\overline{\text{Bit11}}$ MSB	$\overline{\text{Bit11}}$ MSB	$\overline{\text{Bit11}}$ MSB	$\overline{\text{Bit11}}$ MSB	$\overline{\text{Bit11}}$ MSB	$\overline{\text{Bit11}}$ MSB	Bit10	Bit9	Bit8	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2 LSB
8 Bit	$\overline{\text{Bit11}}$ MSB	$\overline{\text{Bit11}}$ MSB	$\overline{\text{Bit11}}$ MSB	$\overline{\text{Bit11}}$ MSB	$\overline{\text{Bit11}}$ MSB	$\overline{\text{Bit11}}$ MSB	$\overline{\text{Bit11}}$ MSB	$\overline{\text{Bit11}}$ MSB	$\overline{\text{Bit11}}$ MSB	Bit10	Bit9	Bit8	Bit7	Bit6	Bit5	Bit4 LSB

Figure 33-12. RRCR for Right Justified Signed

The following figure shows how left-justified unsigned conversion will look in the RRCR.

RRCR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
12 Bit	Bit11 MSB	Bit10	Bit9	Bit8	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0 LSB	0	0	0	0
11 Bit	Bit11 MSB	Bit10	Bit9	Bit8	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1 LSB	0	0	0	0	0
10 Bit	Bit11 MSB	Bit10	Bit9	Bit8	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2 LSB	0	0	0	0	0	0
8 Bit	Bit11 MSB	Bit10	Bit9	Bit8	Bit7	Bit6	Bit5	Bit4 LSB	0	0	0	0	0	0	0	0

Figure 33-13. RRCR for Left Justified Unsigned

The following figure shows how left-justified signed conversion will look in the RRCR. The MSB always shows the inverted value compared to the unsigned numeric format.

RRCR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
12 Bit	$\overline{\text{Bit11}}$ MSB	Bit10	Bit9	Bit8	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0 LSB	0	0	0	0
11 Bit	$\overline{\text{Bit11}}$ MSB	Bit10	Bit9	Bit8	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1 LSB	0	0	0	0	0
10 Bit	$\overline{\text{Bit11}}$ MSB	Bit10	Bit9	Bit8	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2 LSB	0	0	0	0	0	0
8 Bit	$\overline{\text{Bit11}}$ MSB	Bit10	Bit9	Bit8	Bit7	Bit6	Bit5	Bit4 LSB	0	0	0	0	0	0	0	0

Figure 33-14. RRCR for Left Justified Signed 8/10 Bit

The other numeric values will be build accordingly.

The following table should explain in more detail how conversion results will look by giving examples with bit values. The conversion resolution has been chosen as 8-bit:

- Voltage #1: Vmax = +5.120 V, Vmin = 0.000 V; 1 LSB = 20 mV
- Voltage #2: Vmax = +2.560 V, Vmin = –2.560 V; 1 LSB = 20 mV

Please note that the voltage ranges used for the examples shown in Table 33-17 have been selected in a way to produce understandable results. The maximum voltage values (Vmax) may be outside of the valid range. For further details about the (conversion) characteristics of the ATD, please refer to Section A.16, “Analog-to-Digital Converter Characteristics”.

The “regular” printed bits represent the bits used to fill an 8-bit value to a 16-bit value. The bits printed in “**bold**” represent the bits determined by the analog-to-digital conversion.

Table 33-17. Numeric Examples for Result Values

Voltage #1	Voltage #2	Right justified unsigned 8 Bit	Right justified signed 8 Bit	Left justified unsigned 8 Bit	Left justified signed 8 Bit
5.090..5.120	2.530..2.560	0b00000000'**11111111**	0b00000000'**01111111**	0b**11111111**'00000000	0b**01111111**'00000000
5.070..5.090	2.510..2.530	0b00000000'**11111110**	0b00000000'**01111110**	0b**11111110**'00000000	0b**01111110**'00000000
5.050..5.070	2.490..2.510	0b00000000'**11111101**	0b00000000'**01111101**	0b**11111101**'00000000	0b**01111101**'00000000
5.030..5.050	2.470..2.490	0b00000000'**11111100**	0b00000000'**01111100**	0b**11111100**'00000000	0b**01111100**'00000000
:	:	:	:	:	:
2.590..2.610	0.030..0.050	0b00000000'**10000010**	0b00000000'**00000010**	0b**10000010**'00000000	0b**00000010**'00000000
2.570..2.590	0.010..0.030	0b00000000'**10000001**	0b00000000'**00000001**	0b**10000001**'00000000	0b**00000001**'00000000
2.550..2.570	-0.010..0.010	0b00000000'**10000000**	0b00000000'**00000000**	0b**10000000**'00000000	0b**00000000**'00000000
2.530..2.550	-0.030..-0.010	0b00000000'**01111111**	0b11111111'**11111111**	0b**01111111**'00000000	0b**11111111**'00000000
2.510..2.530	-0.050..-0.030	0b00000000'**01111110**	0b11111111'**11111110**	0b**01111110**'00000000	0b**11111110**'00000000
2.490..2.510	-0.070..-0.050	0b00000000'**01111101**	0b11111111'**11111101**	0b**01111101**'00000000	0b**11111101**'00000000
:	:	:	:	:	:
0.050..0.070	-2.510..-2.490	0b00000000'**00000011**	0b11111111'**10000011**	0b**00000011**'00000000	0b**10000011**'00000000
0.030..0.050	-2.530..-2.510	0b00000000'**00000010**	0b11111111'**10000010**	0b**00000010**'00000000	0b**10000010**'00000000
0.010..0.030	-2.550..-2.530	0b00000000'**00000001**	0b11111111'**10000001**	0b**00000001**'00000000	0b**10000001**'00000000
0.000..0.010	-2.560..-2.550	0b00000000'**00000000**	0b11111111'**10000000**	0b**00000000**'00000000	0b**10000000**'00000000

Table 33-18. Channel Selection Table

CWSC	CWCH[4:0]	Sampled voltage
0	0b0_0000 .. 0b1_1111	ATD channel defined by CHWCH[4:0]
1	0b0_0000 .. 0b0_0111	Reserved
1	0b0_1000	ATD reference voltage V_{RH}
1	0b0_1001	ATD reference voltage V_{RL}
1	0b0_1010	ATD reference voltage 50% x (V_{RH} - V_{RL})
1	0b0_1011	ATD reference voltage 75% x (V_{RH} - V_{RL})
1	0b0_1100	ATD reference voltage 25% x (V_{RH} - V_{RL})
1	0b0_1101 .. 0b1_1111	Reserved

NOTE

When converting any of the five ATD reference voltages the predischarge feature must be enabled to reduce the influence of parasitic charges. The predischarge time can be two ATD clock cycles or higher.

33.10.2.10 ATD Result Register (ATDRR)

This register contains the result for the last conversion that has been executed. As the result should never be modified by anyone but the ATD conversion state machine, all bits of this register are read-only.

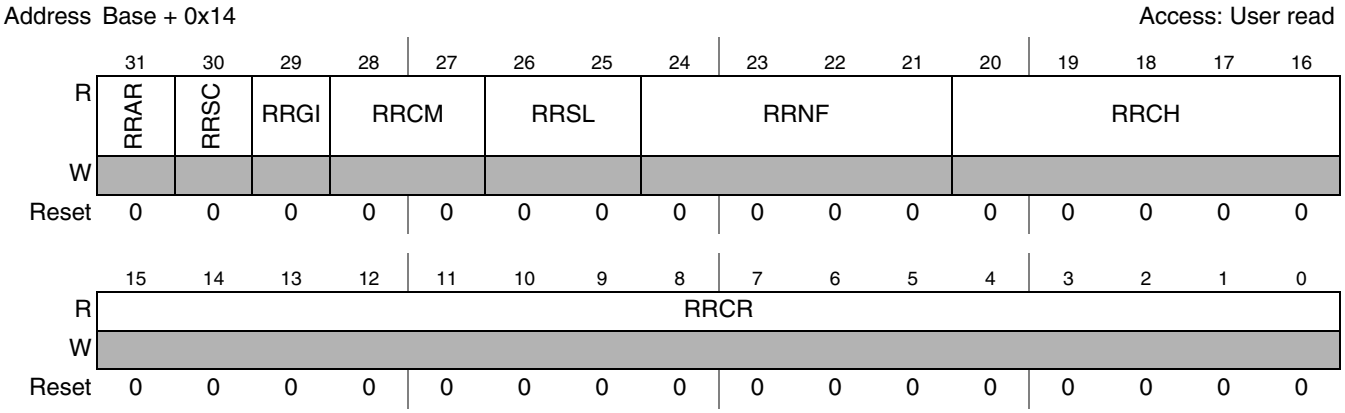


Figure 33-15. ATD Result Register (ATDRR)

After a conversion, bits 31 to 16 of the finished command word will be copied to bits 31 to 16 of the result register without any changes. This allows a better allocation of the executed command word and its conversion result. Each of the ATDRR[31:16] bits is prefixed with RR, rather than the CW prefix used in the command word. For example, the RRCH bits have the same function as the CWCH bits. Refer to Section 33.10.2.9, “ATD Command Word Register (ATDCW) for the function of each bit.

Bits 15 to 0 of the result register (RRCR) will contain the actual conversion result in the numeric format specified by CWNF. For detailed information about the numeric formats see Figure 33-11 to Figure 33-14 and Table 33-17.

After a conversion, the ATD will assert a DMA request to store the result in memory. Once the result was read by the MCU the DMA request will be deasserted.

If a result has not been read before a new conversion result arrives, a result loss will occur: The new conversion result will overwrite the previous one and will set the CRL (conversion result loss) bit in the flag register (ATDFLAG). Despite that, the ATD will continue executing conversions.

NOTE

If this register contains a conversion result any read access to this register will mark the current result as read. If not all bytes of this register are read before a new conversion result arrives no conversion result loss will be flagged.

To avoid data corruption the software must read all relevant bytes of this register with one read access.

33.11 Functional Description

33.11.1 General

The ATD performs analog to digital conversions using a redundant signed digit (RSD) architecture. The ATD is divided into an analog and digital sub-blocks (see Figure 33-1):

The analog sub-block is responsible for multiplexing the various input voltage sources to a sample capacitor. The voltage across this capacitor is compared with a reference voltage. This will deliver the raw result to the digital block.

The digital block partly controls the analog part. E.g. it decides when to start conversions, how to pre-discharge the sample capacitor, how long to sample etc. It also handles read/write accesses and interrupts.

33.11.2 Analog Conversion Circuit

The analog conversion circuit contains all analog components required to perform a conversion according to the RSD-algorithm. A separate power supply VDDA/VSSA allows noise isolation from other MCU circuitry.

This block contains all sub-blocks mentioned in Section 33.11.2.1, “Analog Input Multiplexer” to Section 33.11.2.4, “Conversion Circuit”.

NOTE

When the ATD is turned on for the first time, the analog conversion circuit requires a recovery time t_{REC} until reliable conversion results can be determined[1]. Any conversion started/ended during that time should be ignored.

33.11.2.1 Analog Input Multiplexer

The analog input multiplexer connects one of the 32 analog input channels to the sample capacitor.

33.11.2.2 Sample Capacitor

During the sampling phase the sample capacitor is connected to the analog input channel that is currently converted.

33.11.2.3 Reference Voltage Generation

This block generates all reference voltages needed to determine the raw conversion result.

33.11.2.4 Conversion Circuit

The conversion circuit compares the sampled voltage with the reference voltages.

1. Refer to Appendix A, “Electrical Characteristics” for recovery time t_{REC}

33.11.3 Digital Sub-blocks

This subsection describes the functions implemented in the digital part. See the register descriptions for all details.

This block contains all sub-blocks mentioned in Section 33.11.3.1, “Mode / Timing Control” to Section 33.11.3.8, “Conversion Control”.

33.11.3.1 Mode / Timing Control

The Timing Control is responsible for handling conversions by deciding which channel is to be converted, when the conversion should start etc. It is responsible for the command/result transfer and the interrupt generation.

33.11.3.2 Clock Prescaler

The frequency at which the analog conversion circuit can work is between 0.5 MHz and 12 MHz. A higher system clock frequency must therefore be divided down to fit within this range. This division is provided by the Clock Prescaler.

33.11.3.3 Bus Interface / Registers

The Bus interface handles read/write accesses from the MCU to the registers and DMA transfers of conversion results or commands.

33.11.3.4 Command Word / Result / DMA

This block contains the command word queue which consists of ATDCW, ATDCC and ATDRR. It also takes care of the DMA requests.

33.11.3.5 SYSTRIG0, SYSTRIG1 / External Trigger Input

Triggers can be used to initiate conversions that allow the synchronization of conversions to the external environment, rather than relying on software to signal to the ATD module when conversions should take place. The trigger source can be either from one of the analog inputs (from an external source) or from the SYSTRIG0 / SYSTRIG1 inputs (trigger signals from counters). In this mode there is no access to the registers.

The trigger is programmable to be edge or level sensitive with polarity control. In Section 33.10.2.2, “ATD External Trigger Channel Register (ATDETRIGCH)” and Section 33.10.2.1, “ATD Trigger Control Register (ATDTRIGCTL)” this is described in more details.

33.11.3.6 Flags

The flags reflect the current ATD status and can be used to generate interrupts.

33.11.3.7 Result Adjustment

The analog conversion circuit delivers a so called raw conversion result determined by the RSD-algorithm. To increase the accuracy the difference between an ideal transfer curve and the real transfer curve is minimized. This is done with this block. The adjustment is called multiply-and-add (MADD).

33.11.3.8 Conversion Control

The conversion control block takes care that each conversion is executed in the way specified in the conversion command word. It controls the analog conversion circuit which will have an influence to the delivered result.

33.11.4 Low Power / Operating Modes

The DMADC1032 can be configured for lower MCU power consumption in different ways:

- **Disabled mode:** After reset the ATD is disabled by default as the MDIS bit is set. Maximum power saving is provided. Although disabled most of the registers remain accessible. For exceptions please See 33.10.2.8 ATD Flag Register (ATDFLAG).

  To execute conversions the ATD must first be enabled by clearing the MDIS bit.
- **Normal mode:** To perform conversions, the ATD must be operating in Normal mode. To bring the ATD into normal mode the MDIS bit must be cleared. This will enable the ATD clocks. Now all registers are accessible and immediately a DMA-request for a command word is asserted. Also the analog conversion circuit is powered up. But due to the recovery time t_{REC} of the analog conversion circuit, the result of conversions started before t_{REC} has elapsed should be ignored as the results will be unreliable.[1]

  After the MDIS bit is set the ATD will return to Disabled mode. All clocks will be shut down and the analog conversion circuit will power down. Therefore a running conversion will be terminated and the command word in the ATDCW register will be discarded. The DMA request for a new command word will be deasserted. A pending DMA request of a conversion result will remain asserted until the ATD is enabled again and the result register is read.
- **Doze mode:** The ATD will enter Doze mode when the ATDMODE.DOZE bit is set **and** the MCU entered Doze mode. Otherwise the ATD will operate like in Normal mode. In Doze mode increased power saving is provided as the clocks are disabled. Please refer to Chapter 3, “Low Power Modes” for details on how to enter and exit low power modes.

  Entering Doze mode will terminate any conversion currently running and will flush the command queue. The DMA request for a new command word will be deasserted. A pending DMA request of a conversion result will remain asserted until the ATD is enabled again and the result register is read.

  Although dozed most of the registers remain accessible. Please refer to Section 33.10, “Memory Map and Register Definition” to identify which registers are not accessible.

  To leave DOZE mode either the MCU has to leave Doze mode or the DOZE bit must be cleared.

1. Refer to Appendix A, “Electrical Characteristics” for recovery time t_{REC}

When Doze mode is left all registers are accessible again and immediately a DMA-request for a command word is asserted. Also the analog conversion circuit is powered up. But due to the recovery time t_{REC} of the analog conversion circuit the result of conversions started before t_{REC} has elapsed should be ignored as the results will be unreliable.[1]

- **Debug mode:** This mode is **not** intended for power saving, but it allows the user to halt a conversion for software debug purpose. In this mode, all clocks are running and all registers are accessible. The analog conversion circuit will remain powered.

  To enter this mode, the MCU must enter Debug mode and ATDMODE.DEBUG bits must be set to one of the following combinations:

  — DEBUG = 00: If set the ATD does not support Debug Mode and therefore it will operate like in Normal mode.

  — DEBUG = 01: This mode is reserved for future use and should therefore not be set.

  — DEBUG = 10: When the MCU entered debug mode the ATD will freeze after the current conversion has finished. No new conversion will start until either Debug mode is left or a new command word is written to ATDCW. If there was no conversion running when Debug mode is entered, the ATD will freeze immediately.

  Debug mode can be left by either clearing the DEBUG bits or by forcing the MCU leave Debug mode. If the DEBUG bits are cleared the command word in ATDCW is discarded as writing to ATDMODE always causes any conversion to reset. If the MCU left Debug mode the command word in ATDCW will be executed.

  Writing a new command word to the ATDCW register **after** the ATD was frozen at the end of the previous conversion will start a new conversion. It must be ensured that the write access happens **after** the ATD was frozen. If it happens before the command will be stored in ATDCW but it will not start a conversion (Section 33.12.3.8, “Example 8: Debug Mode”).

  — DEBUG = 11: The ATD will freeze the current conversion immediately after the MCU entered Debug mode.

  Debug mode can be left by either clearing the DEBUG bits or by making the MCU leave Debug mode. If the DEBUG bits are cleared the frozen conversion is terminated and the command word in ATDCW is discarded as writing to ATDMODE always causes any conversion to reset. If the MCU left Debug mode the ATD will resume the conversion from the stage where it was frozen.

  Note that due to sample capacitor leakage the result of a frozen conversion will depend on the time the conversion was frozen. Therefore the result of this conversion should be discarded.

Please refer to Chapter 2, “Modes of Operation for details on how to enter and exit a mode.

33.11.5 Conversion Process

To execute conversions the ATD requires command words. These contain information when a conversion should start and how to continue after the conversion finished. In Section 33.11.5.1, “Command Queue it is described how command words are handled and Section 33.11.5.2, “Command Processing and DMA Requests” describes what effect the various command words have.

1. Refer to Appendix A, “Electrical Characteristics” for recovery time t_{REC}

33.11.5.1 Command Queue

Each command work must go through the three stages of the command queue:

1. ATDCW: Stores a new command word
2. ATDCC: Stores the command word for the conversion currently executed
3. ATDRR: Stores the result of a conversion which has been executed already.

A command word must be written to the new command word register (ATDCW). From there it is pushed further to the current command word register (ATDCC). A command word is pushed to ATDCC when a previous conversion finished. If there was no previous conversion the command word is immediately pushed to ATDCC after it was written to ATDCW.

The ATD conversion state machine (CSM) will execute the current command. Once the result of this conversion has been determined the result (including parts of the current command word) is pushed to the result register (ATDRR):

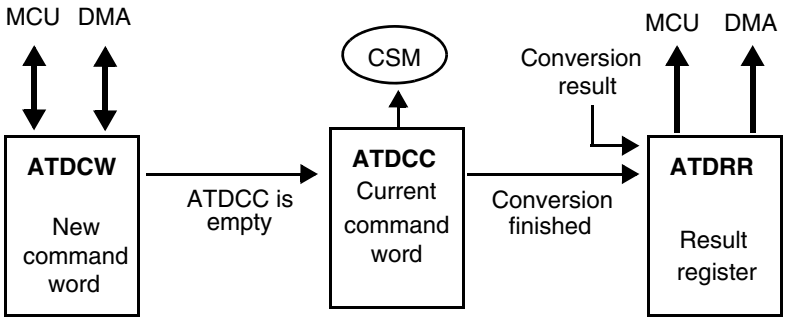


Figure 33-16. ATD Command Queue / Result Register

When the content of the ATDCW is pushed to ATDCC, there is again space for a new command word. This makes it possible to have one command word being executed while fetching a new command word.

Both, MCU and the DMA, have access to the ATDCW register. The current command word can neither be accessed via the MCU nor via the DMA. Only the conversion state machine is influenced by the current command word.

When a conversion finished the result register (ATDRR[31:16]) is updated with the bits ATDCC[31:16] (which have been the bits ATDCW[31:16] before) and the conversion result which is saved in ATDRR[15:0]. The MCU or the DMA can read the result register.

33.11.5.2 Command Processing and DMA Requests

When enabling the ATD a command word will automatically be requested. After receiving the first command word the ATD will start converting. The CWCM bits of the command word will tell the ATD how to perform the conversion and what action to take after the conversion has completed:

Table 33-19. Conversion Start Behavior

CWCM	Behavior
00	“conversion reset”: Writing such a command word to the command register ATDCW will immediately abort the current conversion. All bits in the command word beside the CWCM bits are ignored. If the aborted conversion was either “convert continuously” or “wait for trigger” the ATD will assert a DMA request for a new command. If the aborted conversion was “convert then pause” the ATD will not assert a DMA request for a new command. The DMA request for saving the conversion result remains unaffected in both cases.
01	“convert then pause”: The ATD will start a conversion immediately after the command word has been written to ATDCW and the previous conversion, if any, finished.
10	“wait for trigger”: The ATD will start a conversion only when the previous conversion, if any, finished and a trigger has been asserted. The trigger definition will be explained later.
11	“convert continuously”: The ATD will start a conversion immediately after the command word has been written to ATDCW and the previous conversion, if any, finished.

On completion of a conversion the value of the CWCM bits of the current command in ATDCC will determine how the ATD will continue:

Table 33-20. Conversion Continue Behavior

CWCM	Behavior
00	- Not applicable -
01	The ATD will do nothing but wait. A new conversion will not be started.
10	The ATD will try to start a new conversion. However, this can only be done if there is a new command word available. If this is the case a new conversion can (according to this command word) start. If not, the ATD will continue waiting for a new command.
11	The ATD will try to start a new conversion. However, this can only be done if there is a new command word available. If this is the case a new conversion can (according to this command word) start. If not, the ATD will continue waiting for a new command.

When a conversion finished, the ATD expects the ATDRR to be read. Therefore it will assert a DMA request. This action is independent of the value of CWCM, and runs in parallel to the conversion process (see Figure 33-17). If the result has been read the DMA request is deasserted.

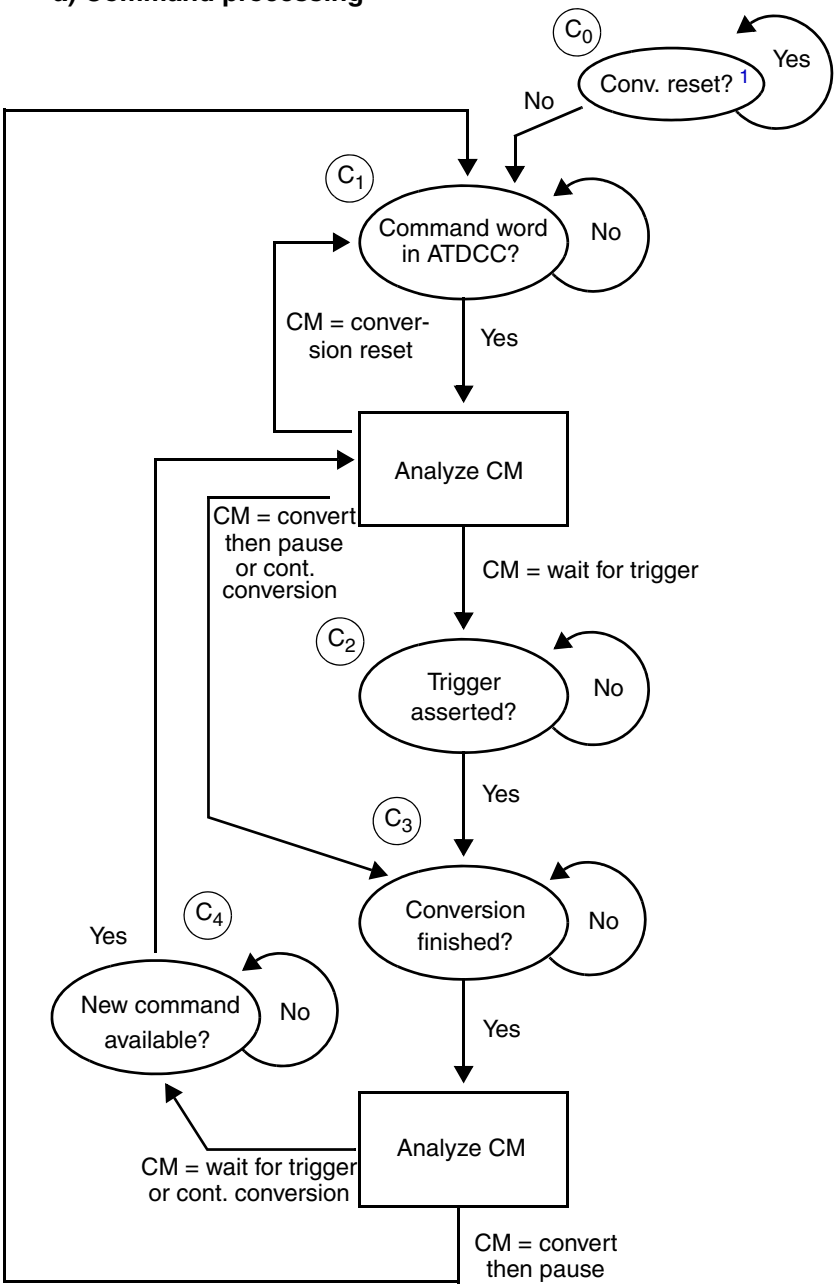
It is not possible for the conversion in progress to have a command word with CWCM=00. This is because writing a command word with this bit value to the ATDCW register will immediately stop any conversion.

While a conversion is running, the ATD will try to get a new command word by asserting a DMA request. When a new command word is written to the ATDCW (either by the DMA or the core) the DMA request is deasserted. This scheme allows a new command word to be available before the current conversion finishes.

The ATD module does not distinguish between an access by the DMA and an access by the MCU. Both can have access to the ATDCW or the ATDRR registers but only one at the same time.

The following two figures show the conversion process in more detail. As previously mentioned, saving results (see Figure 33-18a), fetching commands (see Figure 33-18b) and performing conversions (see Figure 33-17) are independent of each other, and are therefore split into three separate descriptions, shown in the following figures.

a) Command processing



1 A conversion Reset is caused either by a MCU reset or by writing a command word to ATDCW with ATDCW.CWCM = 0b00. When a reset occurs the ATD will always go to state C_0 independent of its previous state.

Figure 33-17. Flow Diagram for Command Processing

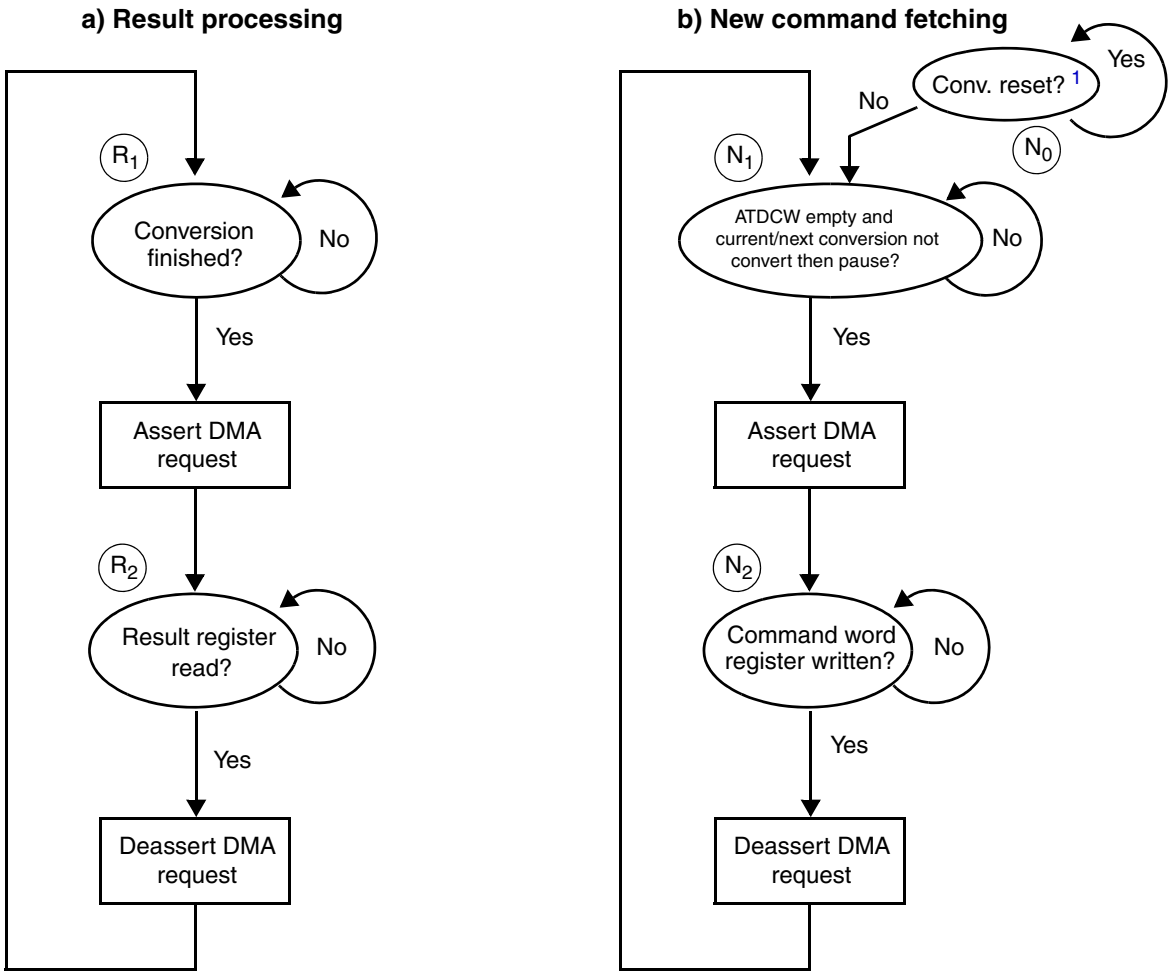


Figure 33-18. Flow Diagram for Result Saving/Command Fetching

33.11.5.3 Command / Result Timing

The following figure should illustrate when a conversion result can be expected after a conversion finished. It also shows at which time the conversion result should be read / transferred at the latest to avoid the result being overwritten by another conversion:

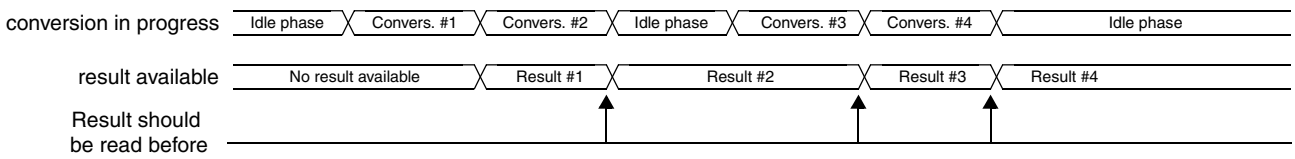
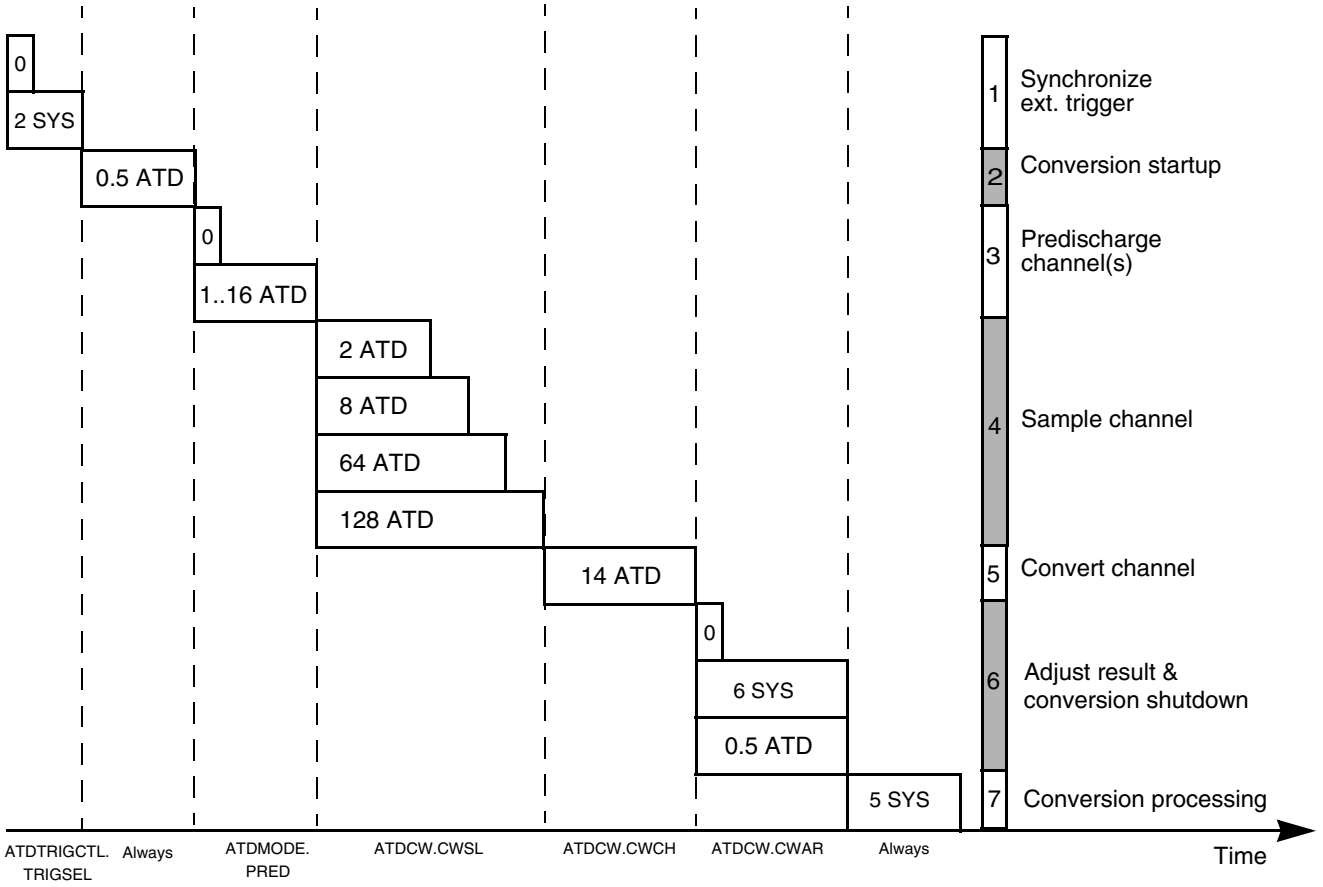


Figure 33-19. Command vs. Result Timing

33.11.6 Conversion Timing

The following figure should illustrate which phases exist during a conversion. Some of the phases are predetermined while others provide options:



NOTE: Not drawn to scale

Figure 33-20. Various Conversion Phases

The boxes contain the number of system clock cycles (x SYS) or ATD clock cycles (x ATD) needed for the corresponding phase. For details about the relation between the two clocks refer to Section 33.10.2.3, “ATD Prescaler Register (ATDPRE).”

In case that the number of ATD clocks is specified as “0.5 ATD” the rounded up number of system clock cycles needed for half an ATD clock is needed. E.g.: If the ATD clock lasts seven times the period of one system clock (prescaler is seven) then 0.5 ATD clock cycles will last four times the period of one system clock cycle. If the ATD clock lasts six times the period of one system clock (prescaler is six) then 0.5 ATD clock cycles will last three times the period of one system clock cycle etc.

The following paragraphs give an overview about the various phases:

Phase 1: Synchronize external triggers

If chip- external triggers are used to start conversions these triggers must be synchronized to the system clock which takes two system clock cycles. Refer to Section 33.10.2.2, “ATD External Trigger Channel Register (ATDETRIGCH),” for details about how to set external triggers.

If chip-internal triggers or no triggers are used then this phase will be skipped.

Phase 2: Conversion startup

Phase two can not be skipped as it is always required. It reflects the conversion startup phase.

Phase 3: Predischarge channel

There is the possibility to predischarge the sample capacitor including the parasitic capacity of an ATD channel. For further details please See 33.11.7 Conversions Using Predischarge. This phase is optional and will be skipped if the predischarge feature is not used.

Phase 4: Sample channel

During this phase the sample capacitor is charged with the voltage applied to the channel that should be converted. This phase can not be skipped. But there is the possibility to set different sampling lengths.

Phase 5: Convert channel

During this phase the voltage at the sample capacitor is converted and a raw conversion result is generated. The duration of this phase is independent of the result resolution. This phase can not be skipped.

Phase 6: Adjust result & conversion shutdown

The raw conversion result can be adjusted to decrease the total unadjusted error (TUE). This adjustment will take 6 system clock cycles. This phase is optional and will be skipped if the result adjustment feature is not used.

In parallel and independent to the result adjustment the conversion is shutdown. This will take half an ATD clock cycle.

The duration of this phase depends on what lasts longer: the result adjustment (if not skipped) or the conversion shutdown.

Phase 7: Conversion processing

Between phase #1 and phase #6 there are additional system clock cycles needed which have no mentionable meaning. To keep Figure 33.11.6 simple and to provide the full picture all these clock cycles have been summed up to phase #7.

Duration of a conversion

The duration of a conversion can easily be calculated by summing up the number of clock cycles needed for the executed phases.

The following example shows the calculation for one of the longest possible conversions:

1. External triggering selected: 2 SYS
2. Conversion startup: 0.5 ATD

3. Predischarge sample capacitor: 2 ATD
4. Sample channel (longest sampling time): 128 ATD
5. Convert channel #0 .. #31: 14 ATD
6. Adjust result & conversion shutdown: 6 SYS or 0.5 ATD
7. Conversion processing: 5 SYS

If the prescaler is smaller than 13 then phase #6 will last 6 SYS. Otherwise it will last 0.5 ATD. For this example we assume a prescaler of five. Therefore phase #6 will last 6 SYS. The total amount of clock cycles for this conversion will be:

2 SYS + 0.5 ATD + 2 ATD + 128 ATD + 14 ATD + 6 SYS + 5 SYS ***Eqn. 33-4***

This sums up to:

2 SYS + 3 SYS + 10 SYS + 640 SYS + 70 SYS + 6 SYS + 5 SYS = 736 SYS ***Eqn. 33-5***

The whole conversion described above will need 736 system clock cycles to complete.

33.11.7 Conversions Using Predischarge

The predischarge feature allows the detection of a broken connection between a sensor and the according ATD channel. A broken connection means that the input of an ATD channel is floating.

To detect a broken connection it is not enough to check that an ATD channel is always staying at VRH or VRL as a floating connection can have any voltage level. Additionally when switching to the broken channel the remaining charge from a previous channel can recharge the broken channel (phantom voltage).

By discharging the sample capacitor and the parasitic capacity of a broken ATD channel it is ensured that the conversion result of the broken channel will be close to 0x000. Please refer to the *Electrical Specification* for the maximum conversion value of a disconnected pin (CVDP).

The predischarge circuit has been designed in a way to be able to detect a broken pin between the sensor circuit and the pad. It has not been designed to detect a broken connection in the sensor RC-filter circuit. Depending on the dimensions of the RC-filter components it might still be possible to detect a broken connection in the RC-filter.

The predischarge circuit consists of the following components:

- pull-down resistor R_{sc} in parallel to the sample cap C_{sc} which will discharge all parasitic capacity between switch S_{mux} and the RSD conversion circuit. The pull-down resistor is controlled via switch S_{sc}
- pull-down device built in the pads. The device is controlled via the ATD when the ATD module is enabled. The pull-down device will discharge the parasitic capacity between the device pin and the ATD channel mux. It is unavoidable that the external RC-filter circuit is discharged at the same time.

The following figure provides further details about the predischarge circuit

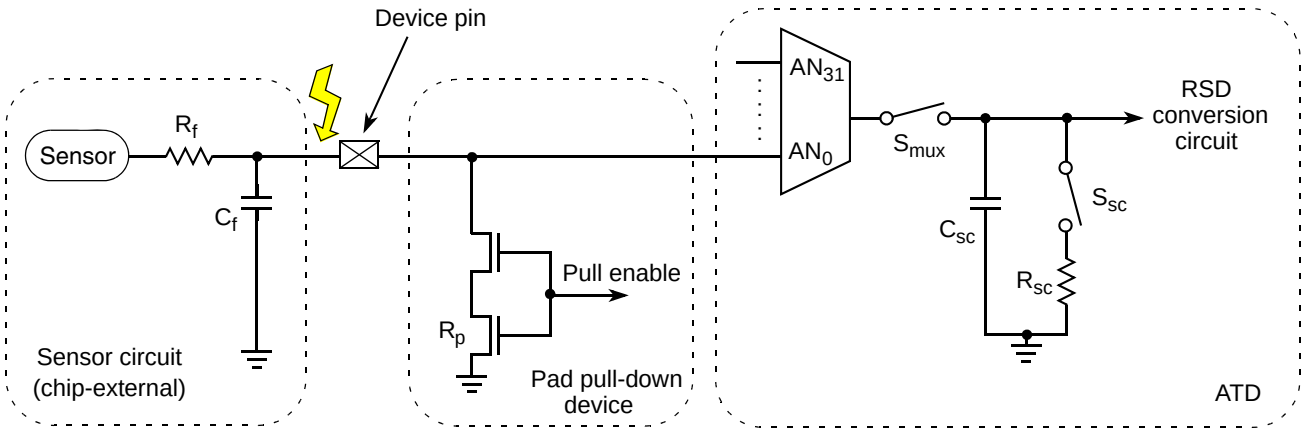


Figure 33-21. Predischarge Circuit

Before a conversion starts the sampling phase it can start the predischarge phase (see Section 33.11.6, “Conversion Timing” for details about the various conversion phases). The predischarge phase is only executed when the ATDMODE.PRED bit is set. Otherwise the phase is skipped. The length of the predischarge phase can be set by the ATDPTS.TIME bits. It can be set between 1 and 16 ATD clock cycles.

Which predischarge time needs to be selected depends on various factors e.g.:

- resistance of the PAD pull-down device (voltage dependent)
- parasitic capacity of the device pin (solder joint etc.)

Discharging the parasitic capacity will also discharge the external RC-filter. Therefore the length of the predischarge and the sampling phase should be selected in a way which allows the sensor to restore the filter capacitor voltage that was available before the predischarge phase started. Otherwise the conversion result value (ATDRR.RRCR) will be lower.

Figure 33-22 shows one timing example for a predischarge phase.:

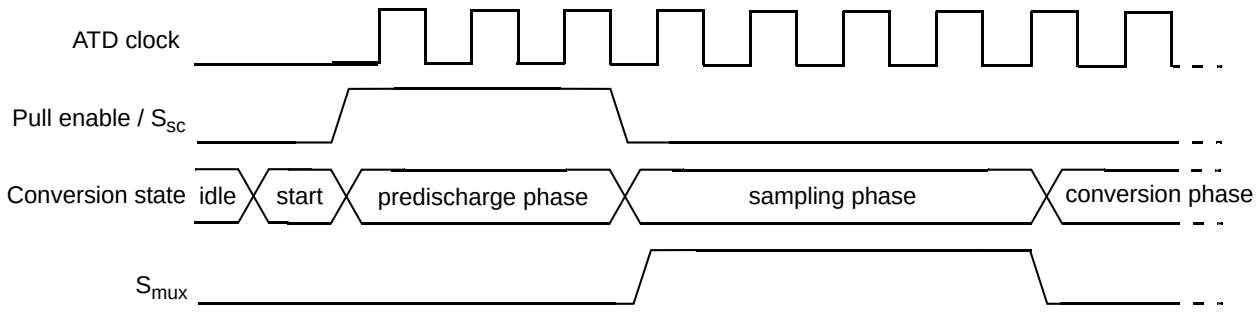


Figure 33-22. Predischarge Timing

In the example shown in Figure 33-22 for the predischarge phase length three ATD clock cycles were selected (ATDPTS.TIME = 0b0010). For the sampling phase four ATD clock cycles were selected.

The ATD is idle until a command word is written to ATDCW. Then the ATD will start with the conversion. It will check the command word and make it the current command. As predischarge is enabled in the ATDMODE register the predischarge phase will start.

The ATD will close the switch S_{sc} to the pull-down resistor R_{sc} and therefore discharge the sample cap. It will also enable the pull-down devices in the pad connected to the ATD channel which is selected by the CWCH bits. For the next three ATD clock cycles the sample cap and the parasitic capacity will be discharged.

Then the Sampling phase will started. Therefore the pull-down device in the pad and the pull-down resistor at the sample cap will be disabled. The switch S_{mux} will be closed to connect the current ATD channel with the sample cap. The sample cap will be charged for the next four ATD clock cycles.

After this time the conversion phase will start. The switch S_{mux} will be opened. The sample cap is now connected with the RSD-conversion circuit only. The conversion will continue from this point as described in e.g. Section 33.12.3.1, “Example 1: A Simple Conversion”.

33.12 Initialization/Application Information

In the following paragraphs, the initialization of the ATD module is shown. Some conversion examples are also explained. The focus in these examples will remain on the CWCM bits.

33.12.1 Initialization

After reset, the ATD module is disabled. To enable the ATD the MDIS bit in the ATDMODE register must be cleared. Once enabled, the ATD will wait, ready to receive command words before starting any conversion. Due to the recovery time t_{REC} of the analog conversion circuit, no conversion may be started before t_{REC} has elapsed[1]. All conversions started within t_{REC} will produce unreliable results.

The following initialization sequence is recommended:

1. Enable ATD: Clear ATDMODE[MDIS]. Ensure t_{REC} elapsed.
2. Set system clock division ratio: write ATDPRE
3. Set trigger:
   — Select trigger source, polarity and sensitivity: write ATDTRIGCTL
   — If trigger source is selected to be an analog input channel determine input channel number and write it to ATDETRIGCH
4. Choose operating modes: write DOZE / DEBUG / PRED / WARP in ATDMODE

The steps #1 to #4 can be done with one single 32-bit write instead of four 8-bit writes.

5. Determine result adjustment constants (see Section 33.12.2, “ATD Calibration / Result Adjustment).
6. Enable desired interrupt(s)

Now the ATD is initialized and ready to do conversions. To start conversions and receive results the following options exist:

1. Refer to Appendix A, “Electrical Characteristics” for recovery time t_{REC}

7. Start conversions:
   — Either program the DMA to write command words to ATDCW or
   — manually write the command word sequence to ATDCW using ATDFLAG.CQNF as an indicator for more command words needed
8. Get conversion results:
   — Either program the DMA to read results from ATDRR or
   — manually read the result sequence from ATDRR using ATDFLAG.CC as an indicator for the end of a conversion

Please refer to 12.2, “The SPP DMA Controller Module (SPP_DMA2)” for further details about how to use the DMA.

33.12.2 ATD Calibration / Result Adjustment

An ideal ATD transfer curve put into a coordinate system is linear, starting in the origin (no offset). It reaches its maximum value 0xFFF (for 12-bit right justified conversions) at (V_{RH}-V_{RL}). The real device behavior will differ slightly. Environment factors during chip production or chip operation (e.g. temperature, voltage etc.) can have an influence.

As the ATD transfer curve is very linear the only two transfer curve factors that will vary are the slope (gain) and the origin (offset). The ATD has the possibility to do a numerical adjustment of the transfer curve: the gain and / or the offset of the raw conversion result can be adjusted by doing a multiply-and-add operation via the MADD unit. This feature can be enabled/disabled on a command word level by setting the ATDCW.CWAR bit. The adjustment of the raw conversion result is called a “result adjustment.”

This adjustment requires two factors: gain (GCC) and offset (OCC). The determination of these two factors is called “calibration.” A calibration must be done each time when the chip operation factors are changing in a way which that it affects the ATD accuracy. The calibration must **not** be done before each conversion.

NOTE

Changing the gain and offset has no influence to the analog behavior.

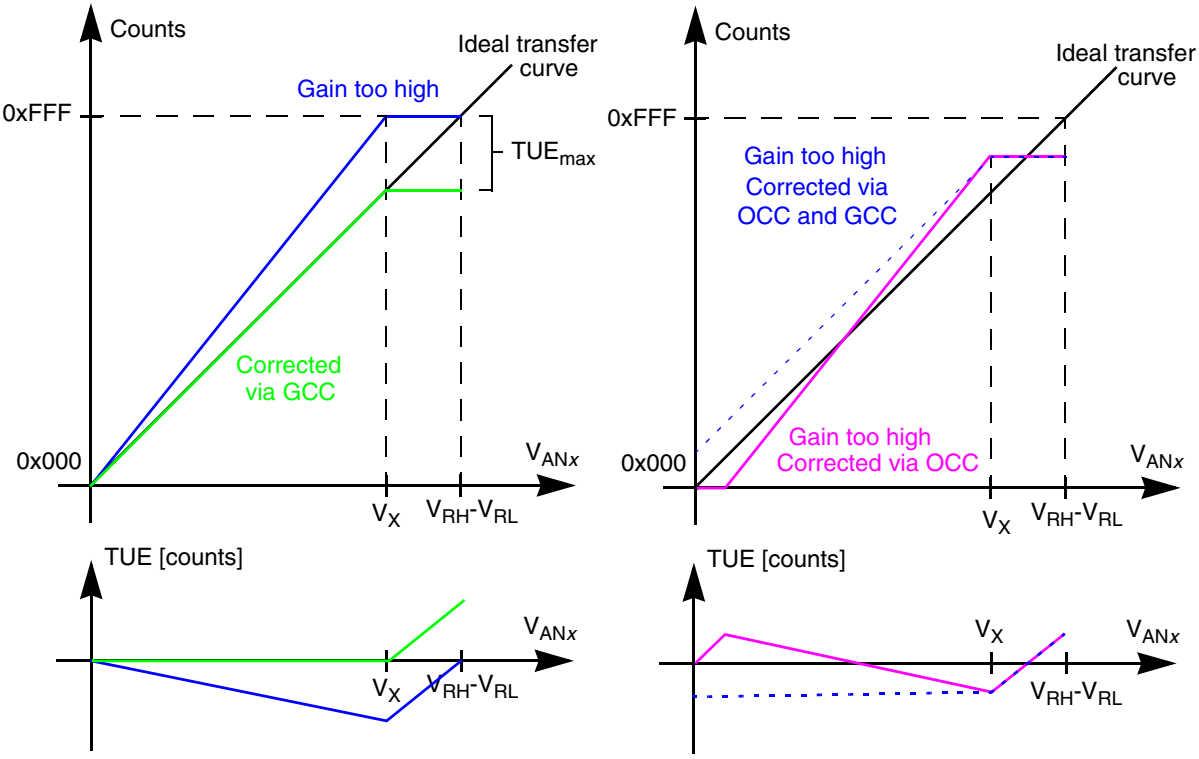
In addition to the MADD unit, the warp-feature is available. This will influence the analog behavior of the ATD. By setting this bit the gain is decreased by a fixed value to avoid that the maximum conversion result of 0xFFF is already reached before V_{RH}-V_{RL} is reached (see Figure 33-23). Enabling the Warp-feature will usually overcompensate an existing gain error and increase the offset. Therefore it is required to adjust the GCC and OCC factors after warp has been enabled. How this is done is explained in Section 33.12.2.3, “Steps Required For Calibration”. Refer to Section 33.10.2.4, “ATD Operating Modes Register (ATDMODE),” for details about how to enable this feature.

33.12.2.1 Gain error

33.12.2.1.1 Gain Error Correction via GCC (Gain Too High)

If the ATD has a gain which is too high, the maximum result value of 0xFFF will already be reached at V_x (see the blue curve in Figure 33-23). This means that for all voltages higher than V_x the transfer curve will be clipped and therefore the result will be 0xFFF.

The total unadjusted error (TUE), which is the difference between the ideal and the actual result value, is shown in the bottom part of Figure 33-23: With the gain too high the TUE reaches its maximum at V_x. From there it decreases again:



a) Gain adjustment: GCC < 1

b) Offset adjustment: OCC < 0

Figure 33-23. Gain Error (Gain Too High) Compensated via GCC and OCC

If the GCC value is chosen correctly the adjusted transfer curve (green curve in Figure 33-23a) will fit the ideal transfer curve and the TUE will become zero for all results until V_x is reached. When reaching V_x the TUE will increase to the maximum value (TUE_{max}) which is the same as in the unadjusted case. Therefore TUE_{max} values in the unadjusted and adjusted cases are always the same.

33.12.2.1.2 Gain Error Correction via OCC (Gain Too High)

Another way to reduce the TUE is the offset adjustment. In the given example it is possible to half the TUE_{max} by changing the offset: When setting the offset to TUE_{max} / 2 the transfer curve (magenta curve in Figure 33-23b) will be moved down. The TUE in this case is shown at the bottom of Figure 33-23b. It

can be seen that the TUE has been successfully halved with the disadvantage that the maximum TUE occurs at the beginning and the end of the transfer curve.

33.12.2.1.3 Gain Error Correction via OCC and GCC (Gain Too High)

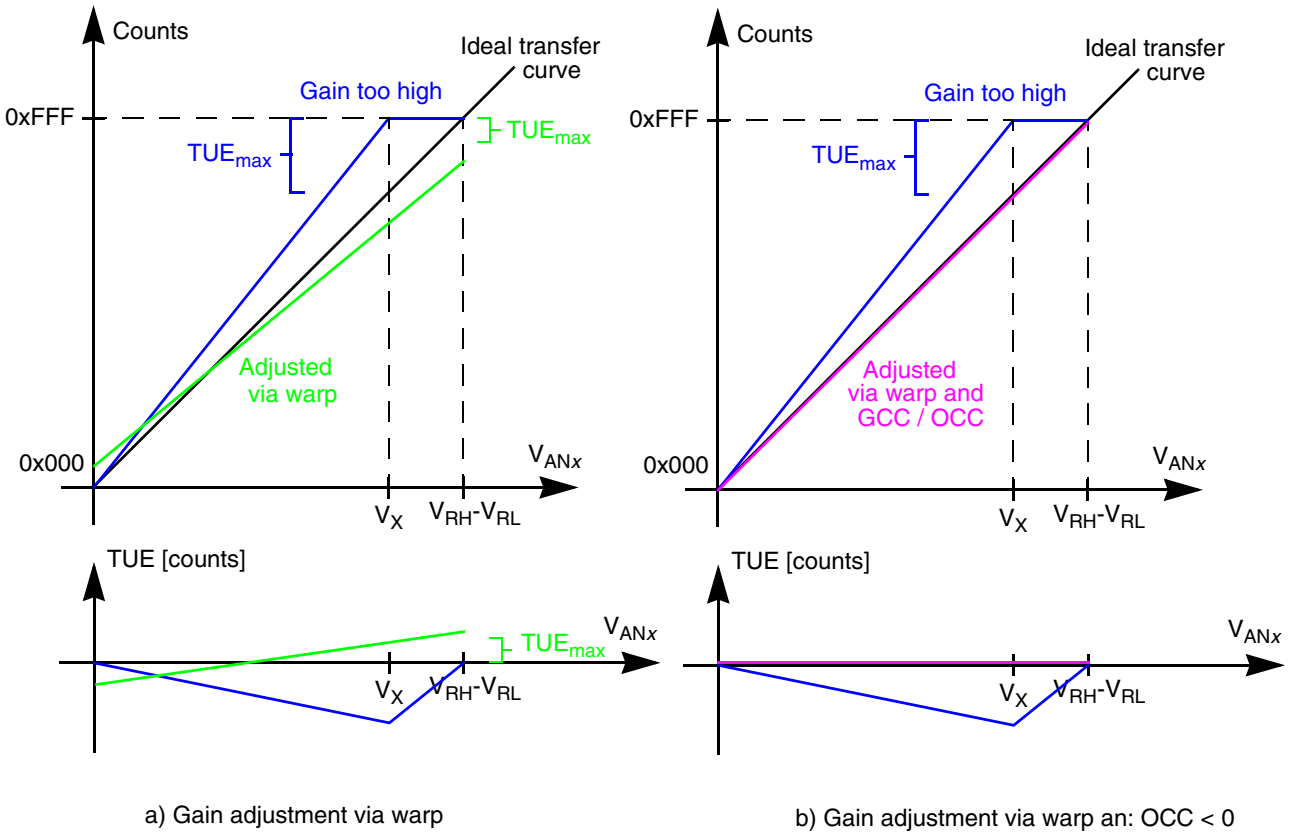
It depends on the application whether the gain error correction is done only via GCC, only via OCC or with both. The advantage of changing both is that the TUE can be made constant until V_x is reached (see blue dotted line in Figure 33-23b). In this case TUE_{max} is the same as when changing OCC only.

33.12.2.1.4 Gain Error Correction via Warp (Gain Too High)

In Section 33.12.2.1.1, “Gain Error Correction via GCC (Gain Too High)” and Section 33.12.2.1.2, “Gain Error Correction via OCC (Gain Too High)” it was shown how the TUE can be reduced by doing a numerical adjustment via GCC and OCC.

Beside this numerical adjustment the ATD has another feature: Warp. This feature will influence the analog behavior of the conversion circuit. It will adjust the gain and the offset by a fixed value which is completely independent of OCC and GCC:

Figure 33-24a assumes a gain error (blue line) with gain too high. By enabling the warp-feature the gain error is reduced and the offset is increased. Usually the error is overcompensated (green curve in Figure 33-24a). Therefore it is necessary to correct the remaining TUE via GCC and OCC. For further details please see Section 33.12.2.1.5, “Gain Error Correction via GCC (Gain Too Low)”.



a) Gain adjustment via warp

b) Gain adjustment via warp an: OCC < 0

Figure 33-24. Gain Error (Gain Too High) Compensated via Warp

33.12.2.1.5 Gain Error Correction via GCC (Gain Too Low)

If the ATD transfer curve shows a gain that is too low (see green line in Figure 33-25) this can simply be corrected by increasing the GCC. If done correctly, the ideal transfer curve and the actual transfer curve will match and the TUE can be reduced to zero.

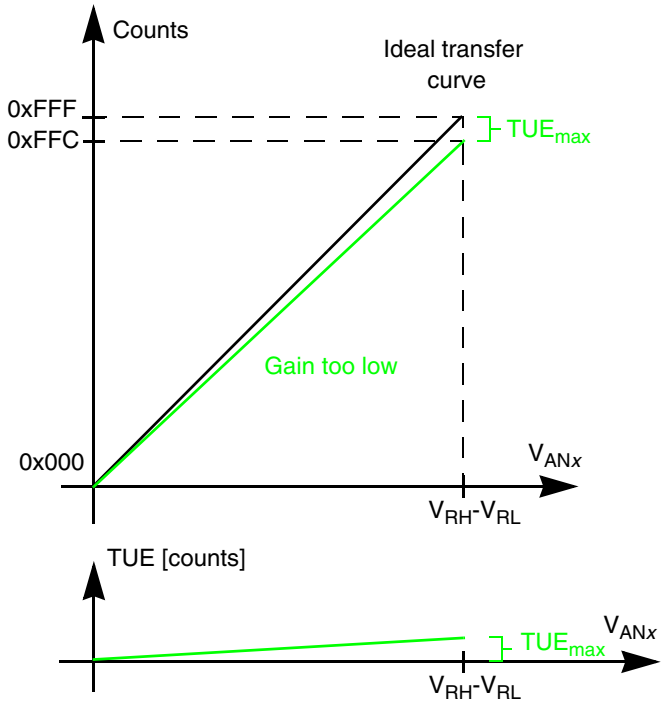


Figure 33-25. Gain Error (Gain Too Low) Compensated via GCC

33.12.2.2 Offset Error

Offset errors will lead to a constant TUE. The transfer curve will be clipped either at the beginning or the end of the curve depending on the offset being negative or positive (see Figure 33-26). Offset errors can easily be corrected by changing the OCC value.

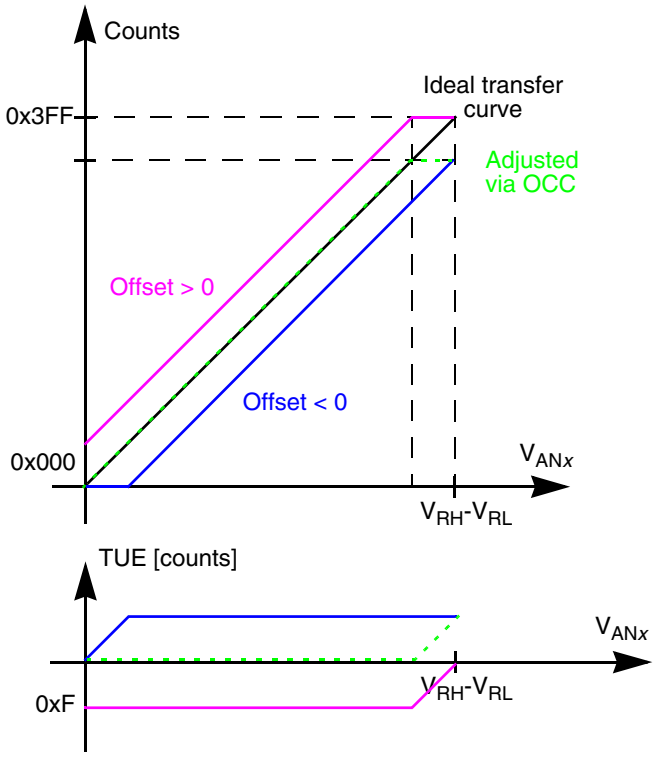


Figure 33-26. Offset Error Compensated via OCC

33.12.2.3 Steps Required For Calibration

To reduce the total unadjusted error of the conversion results the ATD provides a multiply-and-add mechanism (MADD) which will adjust the raw conversion result. This adjustment requires two factors: Gain Calibration Constant (GCC) and Offset Calibration Constant (OCC). The determination of these two factors is called “calibration.” To determine the two factors the following must be done:

1. Execute initialization steps 1 to 4. (see Section 33.12.1, “Initialization”)
2. Sample special channel 75% x (V_{RH} - V_{RL}) multiple times
3. Sample special channel 25% x (V_{RH} - V_{RL}) multiple times
4. Determine gain calibration constant GCC from Gain
5. Determine offset calibration constant OCC from Offset
6. Write GCC and OCC to ATDCAL

This sequence is described in details in the following sections.

The calibration must always be repeated when influence factors for the ATD accurracy change. As the ATD supply voltages are not expected to drift the most important influence factor is the temperature. Although the ATD is very temperature resistant in high accurracy applications a drift might get visible.

The temperature usually changes rapidly when the device is turned on and the ambient temperature is very low. During this time the gain and the offset are expected to change. But once the device heated up the gain and offset should remain constant.

To compensate temperature influences it is recommended to do the calibration e.g. every 100 .. 500ms especially when the device is heating up. The calibration frequency might be reduced when the temperature is expected to be stable.

33.12.2.3.1 Sample Special Channel 75% x (V_{RH} - V_{RL})

To sample this channel a command word with the value 0x638B must be written to ATDCW. Table 33-21 shows of what settings this value is composed:

Table 33-21. Command Word #1 to Determine Gain and Offset

Bit(s)	Value	Description
CWAR	0b0	use raw conversion result
CWSC	0b1	special channel
CWGI	0b1	Generate an interrupt
CWCM	0b00	Continuous conversion
CWSL	0b01	Eight ATD clock cycles sampling time
CWNF	0b1100	right justified unsigned, 12 bit
CWCH	0b0_1011	special channel 75% x (V_{RH} - V_{RL})

When the conversion finished (CC flag is set) the result can be read. To reduce the influence on the conversion result caused by noise on the ATD supply pins this conversion should be repeated several times. Depending on the result variance the mean or median can be used.

NOTE

When converting this special channel the predischarge feature must be enabled to reduce the influence of parasitic charges. The predischarge time can be two ATD clock cycles or higher.

33.12.2.3.2 Sample Special Channel 25% x (V_{RH} - V_{RL})

To sample this channel a command word with the value 0x638C must be written to ATDCW. This value represents the bit values shown in Table 33-21 but CWCH = 0b0_1100.

When the conversion finished (CC flag is set) the result can be read. To reduce the influence on the conversion result caused by noise on the ATD supply pins this conversion should be repeated several times. Depending on the result variance the mean or median can be used.

NOTE

When converting this special channel the predischarge feature must be enabled to reduce the influence of parasitic charges. The predischarge time can be two ATD clock cycles or higher.

33.12.2.3.3 Determine Gain Constant GCC / Determine Offset Constant OCC

The ideal transfer curve put into a coordinate system is linear, starting in the origin with a slope of 1. The real device behavior can slightly differ. As the ATD is very linear the only two transfer curve factors that can vary are the slope (gain) and the origin (offset). Both values must be determined and adjusted.

To determine them two measurements are required. The first measurement (RAW_{75}) is done at 75% of the maximum voltage (See 33.12.2.3.1 Sample Special Channel 75% x (VRH - VRL)). The second one (RAW_{25}) is done at 25% of the maximum voltage (See 33.12.2.3.2 Sample Special Channel 25% x (VRH - VRL)). Both values are compared with the expected values (EXP_{75} and EXP_{25}):

$$EXP_{75} = Gain \times RAW_{75} + Offset + 0.5 \quad \textit{Eqn. 33-6}$$

$$EXP_{25} = Gain \times RAW_{25} + Offset + 0.5 \quad \textit{Eqn. 33-7}$$

The expected values are:

$$EXP_{75} = 0.75 \times 2^{12} = 0.75 \times 4096 = 3072 \quad \textit{Eqn. 33-8}$$

$$EXP_{25} = 0.25 \times 2^{12} = 0.25 \times 4096 = 1024 \quad \textit{Eqn. 33-9}$$

Now the gain can be calculated:

$$Gain = \frac{EXP_{75} - EXP_{25}}{RAW_{75} - RAW_{25}} = \frac{3072 - 1024}{RAW_{75} - RAW_{25}} = \frac{2048}{RAW_{75} - RAW_{25}} \quad \textit{Eqn. 33-10}$$

The offset can be calculated either like this:

$$Offset = EXP_{75} - Gain \times RAW_{75} - 0.5 = 3072 - Gain \times RAW_{75} - 0.5 \quad \textit{Eqn. 33-11}$$

or like this:

$$Offset = EXP_{25} - Gain \times RAW_{25} - 0.5 = 1024 - Gain \times RAW_{25} - 0.5 \quad \textit{Eqn. 33-12}$$

The ‘0.5’ in Equation 33-11 and Equation 33-12 moves the ATD transfer curve down by half an LSB and is therefore reducing the Total Unadjusted Error (TUE) by half an LSB.

The Gain Calibration Constant can be calculated like this:

$$GCC = Gain \times 16384 \quad \textit{Eqn. 33-13}$$

For details about GCC, see Section 33.10.2.5, “ATD Calibration Register (ATDCAL).

For offset values >= 0 the Offset Calibration Constant can be calculated like this:

$$OCC = Offset \times 4 \quad \textit{Eqn. 33-14}$$

For offset values < 0 the Offset Calibration Constant can be calculated like this:

$$OCC = Offset \times 4 + 16384 \quad \textit{Eqn. 33-15}$$

After OCC and GCC have been calculated they must be written to the ATDCAL register. Now each raw conversion result can be automatically adjusted by setting the ATDCW[CWAR] bit.

It is advisable to re-convert the reference channels using the new calibration factors like described in 33.12.2.3.1, “Sample Special Channel 75% x (VRH - VRL) and in 33.12.2.3.2, “Sample Special Channel 25% x (VRH - VRL). The results should match the ideal 25% and 75% results plus a margin that depends upon the noise on the system.

33.12.2.3.4 Adjustment Example

For a better understanding how the single steps mentioned in Section 33.12.2, “ATD Calibration / Result Adjustment work an example is provided in this section:

It is assumed that the (averaged) raw results from the 75% x (V_{RH} - V_{RL}) and the 25% x (V_{RH} - V_{RL}) conversion are 2951 and 951. Therefore the gain is calculated as:

$$Gain = \frac{2048}{RAW_{75} - RAW_{25}} = \frac{2048}{2951 - 951} = \frac{2048}{2000} = 1.024 \qquad \textit{Eqn. 33-16}$$

The gain calibration constant (GCC) is calculated as:

$$GCC = Gain \times 16384 = 16777.2 \approx 16777 \qquad \textit{Eqn. 33-17}$$

When calculating GCC it is highly recommended to use truncation instead of rounding.

The offset is calculated as:

$$Offset = 3072 - Gain \times RAW_{75} - 0.5 = 3072 - \frac{2048}{2000} \times 2951 - 0.5 = 49.676 \qquad \textit{Eqn. 33-18}$$

As the offset is positive the offset calibration constant (OCC) is calculated as:

$$OCC = Offset \times 4 = 49.676 \times 4 = 198.70 \approx 198 \qquad \textit{Eqn. 33-19}$$

When calculating OCC it is highly recommended to use truncation instead of rounding.

Now OCC and GCC can be written to the ATDCAL register. To verify that the two constants have been calculated correctly 75% x (V_{RH} - V_{RL}) and 25% x (V_{RH} - V_{RL}) can be converted again. This time the CWAR bit in the command word must be set. The results should be 3072 and 1024.

33.12.3 Conversion Examples

This section provides some conversion examples which should clarify the ATD working behavior. The following examples are given:

- Example 1: A Simple Conversion
- Example 2: A Simple Conversion Sequence (Convert then Pause)
- Example 3: Interrupted Conversion Sequence
- Example 4: Edge-triggered Conversion
- Example 5: Level-triggered Conversion
- Example 6: Queue Running Idle

- Example 7: Entering Low-power Mode During a Conversion
- Example 8: Debug Mode

Any additional meaning of the remaining command word bits is explained in Section 33.12.3.9, “Conversion Mechanism (bits CWCH, CWNF, CWGI, CWSC, CWAR)”. If not further specified the meaning of the bits in ATDCW are either not important for the given example or are intentionally left out to avoid confusion. All examples refer to the flow diagrams described in Figure 33-17 and Figure 33-18.

33.12.3.1 Example 1: A Simple Conversion

For this example we assume that the ATD has been enabled and t_{REC} has elapsed. At a certain point, a command word is written to the ATDCW register. It is assumed that CWCM is set to “convert then pause”. After the write the CSM will immediately go to state C_2 and execute the conversion. When the conversion is finished, the CSM will store the result in the result register ATDRR and then go back to C_1 (Figure 33-18). The CSM will also set the pause bit (CP) in the flag register ATDFLAG. If the CPIE bit is set, an interrupt will be generated.

As there is a result available in ATDRR the Result State Machine (RSM) will transition from R_1 to R_2 (Figure 33-18**b**) and will assert a DMA request. The DMA request will stay asserted until the ATDRR register is read. After the result has been read, the RSM will go back to the idle state C_1.

In Figure 33-27 the conversion procedure for this example is shown:

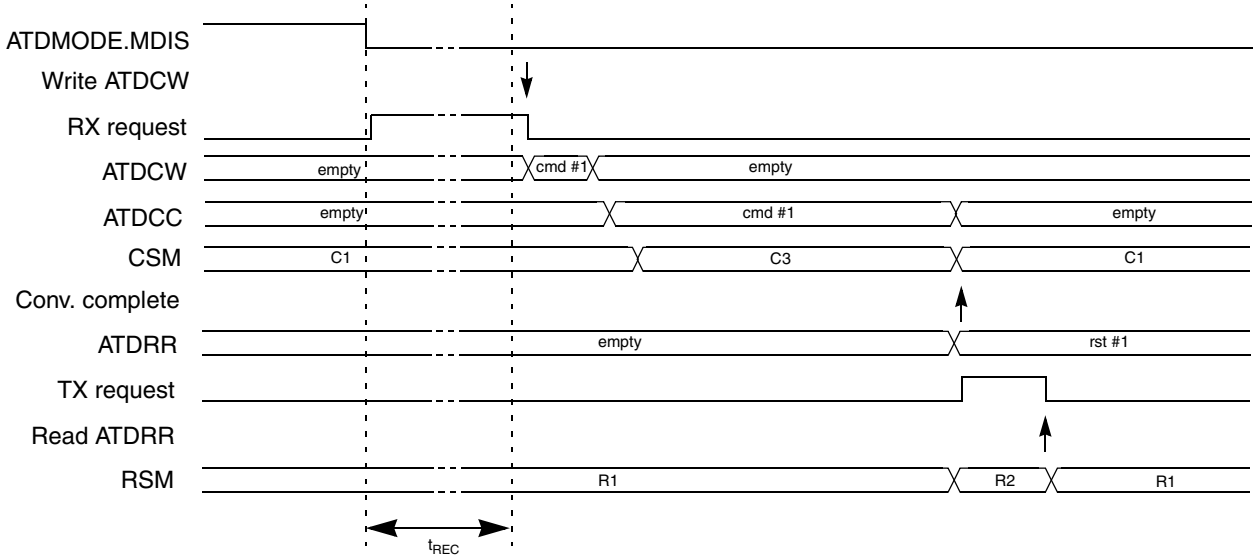


Figure 33-27. Conversion Procedure for Example 1

33.12.3.2 Example 2: A Simple Conversion Sequence (Convert then Pause)

For this example three conversions will be performed in a row with the CSM assumed to be starting from C_1 (Figure 33-17). The command words have the following CWCM bit values:

1. CMD #1, CMD #2: CWCM= “11” (convert continuously)
2. CMD #3: CWCM= “01” (convert then pause)

3. CMD #4, CMD #5: CWCM= “11” (convert continuously)

After writing the first command (cmd #1) to the ATDCW register the CSM will transition from state C_1 to C_3. The command word is pushed from ATDCW to ATDCC.

As ATDCW is again empty the ATD will assert a DMA request to receive a new command word. The state machine that fetches a new command, now called NSM, will transition from state N_1 to N_2. It will wait for a new command word (cmd #2) being written to ATDCW. This process has no influence on the current conversion. When the command word is available the NSM will go back from state N_2 to N_1.

After completion of conversion #1 (the result was pushed to ATDRR) the CSM will transition to state C_4 where it will check that a new command word is available. As in this example this is the case the CSM will transition to state C_3 which will push the command word from ATDCW to ATDCC and start the second conversion.

If there was no command word available in ATDCW, the CSM would stay in state C_4 until the command word was received.

In parallel to the second conversion the RSM will assert a DMA request to indicate that there is a result pending in ATDRR (rst #1).

If the second conversion finishes before the result of the first conversion in the ATDRR register is read a conversion loss will occur. This will be indicated by setting the CRL flag in the ATDFLAG register. For this example it is assumed that ATDRR is read in time.

Also during the second conversion the NSM will assert a DMA request to receive a new command word as ATDCW is empty again. It is assumed that a new command word is available before conversion #2 will finish.

After completion of conversion #2 the CSM will again transition from C_3 to C_4 and check the availability of a new command. As in this example the third command word is available in ATDCW the CSM will transition to C_3 and execute conversion #3.

Again it is assumed that the second conversion result is read before the third conversion finishes.

As the third conversion is “convert then pause” the NSM will not assert a DMA request for a new command word.

After the third conversion has finished, the CSM will move from state C_3 to state C_1. This is because the CWCM bits of the last command word are set to “convert then pause”. The ATD will not start any further conversions, but will try to save the result of the third conversion.

In Figure 33-28 the conversion procedure like described above is shown:

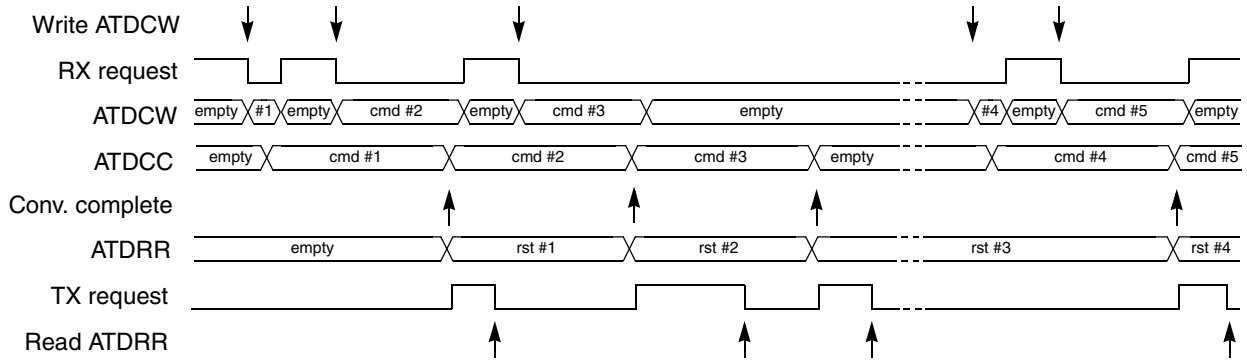


Figure 33-28. Conversion Procedure for Example 2

To make the ATD fetch new command words it is required to write another command word to ATDCW (cmd #4). As this command word has CWCM = “convert continuously” another command word is requested.

The conversion sequence cmd #4 and cmd #5 will be executed in the same way like for cmd #1 to cmd #2.

33.12.3.3 Example 3: Interrupted Conversion Sequence

This example will show how a continuous conversion can be interrupted. Again for simplification all but the CWCM bits are disregarded:

- CMD #1, CMD #2, CMD #3: CWCM = “continuous conversion”
- CMD #4: CWCM = “conversion reset”

When the first conversion is started according to example 2 (CSM goes from C_1 to C_3 and finally to C_4) the ATD will keep switching between the states C_3 and C_4 as long as new command words with CWCM = “continuous conversion” are written to ATDCW. In this example this is true for command #1 and #2 (cmd #1 and cmd #2). The result of each conversion is saved in the same way as described in example 2.

After the third conversion (cmd #3) started the next command (cmd #4) is written to ATDCW. Immediately after the write access the third conversion is terminated. It is irrelevant whether the current CSM state was C_3 or C_4 or C_2 (as explained in Section 33.12.3.4, “Example 4: Edge-triggered Conversion”). The CSM will always move to state C_1 (See Figure 33-17).

If the conversion is terminated while a DMA request for a result is pending (the RSM is in state R_2) this DMA request remains asserted until ATDRR was read.

In Figure 33-29 the conversion procedure for this example is shown:

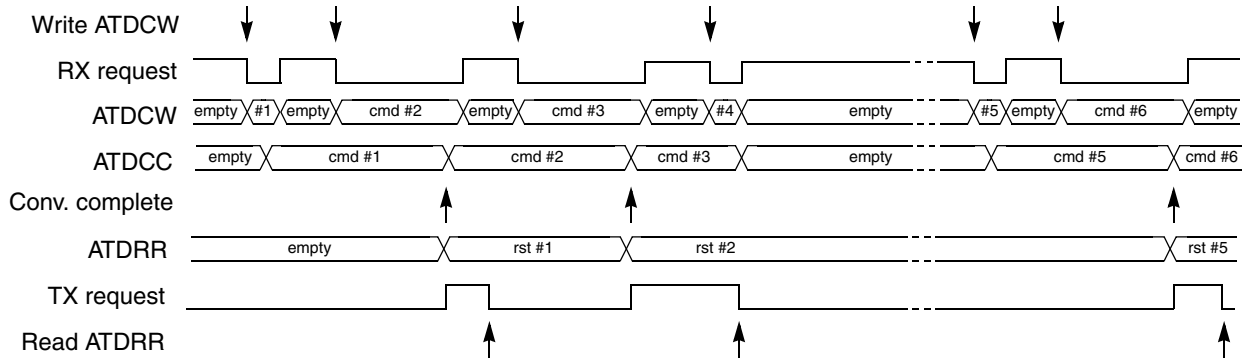


Figure 33-29. Conversion Procedure for Example 3

After the current conversion was terminated the commands in ATDCC and ATDCW are discarded. As the the queue is empty again a DMA request for a new command word is asserted. In this example it is assumed that the whole sequence is repeated which is indicated with cmd #5 and #6. To keep the togetherness of command and result the third conversion result which belongs to cmd #5 is named rst #5 although it is the third conversion result in this example.

33.12.3.4 Example 4: Edge-triggered Conversion

This example shows how a conversion can be started by a trigger other than a write to the ATDCW. The example uses a rising-edge sensitive trigger from analog channel #7. The trigger is set up by:

- ATDTRIGCTL[TRIGSEL] = 11 (use analog input channel as trigger)
- ATDTRIGCTL[TRIGP] = 1 (trigger is rising-edge sensitive)
- ATDTRIGCTL[TRIGLE] = 0 (trigger is edge sensitive)
- ATDETRIGCH[ETRIGCH] = 0_0111 (analog channel #7 is trigger)

Each write to either the ATDTRIGCTL or ATDETRIGCH register will stop any conversion that may be executing. For this example it is assumed that the registers have been configured previously and no conversions were being executed.

The following command words are used for this example:

- CMD #1 .. CMD #3: CWCW = “wait for trigger”
- CMD #4: CWCM = “conversion reset”

When the first command word is written to the ATDCW register, the CSM will move from state C_1 to C_2, where it will wait for a rising edge on the analog input channel 7. When this trigger arrives, the CSM will transition from state C_2 to C_3 and the actual conversion will be executed (as described in Section 33.12.3.1, “Example 1: A Simple Conversion,” to Section 33.12.3.3, “Example 3: Interrupted Conversion Sequence”).

The second command word is executed in the same way. After this conversion finished and the third conversion started cmd #4 is written to ATDCW. Immediately after the write the conversion is reset. The command words in ATDCC and ATDCW are discarded.

If a trigger arrives after the conversion has been reset this will be flagged as an external trigger overrun and the ETO flag will be set. In this example there is no additional trigger and therefore the ETO flag remains unchanged.

Figure 33-30 shows the conversion procedure for this example

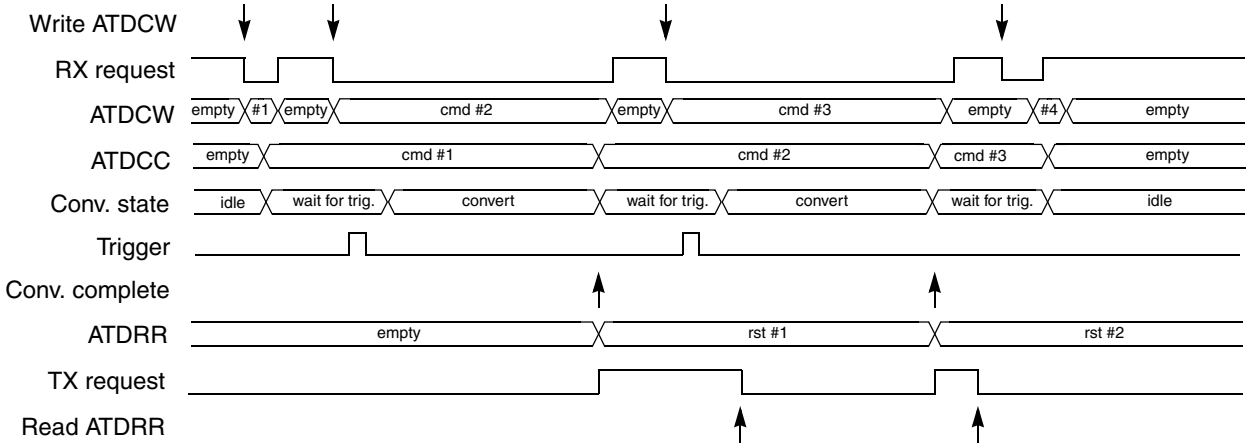


Figure 33-30. Conversion Procedure for Example 4

Figure 33-31 illustrates some examples of valid edge-sensitive triggers. The minimum trigger pulse length must be one system clock cycle. Longer trigger pulses are allowed. The minimum trigger length is independent of the ATD clock period. Due to synchronization and internal processing, the conversion start (symbolized by the arrows in Figure 33-31) will be delayed by 4 system clock cycles.

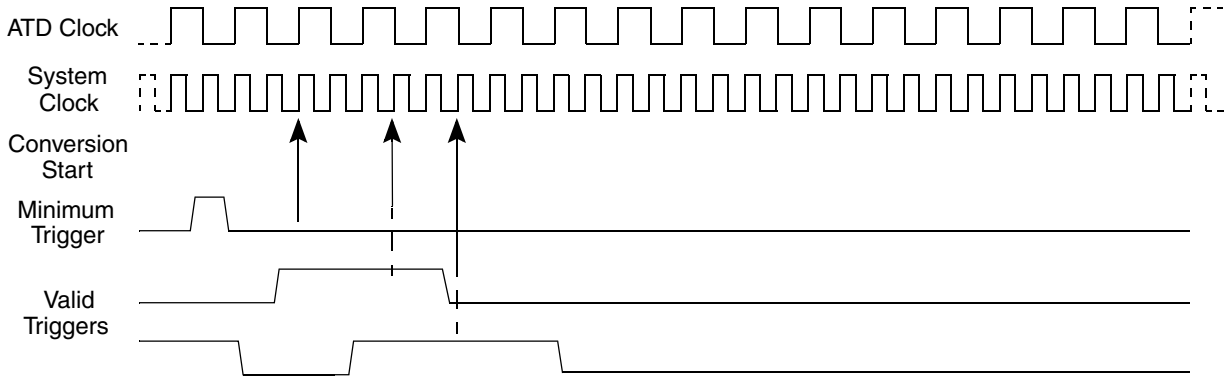


Figure 33-31. ATD Edge-Based Trigger Example

33.12.3.5 Example 5: Level-triggered Conversion

This example shows a conversion started by a high-level sensitive trigger asserted at the analog input channel 7, as shown in Figure 33-33. The trigger is set up by:

- ATDTRIGCTL: TRIGSEL = 11 (use analog input channel as trigger)
- ATDTRIGCTL: TRIGP = 1 (trigger is high-level sensitive)
- ATDTRIGCTL: TRIGLE = 1 (trigger is level sensitive)
- ATDETRIGCH: ETRIGCH = 0111 (analog channel #7 is trigger)

Each write to either the ATDTRIGCTL or ATDETRIGCH register will stop any conversion that may be executing. For this example it is assumed that the registers have been configured previously and no conversions were being executed.

All command words use the same setting:

- CMD #*n*: CWCW = “wait for trigger”

When the first command word is written to the ATDCW register, the CSM will move from state C_1 to C_2, where it will wait for a high level on the analog input channel 7. When this trigger arrives, the CSM will transition from state C_2 to C_3 and the actual conversion will be executed (as described in Section 33.12.3.1, “Example 1: A Simple Conversion,” to Section 33.12.3.3, “Example 3: Interrupted Conversion Sequence”).

The second command word is executed in the same way. During the conversion the trigger is deasserted. Even so the conversion will complete. But as the trigger was deasserted the third conversion will not start after the second one finished. It will start once the trigger is asserted again.

Figure 33-30 shows the conversion procedure for this example:

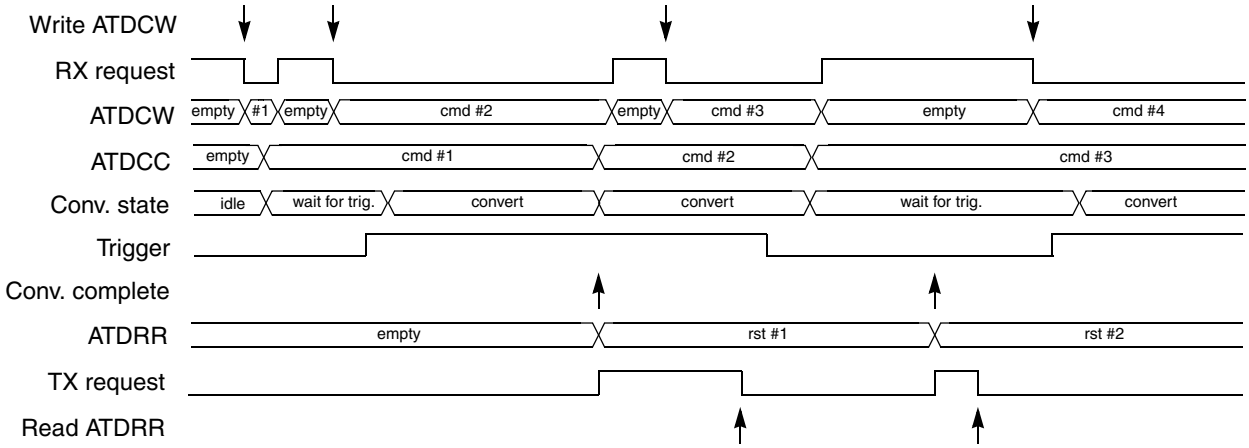


Figure 33-32. Conversion Procedure for Example 4

Like for the edge-triggered conversions the minimum trigger length is one system clock cycle. Although it is not recommended to use a level-sensitive trigger with such a short trigger pulse. Generally, a level-sensitive trigger is used to control how long conversions should be executed.

Due to synchronization and internal processing, the conversion start is delayed by 4 system clock cycles:

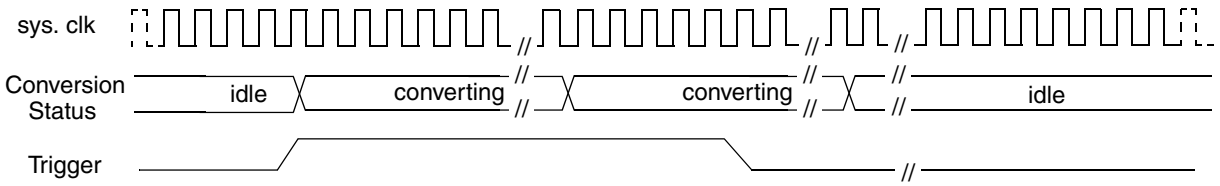


Figure 33-33. ATD Level-Based Minimum Trigger

33.12.3.6 Example 6: Queue Running Idle

This example shows a conversion sequence where the command words are not delivered in time and therefore the sequence stagnates. This can happen e.g. due to DMA performance issues.

In this sequence all command words use the same setting:

- CMD #*n*: CWCW = “continuous conversion”

The ATD will execute conversions as long as there is a command word in the ATDCC register. Command cmd #1 and cmd #2 are executed normally. As after cmd #2 there is no new command word available the ATD will wait until cmd #3 arrives. Once cmd #3 was written to ATDCW and pushed to ATDCC the conversion sequence can continue.

Figure 33-34 shows the described sequence:

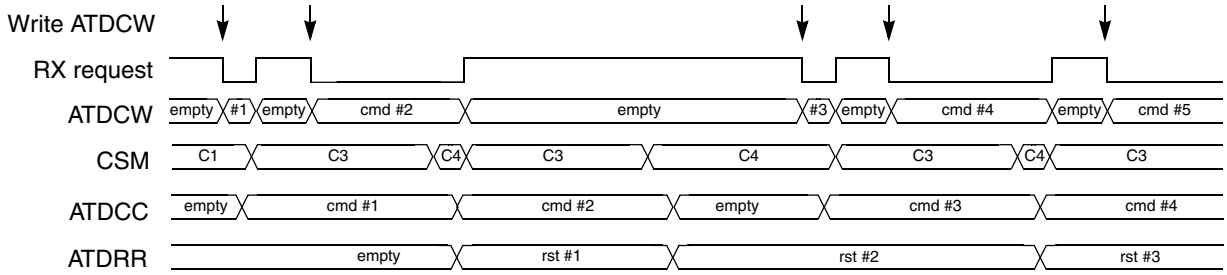


Figure 33-34. Conversion Procedure for Example 6

33.12.3.7 Example 7: Entering Low-power Mode During a Conversion

This example shows a conversion sequence which gets interrupted as a low-power mode is entered. In this sequence all command words use the same setting:

- CMD #*n*: CWCW = “continuous conversion”

The first conversion is started as usual: there is a DMA request for a command word and therefore the DMA writes cmd #1 to ATDCW which is pushed to ATDCC. At the end of the conversion the result is pushed to ATDRR. After this the second conversion starts (cmd #2).

During the second conversion the system enters low-power mode (LP-mode). The conversion is terminated immediately and the command word cmd #3 in ATDCW is discarded. ATDCC and ATDCW are both empty. Nevertheless no DMA request for a command word (RX request) is asserted.

The conversion result (rst #1), which has not been read before low-power mode was entered, is still available in the result register and therefore the DMA request (TX request) remains asserted.

After low-power mode is left a request for a new command word is immediately asserted. But as the analog conversion circuit was disabled during low-power mode no new conversions are allowed to start until the recovery time t_{REC} elapsed. This is why cmd #4 is written to ATDCW after t_{REC} elapsed. The conversions started after that time (cmd #4 and cmd #5) are executed normally.

In this example the conversion result rst #1 is read after low-power mode was left. This was done to illustrate that the TX request is not affected by the low-power mode. It would have been possible to read ATDRR before entering low-power mode.

To keep the togetherness of command and result the result which belongs to cmd #4 is named rst #4 although it is the second conversion result in this example.

Figure 33-35 shows the described sequence:

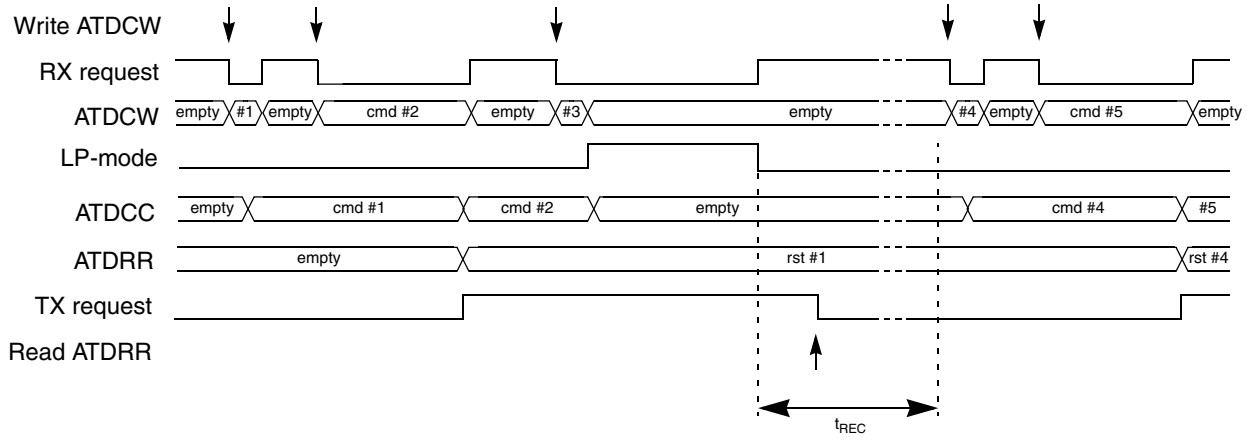


Figure 33-35. Conversion Procedure for Example 7

33.12.3.8 Example 8: Debug Mode

This example shows a conversion sequence during Debug mode. The sequence will be frozen after a conversion finished (“convert then freeze”) and is resumed by leaving Doze mode. In this sequence all command words use the same setting:

- CMD #*n*: CWCW = “continuous conversion”

The first conversion is started as usual: there is a DMA request for a command word and therefore the DMA writes cmd #1 to ATDCW which is pushed to ATDCC. At the end of the conversion the result is pushed to ATDRR. After this the second conversion starts (cmd #2).

During the second conversion the system enters Debug mode. The second conversion will execute normally until it finishes. As the system is in Debug mode and “convert then freeze” was set the next command word (cmd #3) is not pushed to ATDCC. It will remain in ATDCW until Debug mode is left.

Once the system left Debug mode cmd #3 is pushed from ATDCW to ATDCC and the conversion sequence will continue:

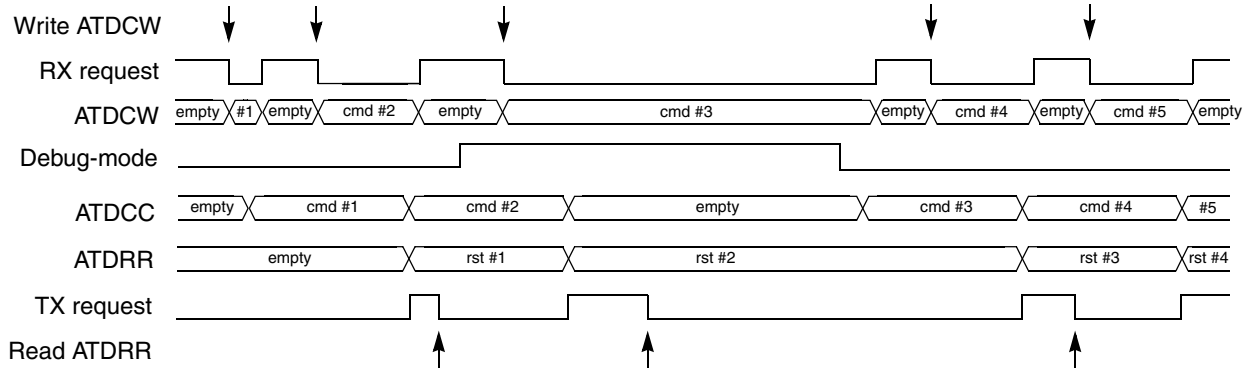


Figure 33-36. Conversion Procedure for Example 8 (part 1)

The following figure shows a similar conversion sequence to that shown in Figure 33-36. Also here Debug mode is entered during the second conversion. But this time another conversion is started during Debug mode by writing the ATDCW register:

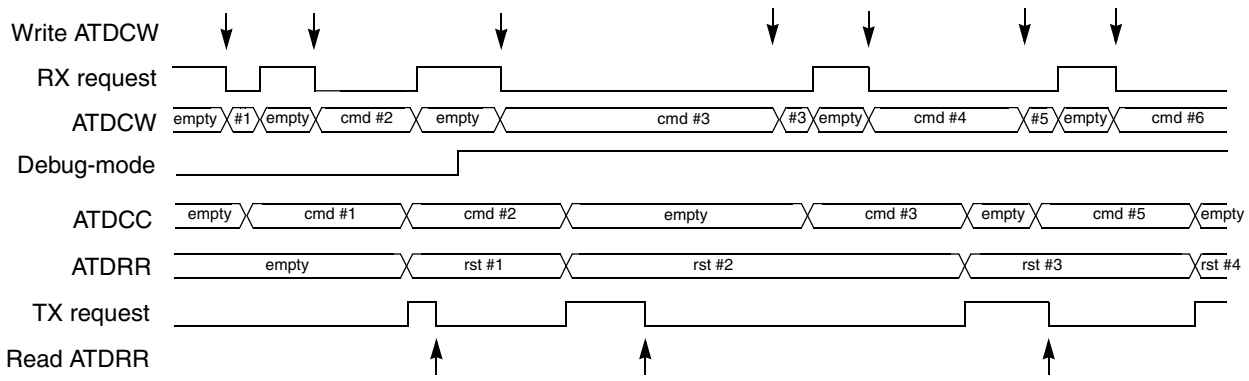


Figure 33-37. Conversion Procedure for Example 8 (part 2)

After the second conversion finished cmd #3 is not pushed to ATDCC as Debug mode is active. No RX request for a new command word is asserted as ATDCW is still full. Therefore ATDCC stays empty.

To force the ATD to execute another conversion a new command word must be written to ATDCW. In this case cmd #3 is written again to ATDCW. This write access will now start a new conversion using cmd #3. An RX request for a new command word is asserted until a new command word is available (cmd #4).

Once the third conversion finishes ATDCC will run empty. Again it is necessary to write another command word to ATDCW to start a new conversion. Instead of re-writing cmd #4 to ATDCW this time a new command (cmd #5) is written to ATDCW. Therefore cmd #4 is overwritten and the ATD will start the conversion using cmd #5.

After the start ATDCW is empty and therefore another command word is requested. This procedure can be repeated.

33.12.3.9 Conversion Mechanism (bits CWCH, CWNF, CWGI, CWSC, CWAR)

In Section 33.11.5.2, “Command Processing and DMA Requests,” and Section 33.12.3.4, “Example 4: Edge-triggered Conversion,” it has been described how conversions are started and what is done after a conversion finished. This section describes the ATD conversion sequence as well as the remaining bits in a command word that have not been described previously.

Table 33-22. Bit Description of the Command Word

Bit name	Function
CWCH	Analog channel that must be sampled
CWNF	Numeric format in which the conversion result is saved
CWSL	Sample length (different sample lengths)
CWCM	Conversion mode (see Section 33.11.5.2, “Command Processing and DMA Requests,” and Section 33.12.3.4, “Example 4: Edge-triggered Conversion”)
CWGI	‘1’ = Set the CC bit in the ATDFLAG register after a conversion finished

Table 33-22. Bit Description of the Command Word (Continued)

Bit name	Function
CWSC	'1' = Sample Special Channels
CWAR	'1' = Adjust Result

Each conversion must define which analog channel needs to be sampled. The CWCH bits in the command word should be programmed with a value from 0 to 31 to identify the appropriate channel number to be sampled. If the CWSC bit is set then fixed reference voltages can be sampled (see Section 33.10.2.9, "ATD Command Word Register (ATDCW)").

When a conversion has been completed, the result is saved using one out of the following numeric formats: right- justified signed, right-justified unsigned, left-justified signed and left justified unsigned. All of them are available in four different resolutions (see Section 33.10.2.9, "ATD Command Word Register (ATDCW)"). The default format is right-justified unsigned 8 bit.

If the CWGI bit is set the Conversion Complete flag (CC) in the ATDCTL register will be set after a conversion finished. Otherwise the flag will not be updated after the conversion finished. This can be used to mark the end of a conversion sequence.

E.g. a conversion complete interrupt should be set after 8 conversions were done:

- CMD #1 .. CMD #7: CWGI = 0
- CMD #8: CWGI = 1
- ATDINT.CCIE = 1

The CC flag will remain unchanged after the first seven command words were executed. After the eight conversions it will get set. If the conversion complete interrupt is enabled an interrupt is generated after the last conversion finished.

The CWSL bits determine how long the sampling phase should be. It depends on the channel impedance how long the sampling must last. Usually the minimum value should be enough.

The CWAR bit determines whether the raw conversion result or an adjusted result should be used (See 33.12.2 ATD Calibration / Result Adjustment).

The CWCM bits are already explained in Section 33.11.5.2, "Command Processing and DMA Requests," and Section 33.12.3.4, "Example 4: Edge-triggered Conversion."

33.12.4 Reset

At reset the DMADC1032 is in Disabled mode (see Section 33.11.4, "Low Power / Operating Modes"). For the reset state of each individual bit see Section 33.10.2, "Register Descriptions."

33.12.5 Interrupts

The following table provides an overview of the possible ATD interrupt sources. All interrupt flags are located in the ATDFLAG register, with their enable bits located in the ATDINT register.

Table 33-23. ATD Interrupt Vectors

INT#	Interrupt Source	Interrupt flag	Enable bit	Other enables
1	Conversion complete	CC	CCIE	CWGI
2	Conversion paused	CP	CPIE	—
3	External trigger overrun	ETO	ETOIE	—
4	Conversion result lost	CRL	CRLIE	—
5	Command queue empty	CQE	CQEIE	—
6	Command queue full	CQF	CQFIE	—
7	Command queue not full	CQNF	CQNFIE	—

Interrupt #1 is the only source that has an additional enable bit. Both the CWGI bit in the command word and the CCIE bit in the ATDINT register must be set in order that the CC flag can generate an interrupt on the completion of a conversion.

With the ATD module operating in Normal mode, the CC, CP, ETO and CRL flags will be cleared by writing a ‘1’ to the flag. The CQF, CQE and CQNF flags will not be cleared as these represent the current state of the command word.

- Interrupt #1: generated after a conversion finished and CWGI = 1
- Interrupt #2: generated after a conversion finished and ATDCC[CWCM]= “convert then pause”
- Interrupt #3: generated when additional edges at the trigger input have been recognized but no triggers were expected
- Interrupt #4: generated when a conversion result was overwritten before it could be read
- Interrupt #5: generated when the command queue is empty
- Interrupt #6: generated when the command queue is full
- Interrupt #7: generated when the command queue can store another command word

All interrupts share the same interrupt line. Therefore the flag register must be read to identify the source of an ATD interrupt.

Chapter 34
Port Integration Module (PIM_MAC7202)

34.1 Introduction

34.1.1 Overview

The Port Integration Module implements the interfaces between the peripheral modules and the I/O pins, as well as the GPIO functionality. Figure 34-1 shows the block diagram of this module. The MAC7200 family implements up to seven 16-bit ports. Each pin on the device can be independently configured as follows:

- Pin functionality (GPIO, interrupt input or peripheral)
- Port direction (GPIO mode only)
- Pull-down or Pull-up enable
- Open drain enable
- Slew rate control

When selected to be in Peripheral mode, both the pin's output and the pin's configuration are controlled by the associated peripheral. Figure 34-1 shows the allocation of the different peripherals to the ports.

Table 34-1. Port Pin and Peripheral Allocation

Port	Peripheral
Port B[0:15]	IIC, DSPI_A, DSPI_B
Port D[2]	GPIO (PIM) Clock
Port D[3:4]	Interrupt Controller
Port E[0:15]	ATD_A
Port F[0:15]	eMIOS, DSPI_C
Port G[0:15]	SCI_A, SCI_B, CAN_A, CAN_B

To assist in software development, all pins can be controlled either in a port-wise or bit-wise manner. Registers are provided which allow each pin to be independently controlled by writing to a separate register, but mirror registers are also available to simplify read or write operations by accessing all pins in a port at the same time.

Each pin is also capable of being used as interrupt source and a pin filter is available to help eliminate glitches that may generate false interrupts. Like all other interrupt sources in the system, the Port interrupts may also be used to wakeup the system from low power modes.

Following a Reset, all GPIO pins are configured as General Purpose Input (GPI) pins, except in Expanded Mode, where the following pins are automatically configured to Peripheral Mode:

Table 34-2. Expanded Mode Startup Pin Configuration

Port	Configuration
Pin A[0:15]	DATA0 to DATA15
Pin C[0:15]	ADDR0 to ADDR15
Pin D[0:2]	BS0, BS1, CLKOUT
Pin D[5:15]	ADDR16 to ADDR21, OE, BURST, TA, CS0, R/W

NOTE

General Purpose Output functionality is not available on pin PD2 (CLKOUT) or on Port E. Configuring this pin to be in Peripheral Mode or in GPO Mode offers virtually the same functionality (CLKOUT). Configuring this pin to be in GPI Mode disables CLKOUT, and the pin behaves like a standard GPI pin.

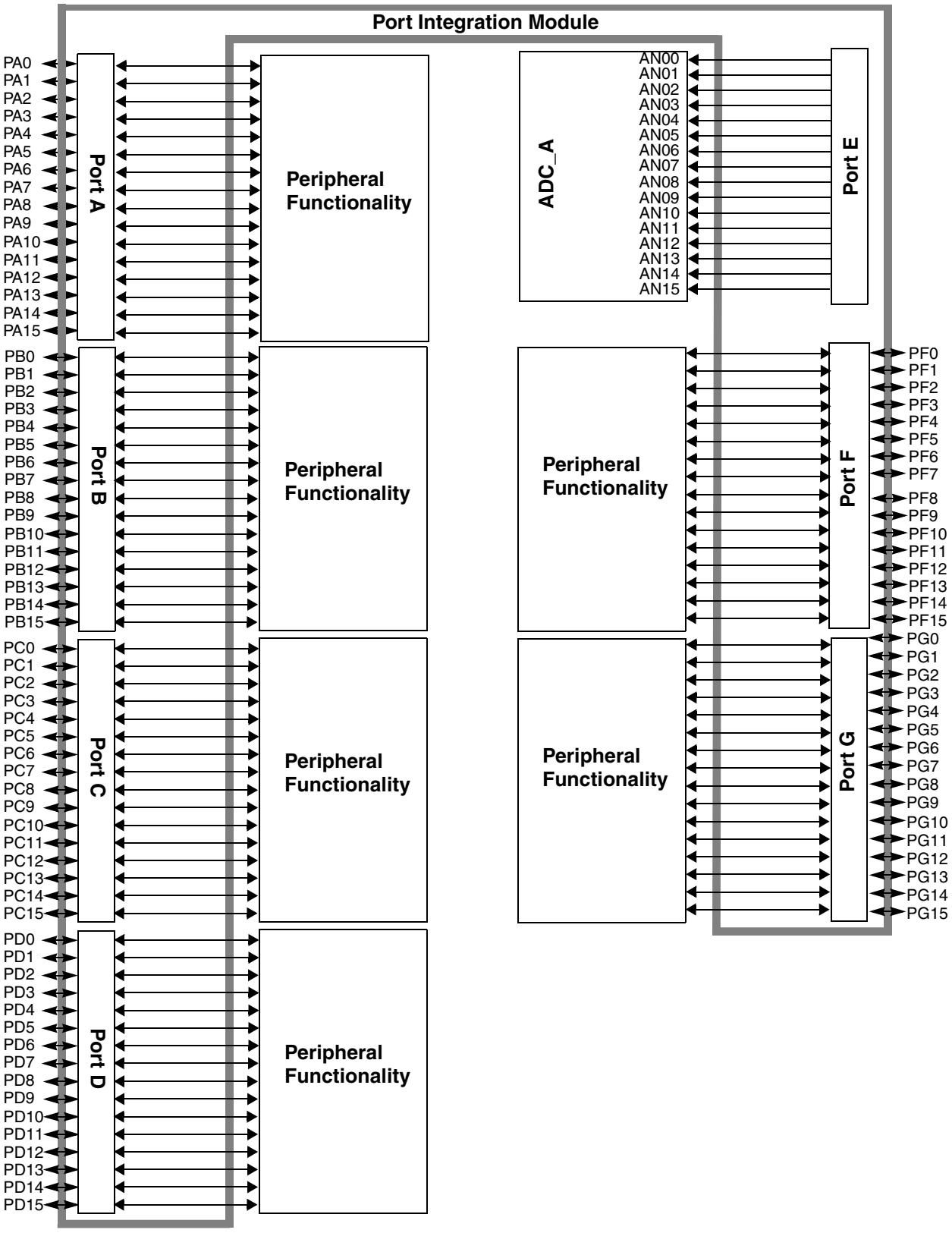


Figure 34-1. Port Integration Module Block Diagram

34.1.2 Features

The Port Integration Module includes these distinctive features:

- Select between GPIO, interrupt or peripheral functionality on a pin-by-pin basis
- Register interface for both port wide and pin data reads and writes
- 6 independent 16-bit ports (Port A, B, C, D, F, G), with each pin having the following features:
  - Peripheral or GPIO Mode selection, with up to 3 peripherals per pin
  - Input/Output selection
  - 5V output drive with three selectable slew rates (Disabled, Slow, Fast)
  - 5V digital inputs
  - Selectable pull-up or pull-down
  - Selectable open drain for wired-or connections
  - Selectable interrupt capability (with glitch filtering and interrupt mask)
- Control of ATD digital inputs (Port E)
  - Selectable interrupt capability (with glitch filtering and interrupt mask)
- Control of the TCK, TMS, TDI and TDO pads
  - Input/Output selection
  - 5V output drive with three selectable slew rates (Disabled, Slow, Fast) (TDO only)
  - 5V digital inputs
  - Selectable pull-up or pull-down (TCK, TMS and TDI only)

34.1.3 Modes of Operation

34.1.3.1 Peripheral Mode (CONFIG::MODE[1:0] ≠ 0 0)

In this mode, the peripheral drives all data and control signals to the pad, with the exception of the Slew Rate (SRE1/SRE0) signals, which are always driven by the **SLEWDIS** and **SLEW** bits in the corresponding **CONFIGxx** register in the PIM.

Table 34-3. PIM Peripheral Mode Configuration

Register / Field	Definition
PORTDATA/PINDATAxx	This value has no effect.
PORTIR	This read-only register returns the value currently driven on the pin by the peripheral (output) or by an external device (input).
DDR	This register bit has no effect.
SLEWDIS/SLEW	Slew Rate control bits: Controls the slew rate of the pin (output only).
SLEWDIS = 0	Slew Rate control is disabled
SLEW	0 = Fast slew rate, 1 = Slow slew rate
PULL	These two register bits have no effect.

34.1.3.2 GPIO Output Mode (CONFIG::MODE[1:0] = 0 0, CONFIG::DDR = 1)

In this mode, also called GPO Mode, the pin is used as a General Purpose Output.

Table 34-4. PIM GPIO Mode Configuration

Register / Field	Definition
PORTDATA/PINDATAxx	This value is driven out the pin
PORTIR	This read-only register returns the value currently driven on the pin by the DATA register.
SLEWDIS/SLEW	Slew Rate control bits: Controls the slew rate of the pin (output only).
SLEWDIS = 0	Slew Rate control is disabled
SLEW	0 = Fast slew rate, 1 = Slow slew rate
PULL	These two register bits selects a Pullup or Pulldown on the input, according to the following table: 00 None 01 None 10 Pulldown 11 Pullup

34.1.3.3 GPIO Input Mode (CONFIG::MODE[1:0] = 0 0, CONFIG::DDR = 0)

In this mode, also called GPI Mode, the pin is used as a General Purpose Input.

Table 34-5. PIM GPIO Input Mode Configuration

Register / Field	Definition
PORTDATA/PINDATAxx	This value has no effect.
PORTIR	This read-only register returns the value currently driven on the pin by an external device.
SLEWDIS/SLEW	Slew Rate control bits: Controls the slew rate of the pin (output only).
SLEWDIS = 0	Slew Rate control is disabled
SLEW	0 = Fast slew rate, 1 = Slow slew rate
PULL	These two register bits selects a Pullup or Pulldown on the input, according to the following table: 00 None 01 None 10 Pulldown 11 Pullup

In GPI mode, it is also possible to use the input as an external interrupt by programming the **PIER** and **PIFR** bits accordingly. When in this mode, the input signals are passed through a glitch filter (running at Peripheral Bus Clock speed) to avoid spurious external interrupts. The filter is designed so that it must see 4 clock cycles of a negated (passive) signal (setup time), followed by 4 clock cycles of an asserted (active) signal (hold time) to capture the active edge, and be flagged as an interrupt.

34.2 External Signal Description

For detailed descriptions of a particular pin or pins, please refer to the specific peripheral's documentation.

Table 34-6. Port Pin to Primary Peripheral Function Assignments

Pin [1, 2]	Primary Peripheral Function
PA0	FlexBus DATA[0]
PA1	FlexBus DATA[1]
PA2	FlexBus DATA[2]
PA3	FlexBus DATA[3]
PA4	FlexBus DATA[4]
PA5	FlexBus DATA[5]
PA6	FlexBus DATA[6]
PA7	FlexBus DATA[7]
PA8	FlexBus DATA[8]
PA9	FlexBus DATA[9]
PA10	FlexBus DATA[10]
PA11	FlexBus DATA[11]
PA12	FlexBus DATA[12]
PA13	FlexBus DATA[13]
PA14	FlexBus DATA[14]
PA15	FlexBus DATA[15]
PB0	IIC SDA [3]
PB1	IIC SCL [3]
PB2	DSPI_A MISO [4]
PB3	DSPI_A MOSI [4]
PB4	DSPI_A SCK [4]
PB5	DSPI_A CS0/SS [4]
PB6	DSPI_A CS1 [4]
PB7	DSPI_A CS2 [4]
PB8	DSPI_A CS5/PCSS [4]
PB9	DSPI_B CS0/SS [4]
PB10	DSPI_B CS5/PCSS [4]
PB11	DSPI_B CS2 [4]
PB12	DSPI_B CS1 [4]
PB13	DSPI_B SCK [4]
PB14	DSPI_B MOSI [4]
PB15	DSPI_B MISO [4]
PC0	FlexBus ADDR[0]
PC1	FlexBus ADDR[1]
PC2	FlexBus ADDR[2]
PC3	FlexBus ADDR[3]
PC4	FlexBus ADDR[4]
PC5	FlexBus ADDR[5]
PC6	FlexBus ADDR[6]
PC7	FlexBus ADDR[7]
PC8	FlexBus ADDR[8]
PC9	FlexBus ADDR[9]

Table 34-6. Port Pin to Primary Peripheral Function Assignments (Continued)

Pin [1, 2]	Primary Peripheral Function
PC10	FlexBus ADDR[10]
PC11	FlexBus ADDR[11]
PC12	FlexBus ADDR[12]
PC13	FlexBus ADDR[13]
PC14	FlexBus ADDR[14]
PC15	FlexBus ADDR[15]
PD0	FlexBus BWE[0]
PD1	FlexBus BWE[1]
PD2 [5]	CLKOUT
PD3	XIRQ/NMI
PD4	IRQ
PD5	FlexBus ADDR[16]
PD6	FlexBus ADDR[17]
PD7	FlexBus ADDR[20]
PD8	FlexBus ADDR[21]
PD9	FlexBus ADDR[18]
PD10	FlexBus ADDR[19]
PD11	FlexBus OE
PD12	FlexBus BURST
PD13	FlexBus TA
PD14	FlexBus CS0
PD15	FlexBus RW
PE0	ADC_A Channel 00
PE1	ADC_A Channel 01
PE2	ADC_A Channel 02
PE3	ADC_A Channel 03
PE4	ADC_A Channel 04
PE5	ADC_A Channel 05
PE6	ADC_A Channel 06
PE7	ADC_A Channel 07
PE8	ADC_A Channel 08
PE9	ADC_A Channel 09
PE10	ADC_A Channel 10
PE11	ADC_A Channel 11
PE12	ADC_A Channel 12
PE13	ADC_A Channel 13
PE14	ADC_A Channel 14
PE15	ADC_A Channel 15
PF0	eMIOS Channel 00 [4]
PF1	eMIOS Channel 01 [4]
PF2	eMIOS Channel 02 [4]
PF3	eMIOS Channel 03 [4]

Table 34-6. Port Pin to Primary Peripheral Function Assignments (Continued)

Pin [1, 2]	Primary Peripheral Function
PF4	eMIOS Channel 04 [4]
PF5	eMIOS Channel 05 [4]
PF6	eMIOS Channel 06 [4]
PF7	eMIOS Channel 07 [4]
PF8	DSPI_C CS5/PCSS
PF9	DSPI_C CS3
PF10	DSPI_C CS2
PF11	DSPI_C SCK
PF12	DSPI_C CS1
PF13	DSPI_C MOSI
PF14	DSPI_C CS0/SS
PF15	DSPI_C MISO
PG0	SCI_B RXD
PG1	SCI_B TXD
PG2	SCI_A RXD
PG3	SCI_A TXD
PG4	CAN_A TXD
PG5	CAN_A RXD
PG6	CAN_B TXD
PG7	CAN_B RXD
PG8	FlexBus CS1
PG9	FlexBus CS2
PG10	None
PG11	None
PG12	DSPI_A CS4
PG13	DSPI_A CS3
PG14	DSPI_B CS4
PG15	DSPI_B CS3

1. At reset, all Ports are in GPIO mode, configured as an input with no pull-up/pull-down and open drain disabled, except for pin PD2.
2. For those pins with no Peripheral Mode functionality, placing the pin into Peripheral Mode will force the pin to become an output with a low (0) state driven out.
3. When IIC SDA or SCL functionality is selected on a pin, Open Drain functionality is enabled for that pin.
4. Pin direction may be changed dynamically by the peripheral for the following functionality: DSPI CS0/SS, DSPI CS5/PCSS, DSPI SCK, eMIOS (all channels).
5. Pin PD2 has no GPO functionality.

NOTES

Table 34-7. JTAG Pin Functions (Peripheral Mode)

Port	Primary Peripheral Function [1]	Secondary Peripheral Function	Tertiary Peripheral Function
TCK	JTAG Interface TCK	None	None
TMS	JTAG Interface TMS	None	None
TDI	JTAG Interface TDI	None	None
TDO	JTAG Interface TDO	None	None

1. The JTAG interface has pull-ups/pull-downs enabled at reset, as follows:
   TCK: Pull-down
   TMS, TDI: Pull-up
   TDO: None

34.2.1 Port A

This port implements 16 GPIO/Peripheral pins. All 16-bits are controllable as GPIO, and the port is controlled as follows:

- PA0-PA15: Switched by the **MODE** bits in Port A of the PIM
- Refer to the PIM Block Guide for the Peripheral Mode functionality of this port

The GPIO registers for this port reside at an offset of $0000 in the PIM memory map slot.

34.2.2 Port B

This port implements 16 GPIO/Peripheral pins. All 16-bits are controllable as GPIO, and the port is controlled as follows:

- PB0-PB15: Switched by the **MODE** bits in Port B of the PIM
- Refer to the PIM Block Guide for the Peripheral Mode functionality of this port

The GPIO registers for this port reside at an offset of $0040 in the PIM memory map slot.

34.2.3 Port C

This port implements 16 GPIO/Peripheral pins. All 16-bits are controllable as GPIO, and the port is controlled as follows:

- PC0-PC15: Switched by the **MODE** bits in Port C of the PIM
- Refer to the PIM Block Guide for the Peripheral Mode functionality of this port

The GPIO registers for this port reside at an offset of $0080 in the PIM memory map slot.

34.2.4 Port D

This port implements 15 GPIO/Peripheral + 1 GPI/Peripheral pins. All 16-bits are controllable as GPIO, and the port is controlled as follows:

- PD0-PD15: Switched by the **MODE** bits in Port D of the PIM
- PD2: Always an output (CLKOUT) in Peripheral Mode or in GPO Mode (i.e.-same functionality). PD2 may also be used as a GPI. Refer to Section 17.7.6, "Enabling and Disabling CLKOUT" for more details. In addition this pin is used to sample $\overline{\text{XCLKS}}$.
- Refer to the PIM Block Guide for the Peripheral Mode functionality of this port

The GPIO registers for this port reside at an offset of $00C0 in the PIM memory map slot.

34.2.5 Port E

This port implements 16 GPI/Peripheral pins. All 16-bits are controllable as GPI, and the port is controlled as follows:

- PE0-PE15: Switched by the **MODE** bits in Port E of the PIM
- Refer to the PIM Block Guide for the Peripheral Mode functionality of this port
- When in Peripheral Mode, a Port E pin may also be used as an external ATD trigger
- No GPO functionality is available on this port

The GPI registers for this port reside at an offset of $0100 in the PIM memory map slot.

34.2.6 Port F

This port implements 16 GPIO/Peripheral pins. All 16-bits are controllable as GPIO, and the port is controlled as follows:

- PF0-PF15: Switched by the **MODE** bits in Port F of the PIM
- Refer to the PIM Block Guide for the Peripheral Mode functionality of this port

The GPIO registers for this port reside at an offset of $0140 in the GPIO memory map slot.

34.2.7 Port G

This port implements 16 GPIO/Peripheral pins. All 16-bits are controllable as GPIO, and the port is controlled as follows:

- PG0-PG15: Switched by the **MODE** bits in Port G of the PIM
- Refer to the PIM Block Guide for the Peripheral Mode functionality of this port

The GPIO registers for this port reside at an offset of $0180 in the GPIO memory map slot.

The IIC SDA and SCL pins have the following special settings in Peripheral Mode: ODE=1, IBE=1 (i.e.-The IIC module requires both SDA and SCL to be open-drain with the input Schmidt trigger enabled).

XIRQ and IRQ have the following special settings in Peripheral Mode: OBE=0, ODE=0, DSE=0, PUS=1.

34.3 PIM Bus Aborts

The PIM module supports Peripheral Bus bus aborts, and enforces the following memory map:

Table 34-8. PIM Bus Aborts

Abort	Allowed
	$0000-$01bf
$01c0-$03bf	
	$03c0-$03cf
$03d0-$03df	
	$03e0-$03ff
$0400-$3fff	

Supervisor Access: Unused.

NOTE

Some addresses in the allowed area are not implemented. For these addresses, READs return $00, and WRITEs are ignored.

34.4 PIM Differences from MAC71xx

- Removed port H and associated registers
- Removed Port E GPO and associated register bits
- Removed RC oscillator
  - Can not wakeup from STOP mode
  - Glitch filter only runs from peripheral (slow) clock
- Fixed MUCts01527: Flash programming overwrites PIM registers (Was fixed on MAC71x1 1L47W)
- Enabled 32-bit **DPORTIR** register access in non-test mode
- Changed from Drive Strength to Slew Rate control. Added slew rate control sub-divider bit
- Added 3 levels of peripheral muxing (**CONFIG[8]**)

34.5 PIM Application Usage

34.5.1 Enabling the PIM

It is not necessary to enable the PIM before it can be used.

34.5.2 Using Single Pins in a Port

There is no limitation on using single pins in a port. Any combination of GPI, GPO and Peripheral Modes may be used in a port. It is even possible to use pins in GPI or GPO mode to mimic slower rate peripheral protocols manually.

34.5.3 Pin versus Port Registers

Table 34-9 summarizes the PIM registers, showing which registers are intended for pin access, which registers are intended for port-wide access, and which registers are intended for double port-wide (32 pins) access.

Table 34-9. PIM Registers

Addr Offset	Register	Read/ Write	Pin	Port	Dbl Port	Register Name	Function
$000 to $01E	CONFIGxx	R/W	X			Pin Config	Configures a single pin MODE[1:0] - Peripheral/GPIO Mode DDR - Data Direction ODER - Open Drain Enable SLEWDIS/SLEW - Slew Rate PULL[1:0] - Pull-up/Pull-down PIER - Interrupt Enable PIFR - Interrupt Flag
$020	PORTIFR	R		X		Port Wide Interrupt Flag	Read-only port-wide mirror of the PIFR bits from each **CONFIGxx** register
$024	PORTDATA	R/W		X		Port Wide Data	Port-wide Data Read/Data Write
$026	PORTIR	R		X		Port Wide Input	The value currently driven on the port
$028 to $037	PINDATAxx	R/W	X			Pin Data	Pin-wide mirror of the **PORTDATA** register. Useful for reading/writing small numbers of pins on a port, especially using a DMA.
$3e0	PORT32IR	R			X	Double Port Wide Input	Combines two PORTIR registers into a single 32-bit wide register.

34.5.4 Peripheral Muxing

Figure 34-2 shows an overview of the four levels of functional muxing available on most pins:

- GPIO (with optional interrupt capability)
- Primary Peripheral (#1)
- Secondary Peripheral (#2)
- Tertiary Peripheral (#3)

Please refer to the PIM Block Guide for a full explanation of using the mode selection for each pin, and for a complete listing of the functionality of each pin in Primary, Secondary and Tertiary peripheral modes.

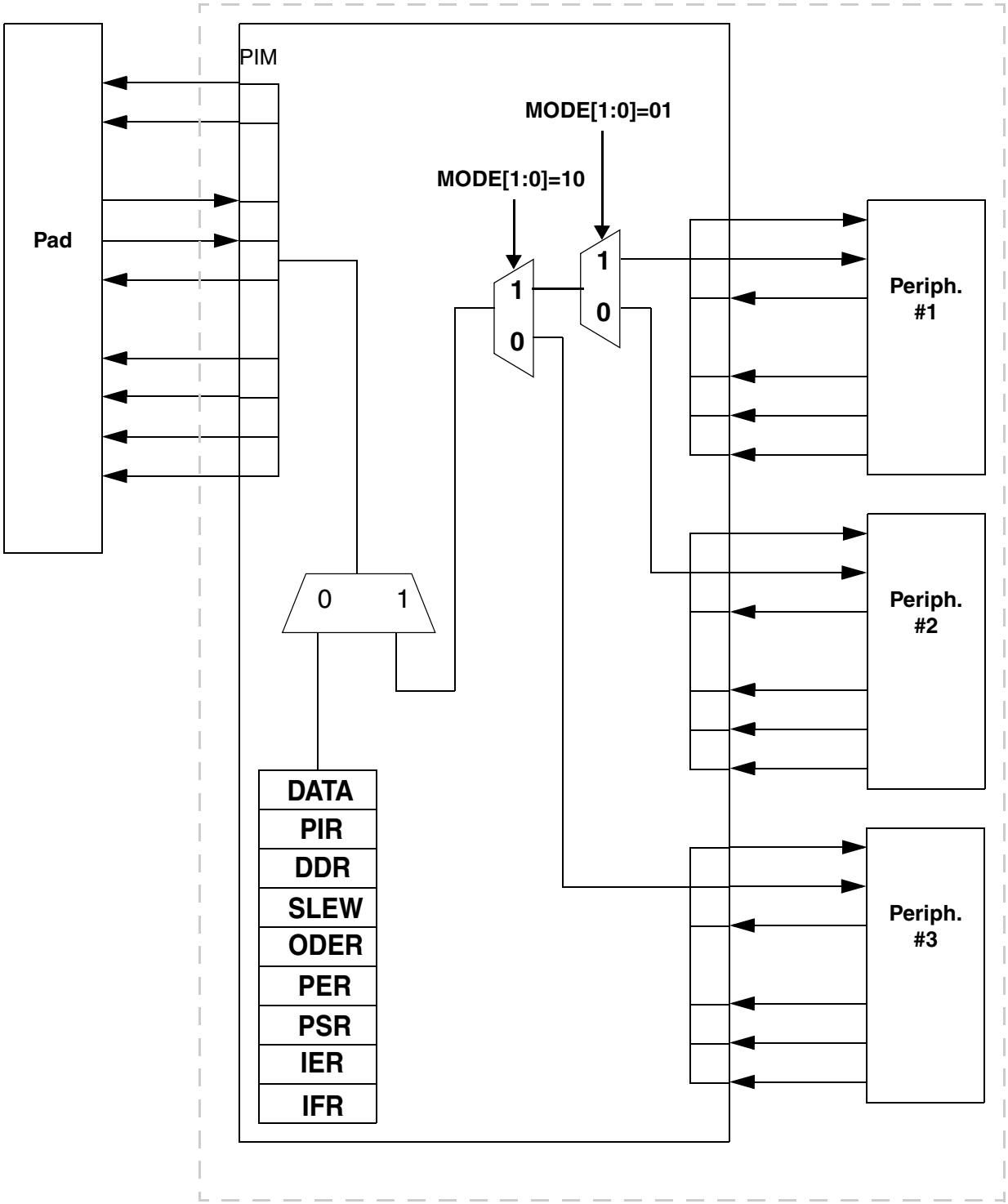


Figure 34-2. PIM Peripheral Muxing

34.6 Memory Map and Register Definition

This section provides a detailed description of all memory-mapped registers in the Port Integration Module.

Table 34-11 shows the memory map for the Port Integration Module. Note that all addresses are offsets; the absolute address may be computed by adding the specified offset to the base address of the Port Integration Module.

Table 34-10. Port Integration Module Memory Map Overview

Address	Port
Base + 0x0000	Port A
Base + 0x0040	Port B
Base + 0x0080	Port C
Base + 0x00C0	Port D
Base + 0x0100	Port E
Base + 0x0140	Port F
Base + 0x0180	Port G
Base + 0x01C0 – Base + 0x03BF	Reserved [1]
Base + 0x03C0	Global Interrupt Status
Base + 0x03C2	PIM Configuration
Base + 0x03C4	TDI Pad Control
Base + 0x03C6	TDO Pad Control
Base + 0x03C8	TMS Pad Control
Base + 0x03CA	TCK Pad Control
Base + 0x03CC	Reserved
Base + 0x03CE	RESET Pad Control
Base + 0x03D0 – Base + 0x03DF	Reserved[1]
Base + 0x03E0 – Base + 0x03FF	PORT32IR Registers
Base + 0x0400 – Base + 0x3FFF	Reserved[1]

1. Registers are marked as reserved. If enabled at the SoC level, accessing these registers will cause bus aborts. Refer to the System Services Module documentation for more details.

NOTE

Not all Ports are available on all packages.

Table 34-11. Port Integration Module Memory Map

	Address	Use	Size	Access	Mode [1]
Port A	Base + 0x000	Port A: Pin 0 Configuration (CONFIG0)	16	R/W	A
	Base + 0x002	Port A: Pin 1 Configuration (CONFIG1)	16	R/W	A
	Base + 0x004	Port A: Pin 2 Configuration (CONFIG2)	16	R/W	A
	Base + 0x006	Port A: Pin 3 Configuration (CONFIG3)	16	R/W	A
	Base + 0x008	Port A: Pin 4 Configuration (CONFIG4)	16	R/W	A
	Base + 0x00A	Port A: Pin 5 Configuration (CONFIG5)	16	R/W	A
	Base + 0x00C	Port A: Pin 6 Configuration (CONFIG6)	16	R/W	A
	Base + 0x00E	Port A: Pin 7 Configuration (CONFIG7)	16	R/W	A
	Base + 0x010	Port A: Pin 8 Configuration (CONFIG8)	16	R/W	A
	Base + 0x012	Port A: Pin 9 Configuration (CONFIG9)	16	R/W	A
	Base + 0x014	Port A: Pin 10 Configuration (CONFIG10)	16	R/W	A
	Base + 0x016	Port A: Pin 11 Configuration (CONFIG11)	16	R/W	A
	Base + 0x018	Port A: Pin 12 Configuration (CONFIG12)	16	R/W	A
	Base + 0x01A	Port A: Pin 13 Configuration (CONFIG13)	16	R/W	A
	Base + 0x01C	Port A: Pin 14 Configuration (CONFIG14)	16	R/W	A
	Base + 0x01E	Port A: Pin 15 Configuration (CONFIG15)	16	R/W	A
	Base + 0x020	Port A: Port Wide Interrupt Flag (PORTIFR)	16	R/W	A
	Base + 0x022	Reserved	Writing this register will have no effect. Reading this register will return 0x00		
	Base + 0x024	Port A: Port Wide Data Read/Write (PORTDATA)	16	R/W	A
	Base + 0x026	Port A: Port Wide Input (PORTIR)	16	R	A
	Base + 0x028	Port A: Pin0 Data Read/Write (PINDATA0)	8	R/W	A
	Base + 0x029	Port A: Pin1 Data Read/Write (PINDATA1)	8	R/W	A
	Base + 0x02A	Port A: Pin2 Data Read/Write (PINDATA2)	8	R/W	A
	Base + 0x02B	Port A: Pin3 Data Read/Write (PINDATA3)	8	R/W	A
	Base + 0x02C	Port A: Pin4 Data Read/Write (PINDATA4)	8	R/W	A
	Base + 0x02D	Port A: Pin5 Data Read/Write (PINDATA5)	8	R/W	A
	Base + 0x02E	Port A: Pin6 Data Read/Write (PINDATA6)	8	R/W	A
	Base + 0x02F	Port A: Pin7 Data Read/Write (PINDATA7)	8	R/W	A
	Base + 0x030	Port A: Pin8 Data Read/Write (PINDATA8)	8	R/W	A
	Base + 0x031	Port A: Pin9 Data Read/Write (PINDATA9)	8	R/W	A
	Base + 0x032	Port A: Pin10 Data Read/Write (PINDATA10)	8	R/W	A
	Base + 0x033	Port A: Pin11 Data Read/Write (PINDATA11)	8	R/W	A
	Base + 0x034	Port A: Pin12 Data Read/Write (PINDATA12)	8	R/W	A
	Base + 0x035	Port A: Pin13 Data Read/Write (PINDATA13)	8	R/W	A
	Base + 0x036	Port A: Pin14 Data Read/Write (PINDATA14)	8	R/W	A
	Base + 0x037	Port A: Pin15 Data Read/Write (PINDATA15)	8	R/W	A
	Base + 0x038 to Base + 0x03F	Reserved	Writing this register will have no effect. Reading this register will return 0x00		

Table 34-11. Port Integration Module Memory Map (Continued)

	Address	Use	Size	Access	Mode [1]
Port B	Base + 0x040	Port B: Pin 0 Configuration (CONFIG0)	16	R/W	A
	Base + 0x042	Port B: Pin 1 Configuration (CONFIG1)	16	R/W	A
	Base + 0x044	Port B: Pin 2 Configuration (CONFIG2)	16	R/W	A
	Base + 0x046	Port B: Pin 3 Configuration (CONFIG3)	16	R/W	A
	Base + 0x048	Port B: Pin 4 Configuration (CONFIG4)	16	R/W	A
	Base + 0x04A	Port B: Pin 5 Configuration (CONFIG5)	16	R/W	A
	Base + 0x04C	Port B: Pin 6 Configuration (CONFIG6)	16	R/W	A
	Base + 0x04E	Port B: Pin 7 Configuration (CONFIG7)	16	R/W	A
	Base + 0x050	Port B: Pin 8 Configuration (CONFIG8)	16	R/W	A
	Base + 0x052	Port B: Pin 9 Configuration (CONFIG9)	16	R/W	A
	Base + 0x054	Port B: Pin 10 Configuration (CONFIG10)	16	R/W	A
	Base + 0x056	Port B: Pin 11 Configuration (CONFIG11)	16	R/W	A
	Base + 0x058	Port B: Pin 12 Configuration (CONFIG12)	16	R/W	A
	Base + 0x05A	Port B: Pin 13 Configuration (CONFIG13)	16	R/W	A
	Base + 0x05C	Port B: Pin 14 Configuration (CONFIG14)	16	R/W	A
	Base + 0x05E	Port B: Pin 15 Configuration (CONFIG15)	16	R/W	A
	Base + 0x060	Port B: Port Wide Interrupt Flag (PORTIFR)	16	R/W	A
	Base + 0x062	Reserved	Writing this register will have no effect. Reading this register will return 0x00		
	Base + 0x064	Port B: Port Wide Data Read/Write (PORTDATA)	16	R/W	A
	Base + 0x066	Port B: Port Wide Input (PORTIR)	16	R	A
	Base + 0x068	Port B: Pin0 Data Read/Write (PINDATA0)	8	R/W	A
	Base + 0x069	Port B: Pin1 Data Read/Write (PINDATA1)	8	R/W	A
	Base + 0x06A	Port B: Pin2 Data Read/Write (PINDATA2)	8	R/W	A
	Base + 0x06B	Port B: Pin3 Data Read/Write (PINDATA3)	8	R/W	A
	Base + 0x06C	Port B: Pin4 Data Read/Write (PINDATA4)	8	R/W	A
	Base + 0x06D	Port B: Pin5 Data Read/Write (PINDATA5)	8	R/W	A
	Base + 0x06E	Port B: Pin6 Data Read/Write (PINDATA6)	8	R/W	A
	Base + 0x06F	Port B: Pin7 Data Read/Write (PINDATA7)	8	R/W	A
	Base + 0x070	Port B: Pin8 Data Read/Write (PINDATA8)	8	R/W	A
	Base + 0x071	Port B: Pin9 Data Read/Write (PINDATA9)	8	R/W	A
	Base + 0x072	Port B: Pin10 Data Read/Write (PINDATA10)	8	R/W	A
	Base + 0x073	Port B: Pin11 Data Read/Write (PINDATA11)	8	R/W	A
	Base + 0x074	Port B: Pin12 Data Read/Write (PINDATA12)	8	R/W	A
	Base + 0x075	Port B: Pin13 Data Read/Write (PINDATA13)	8	R/W	A
	Base + 0x076	Port B: Pin14 Data Read/Write (PINDATA14)	8	R/W	A
	Base + 0x077	Port B: Pin15 Data Read/Write (PINDATA15)	8	R/W	A
	Base + 0x078 to Base + 0x07F	Reserved	Writing this register will have no effect. Reading this register will return 0x00		

Table 34-11. Port Integration Module Memory Map (Continued)

	Address	Use	Size	Access	Mode [1]
Port C	Base + 0x080	Port C: Pin 0 Configuration (CONFIG0)	16	R/W	A
	Base + 0x082	Port C: Pin 1 Configuration (CONFIG1)	16	R/W	A
	Base + 0x084	Port C: Pin 2 Configuration (CONFIG2)	16	R/W	A
	Base + 0x086	Port C: Pin 3 Configuration (CONFIG3)	16	R/W	A
	Base + 0x088	Port C: Pin 4 Configuration (CONFIG4)	16	R/W	A
	Base + 0x08A	Port C: Pin 5 Configuration (CONFIG5)	16	R/W	A
	Base + 0x08C	Port C: Pin 6 Configuration (CONFIG6)	16	R/W	A
	Base + 0x08E	Port C: Pin 7 Configuration (CONFIG7)	16	R/W	A
	Base + 0x090	Port C: Pin 8 Configuration (CONFIG8)	16	R/W	A
	Base + 0x092	Port C: Pin 9 Configuration (CONFIG9)	16	R/W	A
	Base + 0x094	Port C: Pin 10 Configuration (CONFIG10)	16	R/W	A
	Base + 0x096	Port C: Pin 11 Configuration (CONFIG11)	16	R/W	A
	Base + 0x098	Port C: Pin 12 Configuration (CONFIG12)	16	R/W	A
	Base + 0x09A	Port C: Pin 13 Configuration (CONFIG13)	16	R/W	A
	Base + 0x09C	Port C: Pin 14 Configuration (CONFIG14)	16	R/W	A
	Base + 0x09E	Port C: Pin 15 Configuration (CONFIG15)	16	R/W	A
	Base + 0x0A0	Port C: Port Wide Interrupt Flag (PORTIFR)	16	R/W	A
	Base + 0x0A2	Reserved	Writing this register will have no effect. Reading this register will return 0x00		
	Base + 0x0A4	Port C: Port Wide Data Read/Write (PORTDATA)	16	R/W	A
	Base + 0x0A6	Port C: Port Wide Input (PORTIR)	16	R	A
	Base + 0x0A8	Port C: Pin0 Data Read/Write (PINDATA0)	8	R/W	A
	Base + 0x0A9	Port C: Pin1 Data Read/Write (PINDATA1)	8	R/W	A
	Base + 0x0AA	Port C: Pin2 Data Read/Write (PINDATA2)	8	R/W	A
	Base + 0x0AB	Port C: Pin3 Data Read/Write (PINDATA3)	8	R/W	A
	Base + 0x0AC	Port C: Pin4 Data Read/Write (PINDATA4)	8	R/W	A
	Base + 0x0AD	Port C: Pin5 Data Read/Write (PINDATA5)	8	R/W	A
	Base + 0x0AE	Port C: Pin6 Data Read/Write (PINDATA6)	8	R/W	A
	Base + 0x0AF	Port C: Pin7 Data Read/Write (PINDATA7)	8	R/W	A
	Base + 0x0B0	Port C: Pin8 Data Read/Write (PINDATA8)	8	R/W	A
	Base + 0x0B1	Port C: Pin9 Data Read/Write (PINDATA9)	8	R/W	A
	Base + 0x0B2	Port C: Pin10 Data Read/Write (PINDATA10)	8	R/W	A
	Base + 0x0B3	Port C: Pin11 Data Read/Write (PINDATA11)	8	R/W	A
	Base + 0x0B4	Port C: Pin12 Data Read/Write (PINDATA12)	8	R/W	A
	Base + 0x0B5	Port C: Pin13 Data Read/Write (PINDATA13)	8	R/W	A
	Base + 0x0B6	Port C: Pin14 Data Read/Write (PINDATA14)	8	R/W	A
	Base + 0x0B7	Port C: Pin15 Data Read/Write (PINDATA15)	8	R/W	A
	Base + 0x0B8 to Base + 0x0BF	Reserved	Writing this register will have no effect. Reading this register will return 0x00		

Table 34-11. Port Integration Module Memory Map (Continued)

	Address	Use	Size	Access	Mode [1]
Port D	Base + 0x0C0	Port D: Pin 0 Configuration (CONFIG0)	16	R/W	A
	Base + 0x0C2	Port D: Pin 1 Configuration (CONFIG1)	16	R/W	A
	Base + 0x0C4	Port D: Pin 2 Configuration (CONFIG2)[2]	16	R/W	A
	Base + 0x0C6	Port D: Pin 3 Configuration (CONFIG3)	16	R/W	A
	Base + 0x0C8	Port D: Pin 4 Configuration (CONFIG4)	16	R/W	A
	Base + 0x0CA	Port D: Pin 5 Configuration (CONFIG5)	16	R/W	A
	Base + 0x0CC	Port D: Pin 6 Configuration (CONFIG6)	16	R/W	A
	Base + 0x0CE	Port D: Pin 7 Configuration (CONFIG7)	16	R/W	A
	Base + 0x0D0	Port D: Pin 8 Configuration (CONFIG8)	16	R/W	A
	Base + 0x0D2	Port D: Pin 9 Configuration (CONFIG9)	16	R/W	A
	Base + 0x0D4	Port D: Pin 10 Configuration (CONFIG10)	16	R/W	A
	Base + 0x0D6	Port D: Pin 11 Configuration (CONFIG11)	16	R/W	A
	Base + 0x0D8	Port D: Pin 12 Configuration (CONFIG12)	16	R/W	A
	Base + 0x0DA	Port D: Pin 13 Configuration (CONFIG13)	16	R/W	A
	Base + 0x0DC	Port D: Pin 14 Configuration (CONFIG14)	16	R/W	A
	Base + 0x0DE	Port D: Pin 15 Configuration (CONFIG15)	16	R/W	A
	Base + 0x0E0	Port D: Port Wide Interrupt Flag (PORTIFR)	16	R/W	A
	Base + 0x0E2	Reserved	Writing this register will have no effect. Reading this register will return 0x00		
	Base + 0x0E4	Port D: Port Wide Data Read/Write (PORTDATA)	16	R/W	A
	Base + 0x0E6	Port D: Port Wide Input (PORTIR)	16	R	A
	Base + 0x0E8	Port D: Pin0 Data Read/Write (PINDATA0)	8	R/W	A
	Base + 0x0E9	Port D: Pin1 Data Read/Write (PINDATA1)	8	R/W	A
	Base + 0x0EA	Port D: Pin2 Data Read/Write (PINDATA2)	8	R/W	A
	Base + 0x0EB	Port D: Pin3 Data Read/Write (PINDATA3)	8	R/W	A
	Base + 0x0EC	Port D: Pin4 Data Read/Write (PINDATA4)	8	R/W	A
	Base + 0x0ED	Port D: Pin5 Data Read/Write (PINDATA5)	8	R/W	A
	Base + 0x0EE	Port D: Pin6 Data Read/Write (PINDATA6)	8	R/W	A
	Base + 0x0EF	Port D: Pin7 Data Read/Write (PINDATA7)	8	R/W	A
	Base + 0x0F0	Port D: Pin8 Data Read/Write (PINDATA8)	8	R/W	A
	Base + 0x0F1	Port D: Pin9 Data Read/Write (PINDATA9)	8	R/W	A
	Base + 0x0F2	Port D: Pin10 Data Read/Write (PINDATA10)	8	R/W	A
	Base + 0x0F3	Port D: Pin11 Data Read/Write (PINDATA11)	8	R/W	A
	Base + 0x0F4	Port D: Pin12 Data Read/Write (PINDATA12)	8	R/W	A
	Base + 0x0F5	Port D: Pin13 Data Read/Write (PINDATA13)	8	R/W	A
	Base + 0x0F6	Port D: Pin14 Data Read/Write (PINDATA14)	8	R/W	A
	Base + 0x0F7	Port D: Pin15 Data Read/Write (PINDATA15)	8	R/W	A
	Base + 0x0F8 to Base + 0x0FF	Reserved	Writing this register will have no effect. Reading this register will return 0x00		

Table 34-11. Port Integration Module Memory Map (Continued)

	Address	Use	Size	Access	Mode [1]
Port E	Base + 0x100	Port E: Pin 0 Configuration (CONFIG0)	16	R/W	A
	Base + 0x102	Port E: Pin 1 Configuration (CONFIG1)	16	R/W	A
	Base + 0x104	Port E: Pin 1 Configuration (CONFIG2)	16	R/W	A
	Base + 0x106	Port E: Pin 3 Configuration (CONFIG3)	16	R/W	A
	Base + 0x108	Port E: Pin 4 Configuration (CONFIG4)	16	R/W	A
	Base + 0x10A	Port E: Pin 5 Configuration (CONFIG5)	16	R/W	A
	Base + 0x10C	Port E: Pin 6 Configuration (CONFIG6)	16	R/W	A
	Base + 0x10E	Port E: Pin 7 Configuration (CONFIG7)	16	R/W	A
	Base + 0x110	Port E: Pin 8 Configuration (CONFIG8)	16	R/W	A
	Base + 0x112	Port E: Pin 9 Configuration (CONFIG9)	16	R/W	A
	Base + 0x114	Port E: Pin 10 Configuration (CONFIG10)	16	R/W	A
	Base + 0x116	Port E: Pin 11 Configuration (CONFIG11)	16	R/W	A
	Base + 0x118	Port E: Pin 12 Configuration (CONFIG12)	16	R/W	A
	Base + 0x11A	Port E: Pin 13 Configuration (CONFIG13)	16	R/W	A
	Base + 0x11C	Port E: Pin 14 Configuration (CONFIG14)	16	R/W	A
	Base + 0x11E	Port E: Pin 15 Configuration (CONFIG15)	16	R/W	A
	Base + 0x110	Port E: Port Wide Interrupt Flag (PORTIFR)	16	R/W	A
	Base + 0x122	Reserved	Writing this register will have no effect. Reading this register will return 0x00		
	Base + 0x124	Port E: Port Wide Data Read (PORTDATA)	16	R	A
	Base + 0x126	Port E: Port Wide Input (PORTIR)	16	R	A
	Base + 0x128	Port E: Pin0 Data Read (PINDATA0)	8	R	A
	Base + 0x129	Port E: Pin1 Data Read (PINDATA1)	8	R	A
	Base + 0x12A	Port E: Pin2 Data Read (PINDATA2)	8	R	A
	Base + 0x12B	Port E: Pin3 Data Read (PINDATA3)	8	R	A
	Base + 0x12C	Port E: Pin4 Data Read (PINDATA4)	8	R	A
	Base + 0x12D	Port E: Pin5 Data Read (PINDATA5)	8	R	A
	Base + 0x12E	Port E: Pin6 Data Read (PINDATA6)	8	R	A
	Base + 0x12F	Port E: Pin7 Data Read (PINDATA7)	8	R	A
	Base + 0x130	Port E: Pin8 Data Read (PINDATA8)	8	R	A
	Base + 0x131	Port E: Pin9 Data Read (PINDATA9)	8	R	A
	Base + 0x132	Port E: Pin10 Data Read (PINDATA10)	8	R	A
	Base + 0x133	Port E: Pin11 Data Read (PINDATA11)	8	R	A
	Base + 0x134	Port E: Pin12 Data Read (PINDATA12)	8	R	A
	Base + 0x135	Port E: Pin13 Data Read (PINDATA13)	8	R	A
	Base + 0x136	Port E: Pin14 Data Read (PINDATA14)	8	R	A
	Base + 0x137	Port E: Pin15 Data Read (PINDATA15)	8	R	A
	Base + 0x138 to Base + 0x13F	Reserved	Writing this register will have no effect. Reading this register will return 0x00		

Table 34-11. Port Integration Module Memory Map (Continued)

	Address	Use	Size	Access	Mode [1]
Port F	Base + 0x140	Port F: Pin 0 Configuration (CONFIG0)	16	R/W	A
	Base + 0x142	Port F: Pin 1 Configuration (CONFIG1)	16	R/W	A
	Base + 0x144	Port F: Pin 2 Configuration (CONFIG2)	16	R/W	A
	Base + 0x146	Port F: Pin 3 Configuration (CONFIG3)	16	R/W	A
	Base + 0x148	Port F: Pin 4 Configuration (CONFIG4)	16	R/W	A
	Base + 0x14A	Port F: Pin 5 Configuration (CONFIG5)	16	R/W	A
	Base + 0x14C	Port F: Pin 6 Configuration (CONFIG6)	16	R/W	A
	Base + 0x14E	Port F: Pin 7 Configuration (CONFIG7)	16	R/W	A
	Base + 0x150	Port F: Pin 8 Configuration (CONFIG8)	16	R/W	A
	Base + 0x152	Port F: Pin 9 Configuration (CONFIG9)	16	R/W	A
	Base + 0x154	Port F: Pin 10 Configuration (CONFIG10)	16	R/W	A
	Base + 0x156	Port F: Pin 11 Configuration (CONFIG11)	16	R/W	A
	Base + 0x158	Port F: Pin 12 Configuration (CONFIG12)	16	R/W	A
	Base + 0x15A	Port F: Pin 13 Configuration (CONFIG13)	16	R/W	A
	Base + 0x15C	Port F: Pin 14 Configuration (CONFIG14)	16	R/W	A
	Base + 0x15E	Port F: Pin 15 Configuration (CONFIG15)	16	R/W	A
	Base + 0x160	Port F: Port Wide Interrupt Flag (PORTIFR)	16	R/W	A
	Base + 0x162	Reserved	Writing this register will have no effect. Reading this register will return 0x00		
	Base + 0x164	Port F: Port Wide Data Read/Write (PORTDATA)	16	R/W	A
	Base + 0x166	Port F: Port Wide Input (PORTIR)	16	R	A
	Base + 0x168	Port F: Pin0 Data Read/Write (PINDATA0)	8	R/W	A
	Base + 0x169	Port F: Pin1 Data Read/Write (PINDATA1)	8	R/W	A
	Base + 0x16A	Port F: Pin2 Data Read/Write (PINDATA2)	8	R/W	A
	Base + 0x16B	Port F: Pin3 Data Read/Write (PINDATA3)	8	R/W	A
	Base + 0x16C	Port F: Pin4 Data Read/Write (PINDATA4)	8	R/W	A
	Base + 0x16D	Port F: Pin5 Data Read/Write (PINDATA5)	8	R/W	A
	Base + 0x16E	Port F: Pin6 Data Read/Write (PINDATA6)	8	R/W	A
	Base + 0x16F	Port F: Pin7 Data Read/Write (PINDATA7)	8	R/W	A
	Base + 0x170	Port F: Pin8 Data Read/Write (PINDATA8)	8	R/W	A
	Base + 0x171	Port F: Pin9 Data Read/Write (PINDATA9)	8	R/W	A
	Base + 0x172	Port F: Pin10 Data Read/Write (PINDATA10)	8	R/W	A
	Base + 0x173	Port F: Pin11 Data Read/Write (PINDATA11)	8	R/W	A
	Base + 0x174	Port F: Pin12 Data Read/Write (PINDATA12)	8	R/W	A
	Base + 0x175	Port F: Pin13 Data Read/Write (PINDATA13)	8	R/W	A
	Base + 0x176	Port F: Pin14 Data Read/Write (PINDATA14)	8	R/W	A
	Base + 0x177	Port F: Pin15 Data Read/Write (PINDATA15)	8	R/W	A
	Base + 0x178 to Base + 0x17F	Reserved	Writing this register will have no effect. Reading this register will return 0x00		

Table 34-11. Port Integration Module Memory Map (Continued)

	Address	Use	Size	Access	Mode [1]
Port G	Base + 0x180	Port G: Pin 0 Configuration (CONFIG0)	16	R/W	A
	Base + 0x182	Port G: Pin 1 Configuration (CONFIG1)	16	R/W	A
	Base + 0x184	Port G: Pin 2 Configuration (CONFIG2)	16	R/W	A
	Base + 0x186	Port G: Pin 3 Configuration (CONFIG3)	16	R/W	A
	Base + 0x188	Port G: Pin 4 Configuration (CONFIG4)	16	R/W	A
	Base + 0x18A	Port G: Pin 5 Configuration (CONFIG5)	16	R/W	A
	Base + 0x18C	Port G: Pin 6 Configuration (CONFIG6)	16	R/W	A
	Base + 0x18E	Port G: Pin 7 Configuration (CONFIG7)	16	R/W	A
	Base + 0x190	Port G: Pin 8 Configuration (CONFIG8)	16	R/W	A
	Base + 0x192	Port G: Pin 9 Configuration (CONFIG9)	16	R/W	A
	Base + 0x194	Port G: Pin 10 Configuration (CONFIG10)	16	R/W	A
	Base + 0x196	Port G: Pin 11 Configuration (CONFIG11)	16	R/W	A
	Base + 0x198	Port G: Pin 12 Configuration (CONFIG12)	16	R/W	A
	Base + 0x19A	Port G: Pin 13 Configuration (CONFIG13)	16	R/W	A
	Base + 0x19C	Port G: Pin 14 Configuration (CONFIG14)	16	R/W	A
	Base + 0x19E	Port G: Pin 15 Configuration (CONFIG15)	16	R/W	A
	Base + 0x1A0	Port G: Port Wide Interrupt Flag (PORTIFR)	16	R/W	A
	Base + 0x1A2	Reserved	Writing this register will have no effect. Reading this register will return 0x00		
	Base + 0x1A4	Port G: Port Wide Data Read/Write (PORTDATA)	16	R/W	A
	Base + 0x1A6	Port G: Port Wide Input (PORTIR)	16	R	A
	Base + 0x1A8	Port G: Pin0 Data Read/Write (PINDATA0)	8	R/W	A
	Base + 0x1A9	Port G: Pin1 Data Read/Write (PINDATA1)	8	R/W	A
	Base + 0x1AA	Port G: Pin2 Data Read/Write (PINDATA2)	8	R/W	A
	Base + 0x1AB	Port G: Pin3 Data Read/Write (PINDATA3)	8	R/W	A
	Base + 0x1AC	Port G: Pin4 Data Read/Write (PINDATA4)	8	R/W	A
	Base + 0x1AD	Port G: Pin5 Data Read/Write (PINDATA5)	8	R/W	A
	Base + 0x1AE	Port G: Pin6 Data Read/Write (PINDATA6)	8	R/W	A
	Base + 0x1AF	Port G: Pin7 Data Read/Write (PINDATA7)	8	R/W	A
	Base + 0x1B0	Port G: Pin8 Data Read/Write (PINDATA8)	8	R/W	A
	Base + 0x1B1	Port G: Pin9 Data Read/Write (PINDATA9)	8	R/W	A
	Base + 0x1B2	Port G: Pin10 Data Read/Write (PINDATA10)	8	R/W	A
	Base + 0x1B3	Port G: Pin11 Data Read/Write (PINDATA11)	8	R/W	A
	Base + 0x1B4	Port G: Pin12 Data Read/Write (PINDATA12)	8	R/W	A
	Base + 0x1B5	Port G: Pin13 Data Read/Write (PINDATA13)	8	R/W	A
	Base + 0x1B6	Port G: Pin14 Data Read/Write (PINDATA14)	8	R/W	A
	Base + 0x1B7	Port G: Pin15 Data Read/Write (PINDATA15)	8	R/W	A
	Base + 0x1B8 to Base + 0x1BF	Reserved	Writing this register will have no effect. Reading this register will return 0x00		

Table 34-11. Port Integration Module Memory Map (Continued)

	Address	Use	Size	Access	Mode [1]
Misc	Base + 0x01C0 to Base + 0x03BF	Reserved	See Note [2]		
	Base + 0x03C0	Global Interrupt Status (GLBLINT)	16	R	A
	Base + 0x03C2	PIM Configuration (PIMCONFIG)	16	R/W	A
	Base + 0x03C4	TDI Pin Configuration (CONFIG_TDI)	16	R/W	A
	Base + 0x03C6	TDO Pin Configuration (CONFIG_TDO)	16	R/W	A
	Base + 0x03C8	TMS Pin Configuration (CONFIG_TMS)	16	R/W	A
	Base + 0x03CA	TCK Pin Configuration (CONFIG_TCK)	16	R/W	A
	Base + 0x03CC	Reserved	Writing this register will have no effect. Reading this register will return 0x0000		
	Base + 0x03CE	RESET Pin Configuration (CONFIG_RESET)	16	R/W	A
	Base + 0x03D0 to Base + 0x03DF	Reserved	See Note [2]		
	Base + 0x03E0	PORT32IR_AB	32	R	A
	Base + 0x03E4	PORT32IR_CD	32	R	A
	Base + 0x03E8	PORT32IR_F	32	R	A
	Base + 0x03EC	PORT32IR_G	32	R	A
	Base + 0x03F0	PORT32IR_BC	32	R	A
	Base + 0x03F4	PORT32IR_D	32	R	A
	Base + 0x03F8	PORT32IR_FG	32	R	A
	Base + 0x03FC	Reserved	Writing this register will have no effect. Reading this register will return 0x0000		
	Base + 0x0400 to Base + 0x3FFF	Reserved	See Note [2]		

1. **U** = User Mode, **S** = Supervisor Mode, **A** = All (No restrictions)
2. If enabled at the SoC level, accessing these registers will cause bus aborts. Refer to the System Services Module documentation for more details.

All registers are accessible via 8-bit, 16-bit or 32-bit accesses. However, 16-bit accesses must be aligned to 16-bit boundaries, and 32-bit accesses must be aligned to 32-bit boundaries. As an example, the CONFIG1 register for Port A is accessible by a 16-bit READ/WRITE to address 'Base + 0x000', but performing a 16-bit access to 'Base + 0x001' is illegal.

34.6.1 Register Descriptions

This section consists of register descriptions in address order. Because 6 Ports (Port A, B, C, D, F, G) are identical in their memory map, the following descriptions will only be given for a single Port.

34.6.1.1 Pin Configuration Register (Port A, B, C, D, F, G)

Each pin of Port A, B, C, D, F, G (16 per port) may be independently configured to select between Peripheral and GPIO Modes, as well as the pin characteristics when GPIO Mode is selected. Note that

some pins do not have Peripheral Mode functionality. In this case, when the pin is programmed to be in Peripheral Mode, it will configured to be an output and will drive a low (0) signal out.

Offset see Table 34-11 Access: see Table 34-13

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	SLEWDIS	0	0	0	0	0	0	MODE		DDR	ODER	SLEW	PULL		PIER	PIFR
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-3. Pin Configuration Register (CONFIG*xx*, Port A, B, C, D, F, G)

Table 34-12. CONFIG*xx* (Port A, B, C, D, F, G) Field Descriptions

Field	Description
15 SLEWDIS	Slew Rate Disable Register. This bit enables/disables slew rate control. If enabled, the Slew Rate Control Register can be used to select between Slow and Fast slew rates. If disabled, no slew rate control is in effect, and the Slew Rate Control Register has no effect. 0 Slew rate control enabled 1 Slew rate control disabled
14–9	Reserved, should be cleared.
8–7 MODE	Pin Mode Register. These bits switch the associated pin between Peripheral and GPI Modes. 00 GPIO Mode 01 Primary Peripheral Mode 10 Secondary Peripheral Mode 11 Tertiary Peripheral Mode **Note:** It is not possible to use pad PD2 as a General Purpose Output. Pad PD2 behaves virtually identically if it is configured in General Purpose Output Mode or in Peripheral Mode except that, in Peripheral Mode, the open drain functionality is disabled. In GPO Mode, the open drain functionality may be enabled/disabled via the ODER bit. **Note:** If a Secondary Peripheral is not defined for the pin, then setting the Pin Mode to Secondary Peripheral Mode is exactly equivalent to setting it to Primary Peripheral Mode. The same is true for Tertiary Peripheral Mode. **Note:** Currently, only pins PA8 and PA9 allow usage of non-primary peripheral mode. For all other pins only MODE[1:0] values of 2'b00 or 2'b01 are allowed.
6 DDR	Data Direction Register. This bit switches the pin between output and input *when the pin is in GPIO Mode* (MODE=0). 0 Pin is an input 1 Pin is an output
5 ODER	Open Drain Enable Register. This bit configures the pin to be open drain *when the pin is in GPIO Mode* (MODE=0). 0 Open Drain is disabled on the pin 1 Open Drain is enabled on the pin
4 SLEW	Slew Rate Control Register. This bit selects the slew rate of the pin, and is used only when the pin is selected as an output and the Slew Rate Enable is set. 0 Fast slew rate selected 1 Slow slew rate selected

Table 34-12. CONFIG*xx* (Port A, B, C, D, F, G) Field Descriptions (Continued)

Field	Description
3–2 PULL	Pull-up/Pull-down Enable Register. These bits serves a dual purpose by selecting the polarity of the active interrupt edge as well as selecting a pull-up or pull-down device if enabled. Pull-ups/pull-downs may be enabled or disabled when the pin is in either Peripheral or GPIO modes. **Value** / **Function** / **External Interrupt Edge** 00 / No Pull-up/Pull-down / Rising Edge 01 / No Pull-up/Pull-down / Falling Edge 10 / Pull-down Enabled / Rising Edge 11 / Pull-up Enabled / Falling Edge
1 PIER	Pin Interrupt Enable Register. This bit masks the interrupt associated with a particular Pin Interrupt Flag Register bit. If this bit is set, an interrupt will occur if the corresponding Flag Register bit is set. If this bit is cleared, then no interrupt will occur if the Flag Register bit is set. Note that this bit does not affect the setting/clearing of the Flag Register bit, it only determines whether a system interrupt will be generated or not. 0 Interrupt is masked 1 Interrupt is generated when PIFR is set by an active edge
0 PIFR	Pin Interrupt Flag Register. This bit signals that an active edge (i.e.-external interrupt) has been detected. Writing a '1' to this bit clears the pending interrupt. Writing '0' has no effect. Interrupts are enabled on the pin *only when the pin is in GPIO Mode and is configured to be an input* (MODE=0, DDR=0). 0 No active edge pending. 1 An active edge has been detected on the corresponding pin. If the PIER bit is also set, an interrupt will occur.

Table 34-13. CONFIG*xx* (Port A, B, C, D, F, G) Allowed Register Accesses

	8-bit	16-bit	32-bit
READ	Allowed	Allowed	Allowed [1] (Reads multiple CONFIG registers)
WRITE	Not Recommended [2]	Allowed	Allowed [1] (Writes multiple CONFIG registers)

1. All 32-bit accesses must be aligned to 32-bit addresses (i.e.-0xxx0, 0xxx4, 0xxx8 or 0xxxC).
2. For backwards compatibility, 8-bit writes to the lower 8-bits (bit [7:0]) of a CONFIG register may be done if the upper 8-bits are cleared (zero). For future compatibility, it is recommended to use a 16-bit read-mask-write operation on this register.

34.6.1.2 Pin Configuration Register (Port E)

NOTE

Because Port E implements only General Purpose Input (GPI) functionality, the functionality of this register differs from the rest of the Ports. Careful attention should be paid to the description of this register.

Each pin of Port E (16 pins) may be independently configured to select an optional pull-up or pull-down. In order to use pins in Port E as ATD analog channels or external triggers, the pin must be configured to Peripheral Mode.

Offset see Table 34-11 Access: see Table 34-15

	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	MODE	0	0	0	PULL		PIER	PIFR
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-4. Pin Configuration Register (CONFIG*xx*, Port E)

Table 34-14. CONFIG*xx* (Port E) Field Descriptions

Field	Description
15–8	Reserved, should be cleared.
7 MODE	Pin Mode Register. This bit switches the associated pin between Peripheral and GPI Modes 0 GPI Mode 1 Peripheral Mode
6–4	Reserved, should be cleared.
3–2 PULL	Pull-up/Pull-down Enable Register. These bits serves a dual purpose by selecting the polarity of the active interrupt edge as well as selecting a pull-up or pull-down device if enabled. Pull-ups/pull-downs may be enabled or disabled anytime, but only take effect when the pin is in GPIO mode. In Peripheral mode the peripheral takes control of the pull configuration and interrupt functionality is disabled.

Value	Function	External Interrupt Edge
00	No Pull-up/Pull-down	Rising Edge
01	No Pull-up/Pull-down	Falling Edge
10	Pull-down Enabled	Rising Edge
11	Pull-up Enabled	Falling Edge

1 PIER	Pin Interrupt Enable Register. This bit masks the interrupt associated with a particular Pin Interrupt Flag Register bit. If this bit is set, an interrupt will occur if the corresponding Flag Register bit is set. If this bit is cleared, then no interrupt will occur if the Flag Register bit is set. Note that this bit does not affect the setting/clearing of the Flag Register bit, it only determines whether a system interrupt will be generated or not. 0 Interrupt is masked 1 Interrupt is generated when **PIFR** is set by an active edge
0 PIFR	Pin Interrupt Flag Register. This bit signals that an active edge (i.e.-external interrupt) has been detected. Writing a '1' to this bit clears the pending interrupt. Writing '0' has no effect. Interrupts are enabled on the pin *only when the pin is in GPI Mode* (**MODE**=0). 0 No active edge pending. 1 An active edge has been detected on the corresponding pin. If the **PIER** bit is also set, an interrupt will occur.

Table 34-15. CONFIG*xx* (Port E) Allowed Register Accesses

	8-bit	16-bit	32-bit
READ	Not Recommended [1]	Allowed	Allowed [2] (Reads multiple CONFIG registers)
WRITE	Not Recommended [1]	Allowed	Allowed [2] (Writes multiple CONFIG registers)

1. For future compatibility, it is recommended to use a 16-bit or 32-bit read-mask-write operations on this register.
2. All 32-bit accesses must be aligned to 32-bit addresses (i.e.-0xxx0, 0xxx4, 0xxx8 or 0xxxC).

34.6.1.3 Port Wide Interrupt Flag Register

Offset see Table 34-11 Access: see Table 34-17

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	IFR15	IFR14	IFR13	IFR12	IFR11	IFR10	IFR9	IFR8	IFR7	IFR6	IFR5	IFR4	IFR3	IFR2	IFR1	IFR0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-5. Port Wide Interrupt Flag Register (PORTIFR)

Table 34-16. PORTIFR Field Descriptions

Field	Descriptions
15–0 IFR*xx*	Interrupt Flag bit. This 16-bit register allows the reading and clearing of the PIFR (Pin Interrupt Flag Register) bits for all pins in a single bus access. This is useful for determining the exact source of an interrupt/wakeup, and for clearing multiple interrupts simultaneously. Writing a '1' to a bit clears the corresponding pending interrupt. Writing '0' has no effect. Interrupts are enabled on the pin only when the pin is in GPIO Mode and is configured to be an input (MODE=0, DDR=0). Note that writing/reading to/from a bit in this register is functionally identical to writing/reading to/from the PIFR bit in the CONFIG register for that pin. As an example, setting IFR1 is the same as setting the PIFR bit in register CONFIG1.

Table 34-17. PORTIFR Allowed Register Accesses

	8-bit	16-bit	32-bit
READ	Allowed	Allowed	Not Recommended
WRITE	Allowed	Allowed	Not Recommended

34.6.1.4 Port Wide Data Register (Port A, B, C, D, F, G)

Offset see Table 34-11 Access: see Table 34-19

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	DATA															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-6. Port Wide Data Register (PORTDATA, Port A, B, C, D, F, G)

Table 34-18. PORTDATA (Port A, B, C, D, F, G) Field Descriptions

Field	Descriptions
15–0 DATA	16-bit Write/Read Data Register. This 16-bit field is used to drive and sample values to and from the associated pins. It has the following functionality (on a pin-by-pin basis): • In Peripheral Mode, the corresponding bit of the register is not used • In GPIO Mode, when the pin is configured as an input (MODE=0, DDR=0), the corresponding bit in the PORTDATA register reflects the value(s) driven onto the pin(s) by external hardware, not the value previously written to the PORTDATA register. Note that there is a 2 clock cycle delay between the changing of the value on a pin and the reflection of the new value in the PORTDATA register. In GPIO Mode, when the pin is configured as an output (MODE=0, DDR=1), the corresponding bit in the PORTDATA register is used to specify the value to drive onto the pin.

Table 34-19. PORTDATA (Port A, B, C, D, F, G) Allowed Register Accesses

	8-bit	16-bit	32-bit
READ	Allowed	Allowed	Not Recommended
WRITE	Allowed	Allowed	Not Recommended

34.6.1.5 Port Wide Data Register (Port E)

NOTE

Because Port E implements only General Purpose Input (GPI) functionality, the functionality of this register differs from the rest of the Ports. Careful attention should be paid to the description of this register.

Offset see Table 34-11 Access: see Table 34-21

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	DATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-7. Port Wide Data Register (PORTDATA Port E)

Table 34-20. PORTDATA (Port E) Field Description

Field	Description
15–0 DATA	16-bit Write/Read Data Register. This 16-bit field is used to sample values to and from the associated pins. It has the following functionality (on a pin-by-pin basis): • In Peripheral Mode, the corresponding bit of the register is not used • In GPI Mode the corresponding bit in the PORTDATA register reflects the value(s) driven onto the pin(s) by external hardware. Note that there is a 2 clock cycle delay between the changing of the value on a pin and the reflection of the new value in the PORTDATA regist

Table 34-21. PORTDATA (Port E) Allowed Register Accesses

	8-bit	16-bit	32-bit
READ	Allowed	Allowed	Not Recommended
WRITE	Not Allowed	Not Allowed	Not Allowed

34.6.1.6 Port Wide Input Register (Port A, B, C, D, F, G)

Offset see Table 34-11 Access: see Table 34-23

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	DATA															
W																
Reset	0/1[1]	0/1[1]	0/1[1]	0/1[1]	0/1[1]	0/1[1]	0/1[1]	0/1[1]	0/1[1]	0/1[1]	0/1[1]	0/1[1]	0/1[1]	0/1[1]	0/1[1]	0/1[1]

[1] The reset value of this register depends on the value of the pins at the time of the first register READ after reset.

Figure 34-8. Port Wide Input Register (PORTIR, Port A, B, C, D, F, G)

Table 34-22. PORTIR (Port A, B, C, D, F, G) Field Descriptions

Field	Description
15–0 DATA	16-bit Port Read Register. This 16-bit read-only register reads back the value on the associated pins, and can be used to detect overload or short circuit conditions on output pins. Functionally, it is related to the PORTDATA register in the following manner: • When the pin is configured as an input (DDR=0), reading the PORTDATA and PORTIR registers will yield the same result • When the pin is configured as an output (DDR=1), reading the PORTIR register will yield the value on the pin, while reading the PORTDATA register will yield the value in the PORTDATA register. If the pin is in Peripheral Mode (MODE=1), then the value in the PORTDATA register may not be the same as the value driven onto the pin by the peripheral. **Note:** There is a 2 cycle synchronization that is performed on pin values before registering the value. Therefore, there must be a minimum 2 system clock cycles between the time a pin is driven, and the time that the register read is performed. **Note:** Not all Ports are available on all packages. In the case where a port, or some pins in a port are not available on a package, the value of the corresponding bit(s) in the PORTIR register are indeterminate.

Table 34-23. PORTIR (Port A, B, C, D, F, G) Allowed Register Accesses

	8-bit	16-bit	32-bit
READ	Allowed	Allowed	Not Allowed
WRITE	Not Allowed	Not Allowed	Not Allowed

34.6.1.7 Port Wide Input Register (Port E)

NOTE

Because Port E implements only General Purpose Input (GPI) functionality, the functionality of this register differs from the rest of the Ports. Careful attention should be paid to the description of this register.

Offset see Table 34-11 — Access: see Table 34-25

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	DATA															
W																
Reset	0/1[1]	0/1[1]	0/1[1]	0/1[1]	0/1[1]	0/1[1]	0/1[1]	0/1[1]	0/1[1]	0/1[1]	0/1[1]	0/1[1]	0/1[1]	0/1[1]	0/1[1]	0/1[1]

[1] The reset value of this register depends on the value of the pins at the time of the first register READ after reset.

Figure 34-9. Port Wide Input Register (PORTIR, Port E)

Table 34-24. PORTIR (Port E) Field Descriptions

Field	Description
15–0 DATA	16-bit Port Read Register. This 16-bit read-only register reads back the value on the associated pins, and can be used to detect overload or short circuit conditions on output pins. **Note:** There is a 2 cycle synchronization that is performed on pin values before registering the value. Therefore, there must be a minimum 2 system clock cycles between the time a pin is driven, and the time that the register read is performed. **Note:** Not all Ports are available on all packages. In the case where a port, or some pins in a port are not available on a package, the value of the corresponding bit(s) in the PORTIR register are indeterminate.

Table 34-25. PORTIR (Port E) Allowed Register Accesses

	8-bit	16-bit	32-bit
READ	Allowed	Allowed	Not Allowed
WRITE	Not Allowed	Not Allowed	Not Allowed

34.6.1.8 Pin Data Register (Port A, B, C, D, F, G)

Offset see Table 34-11 — Access: see Table 34-27

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

Figure 34-10. Pin Data Register (PINDATA*xx*, Port A, B, C, D, F, G)

Table 34-26. PINDATA*xx* (Port A, B, C, D, F, G) Field Descriptions

Field	Description
7–1	Reserved, should be cleared.
0 DATA	1-bit Write/Read Data Register. This 1-bit field provides a means to drive and sample individual pins without the need for mask and shift operations in software, and is particularly useful for using the GPIO functionality of a pin under DMA control. Writing/Reading to/from this register is functionally equivalent to writing/reading the corresponding bit in the PORTDATA register, but without the need to mask and shift the value. (i.e.-Reading from PINDATA3 is equivalent to reading PORTDATA, AND'ing the result with 0x0008, and shifting the result right by 3 places). See Section 34.8.2.2.2, "Driving/Sampling Individual Pins," for examples using this feature.

Table 34-27. PINDATA*xx* (Port A, B, C, D, F, G) Allowed Register Accesses

	8-bit	16-bit	32-bit
READ	Allowed	Allowed	Allowed
WRITE	Allowed	Allowed	Allowed

34.6.1.9 Pin Data Register (Port E)

NOTE

Because Port E implements only General Purpose Input (GPI) functionality, the functionality of this register differs from the rest of the Ports. Careful attention should be paid to the description of this register.

Offset see Table 34-11 Access: see Table 34-29

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

Figure 34-11. Pin Data Register (PINDATA*xx*, Port E)

Table 34-28. PINDATA*xx* (Port E) Field Descriptions

Field	Description
7–1	Reserved, should be cleared.
0 DATA	1-bit Read Data Register. This 1-bit field provides a means to sample individual pins without the need for mask and shift operations in software. Reading from this register is functionally equivalent to reading the corresponding bit in the PORTIR register, but without the need to mask and shift the value. (i.e.-Reading from PINDATA3 is equivalent to reading PORTIR, AND'ing the result with 0x0008, and shifting the result right by 3 places). See Section 34.8.2.2.2, "Driving/Sampling Individual Pins," for examples using this feature.

Table 34-29. PINDATA*xx* (Port E) Allowed Register Accesses

	8-bit	16-bit	32-bit
READ	Allowed	Allowed	Allowed
WRITE	Not Allowed [1]	Not Allowed [1]	Not Allowed [1]

1. Functionality differs from other Ports.

34.6.1.10 Global Interrupt Status Register

This register allows software to determine which ports currently have interrupts pending.

Offset see Table 34-11 Access: see Table 34-31

	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	INT_PENDING						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-12. Global Interrupt Status Register (GLBLINT)

Table 34-30. GLBLINT Field Descriptions

Field	Description
15–7	Reserved, should be cleared.
6–0 INT_PENDING	Interrupt Pending. These bits reflect any pending interrupts on Port A, B, C, D, F, E, G. 0 No interrupt is pending 1 Interrupt is pending

Bit	Port	Bit	Port
0	Port A	4	Port E
1	Port B	5	Port F
2	Port C	6	Port G
3	Port D		

Table 34-31. GLBLINT Allowed Register Accesses

	8-bit	16-bit	32-bit
READ	Allowed	Allowed	Not Recommended
WRITE	Not Allowed	Not Allowed	Not Allowed

34.6.1.11 PIM Configuration Register

This register allows software to disable or enable the external bus interface of the device.

Offset see Table 34-11 Access: see Table 34-33

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	EIMCLKEN	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1/0[1]	0

[1] The reset value of EIMCLKEN depends on the Chip Mode of the system. If the system is placed into a mode where booting is done from the external bus, the reset value will be ’1’. Otherwise, the reset value will be ’0’.

Figure 34-13. PIM Configuration Register (PICONFIG)

Table 34-32. PIMCONFIG Field Descriptions

Field	Description
15–2	Reserved, should be cleared.
1 EIMCLKEN	Clock Enable for the EIM module. This bit enables or disables the clock to the EIM module, which acts as the bridge between the internal bus and the external bus. If you are not using the external bus, you may clear this bit for lower power operation. 0 EIM module clock is disabled 1 EIM module clock is enabled
0	Reserved, should be cleared.

Table 34-33. PIMCONFIG Allowed Register Accesses

	8-bit	16-bit	32-bit
READ	Allowed	Allowed	Not Allowed
WRITE	Allowed	Allowed	Not Allowed

34.6.1.12 TDI Pin Configuration Register

The TDI Pin Configuration Register (CONFIG_TDI) may be used to enable or disable certain characteristics of the TDI pad.

Offset see Table 34-11 Access: see Table 34-35

	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	DDR	0	0	PULL		0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0

Figure 34-14. Pin Configuration Register (CONFIG_TDI)

Table 34-34. CONFIG_TDI Field Descriptions

Field	Description
15–7	Reserved, should be cleared.
6 DDR	Data Direction Register. This bit switches the pin between output and input. 0 Pin is an input. The TDI functionality is enabled. 1 Pin is an output. The TDI functionality is disabled and the pad is driven low.
5–4	Reserved, should be cleared.
3–2 PULL	Pull-up/Pull-down Enable Register. These bits select a pull-up or pull-down device if enabled, and is only valid when the pin is selected as an input. Note that the Pull-up functionality is enabled at reset. 00 No Pull-up/Pull-down 01 No Pull-up/Pull-down 10 Pull-down Enabled 11 Pull-up Enabled
1–0	Reserved, should be cleared.

Table 34-35. CONFIG_TDI Allowed Register Accesses

	8-bit	16-bit	32-bit
READ	Allowed	Allowed	Allowed (Reads CONFIG_TDI + CONFIG_TDO)
WRITE	Allowed	Allowed	Allowed (Writes CONFIG_TDI + CONFIG_TDO)

34.6.1.13 TDO Pin Configuration Register

The TDO Pin Configuration Register (CONFIG_TDO) may be used to enable or disable certain characteristics of the TDO pad.

Offset see Table 34-11 Access: see Table 34-37

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	SLEWDIS	0	0	0	0	0	0	0	0	DDR	0	SLEW	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0

Figure 34-15. Pin Configuration Register (CONFIG_TDO)

Table 34-36. CONFIG_TDO Field Descriptions

Field	Description
15 SLEWDIS	Slew Rate Disable Register. This bit enables/disables slew rate control. If enabled, the Slew Rate Control Register can be used to select between Slow and Fast slew rates. If disabled, no slew rate control is in effect, and the Slew Rate Control Register has no effect. 0 Slew rate control enabled 1 Slew rate control disabled
14–7	Reserved, should be cleared.
6 DDR	Data Direction Register. This bit switches the pin between output and input. 0 Pin is an input. The TDO functionality is disabled. All changes on the pad are ignored. 1 Pin is an output. The TDO functionality is enabled.
5	Reserved, should be cleared.
4 SLEW	Slew Rate Control Register. This bit selects the slew rate of the pin, and is used only when the pin is selected as an output and the Slew Rate Enable is set. 0 Fast slew rate selected 1 Slow slew rate selected
3–0	Reserved, should be cleared.

Table 34-37. CONFIG_TDO Allowed Register Accesses

	8-bit	16-bit	32-bit
READ	Allowed	Allowed	Not Allowed
WRITE	Allowed	Allowed	Not Allowed

34.6.1.14 TMS Pin Configuration Register

The TMS Pin Configuration Register (CONFIG_TMS) may be used to enable or disable certain characteristics of the TMS pad.

Offset see Table 34-11 Access: see Table 34-39

	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	DDR	0	0	PULL		0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0

Figure 34-16. Pin Configuration Register (CONFIG_TMS)

Table 34-38. CONFIG_TMS Field Descriptions

Field	Description
15–7	Reserved, should be cleared.
6 DDR	Data Direction Register. This bit switches the pin between output and input. 0 Pin is an input. The TDI functionality is enabled. 1 Pin is an output. The TDI functionality is disabled and the pad is driven low.
5–4	Reserved, should be cleared.
3–2 PULL	Pull-up/Pull-down Enable Register. These bits select a pull-up or pull-down device if enabled, and is only valid when the pin is selected as an input. Note that the Pull-up functionality is enabled at reset. 00 No Pull-up/Pull-down 01 No Pull-up/Pull-down 10 Pull-down Enabled 11 Pull-up Enabled
1–0	Reserved, should be cleared.

Table 34-39. CONFIG_TMS Allowed Register Accesses

	8-bit	16-bit	32-bit
READ	Allowed	Allowed	Allowed (Reads CONFIG_TMS + CONFIG_TCK)
WRITE	Allowed	Allowed	Allowed (Writes CONFIG_TMS + CONFIG_TCK)

34.6.1.15 TCK Pin Configuration Register

The TCK Pin Configuration Register (CONFIG_TCK) may be used to enable or disable certain characteristics of the TCK pad.

Offset see Table 34-11 Access: see Table 34-41

	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	DDR	0	0	PULL		0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

Figure 34-17. Pin Configuration Register (CONFIG_TCK)

Table 34-40. CONFIG_TCK Field Descriptions

Field	Description
15–7	Reserved, should be cleared.
6 DDR	Data Direction Register. This bit switches the pin between output and input. 0 Pin is an input. The TDI functionality is enabled. 1 Pin is an output. The TDI functionality is disabled and the pad is driven low.
5–4	Reserved, should be cleared.
3–2 PULL	Pull-up/Pull-down Enable Register. These bits select a pull-up or pull-down device if enabled, and is only valid when the pin is selected as an input. Note that the Pull-down functionality is enabled at reset. 00 No Pull-up/Pull-down 01 No Pull-up/Pull-down 10 Pull-down Enabled 11 Pull-up Enabled
1–0	Reserved, should be cleared.

Table 34-41. CONFIG_TCK Allowed Register Accesses

	8-bit	16-bit	32-bit
READ	Allowed	Allowed	Not Allowed
WRITE	Allowed	Allowed	Not Allowed

34.6.1.16 RESET Pin Configuration Register

The RESET Pin Configuration Register (CONFIG_RESET) may be used to enable a pull-down device on the RESET pad. Since the RESET line is ACTIVE_LOW use this feature with caution. Also note that this configuration can be written only once between two system resets and that the configuration is reset only with a Power on Reset. So in order to securely disable this feature it is sufficient to write a 0x0000 to this configuration register during startup.

Offset see Table 34-11 Access: see Table 34-43

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	PULL	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-18. Pin Configuration Register (CONFIG_RESET)

Table 34-42. CONFIG_RESET Field Descriptions

Field	Description
15–4	Reserved, should be cleared.
3 PULL	Pull-down Enable Register. This bit selects a pull-down device on the RESET pad. This bit is write once. Only a Power on Reset will reset this bit. 0 No Pull-down 1 Pull-down Enabled
2–0	Reserved, should be cleared.

Table 34-43. CONFIG_RESET Allowed Register Accesses

	8-bit	16-bit	32-bit
READ	Allowed	Allowed	Not Allowed
WRITE	Allowed (once to [7:0])	Allowed (once)	Not Allowed

34.6.1.17 Double Port Wide Input Registers

There are several Double Port Wide Input Registers available that can be used to read the value of 2 ports simultaneously. Each 32-bit register is read-only and will read back the *synchronized* value on the corresponding pin. The following table lists these registers.

Table 34-44. Double Port Wide Input Registers (DPORTIR)

Address Offset	Ports
0x03e0	Port A / Port B
0x03e4	Port C / Port D
0x03e8	Port E / Port F
0x03ec	Port G / 0x0000
0x03f0	Port B / Port C
0x03f4	Port D / Port E
0x03f8	Port F / Port G

NOTE

The reset value of this register depends on the value of the pins at the time of the first register READ after reset.

NOTE

Not all Ports are available on all packages. In the case where a port, or some pins in a port are not available on a package, the value of the corresponding bit(s) in the DPORTIR register are indeterminate.

Table 34-45. DPORTIR Allowed Register Accesses

	8-bit	16-bit	32-bit
READ	Allowed [1]	Allowed[1]	Allowed
WRITE	Not Allowed	Not Allowed	Not Allowed

1. While these accesses are allowed, the intended use of this register is for 32-bits. For 8 and 16 bit port reads, it is recommended to use the PORTIR register for each individual port instead. This will ensure better future compatibility.

34.7 Functional Description

The Port Integration Module provides the means to utilize unused pins as General Purpose Inputs/Outputs (GPIO), with the following configurable capabilities:

- 5V output drive with three selectable slew rates (Disabled, Slow, Fast)
- Pull-up or pull-down
- Open drain for wired-or connections
- Interrupt capability (with glitch filtering and interrupt mask)

NOTE

All references in this section to GPO Mode do not apply to Port E.

34.7.1 Reset

After reset, all ports (Port A, B, C, D, E, F, G) are configured as General Purpose Inputs (See Section 34.7.4, “General Purpose Input mode”). In addition, the JTAG pads are configured as follows:

- TDI Pad:
  - — Input
  - — Pull-up enabled
- TDO Pad:
  - — Output
  - — Fast slew rate enabled
- TMS Pad:
  - — Input
  - — Pull-up enabled
- TCK Pad:
  - — Input
  - — Pull-down enabled

34.7.2 Peripheral Mode

In Peripheral mode, the peripheral functionality associated with the pin(s) are under the control of the peripheral itself.

Peripheral mode is set on a particular pin by setting the MODE bit in the pin's corresponding CONFIG*xx* register (See Section 34.8.1, "Using a Pin in Peripheral Mode," for an example).

In Peripheral mode, the following register bits are unused:

- DDR (CONFIG*xx*[6])
- ODER (CONFIG*xx*[5])
- PIER (CONFIG*xx*[1])
- PIFR (CONFIG*xx*[0])

NOTE

Even though the above bits are unused in this mode, they can still be written to. The new values will take effect once the mode of the pin is switched to GPI or GPO mode.

In addition, the corresponding bit in the PIFR register will not be changed while the pin is in Peripheral mode. While in Peripheral mode, the data registers PORTDATA**,** PORTIR and PINDATA*xx* may be used as follows:

Table 34-46. PIM Register Behavior in Peripheral Mode

Register	Write	Read
PORTDATA	Writing to the corresponding bit in the PORTDATA register will not drive the value onto the pin, but it will set/clear the bit in the register. When the pin is switched to General Purpose Output mode, the new value will be driven onto the pin.	The value read from the PORTDATA register in Peripheral mode depends on the current setting of the pin's DDR (Data Direction) bit. • DDR = 1 (Output) - Returns the actual value in the current PORTDATA register. • DDR = 0 (Input) - Returns the value driven on the pin, synchronized to the system clock. This is identical to reading the corresponding bit in the PORTIR register.
PORTIR	Writing to the PORTIR register will have no effect, as it is a read-only register in all modes.	Reading from the PORTIR register will return the value on the pin. Note that this value is internally synchronized to the system clock, and that there is a two clock cycle delay between the changing of the value on the pin, and the update of the PORTIR register.
PINDATA*xx*	Writing to the PINDATA*xx* register is identical to writing the corresponding bit in the PORTDATA register.	Reading from the PINDATA*xx* register is identical to reading the corresponding bit from the PORTDATA register.

Figure 34-19 illustrates the anatomy of a single PIM pad cell in Peripheral mode.

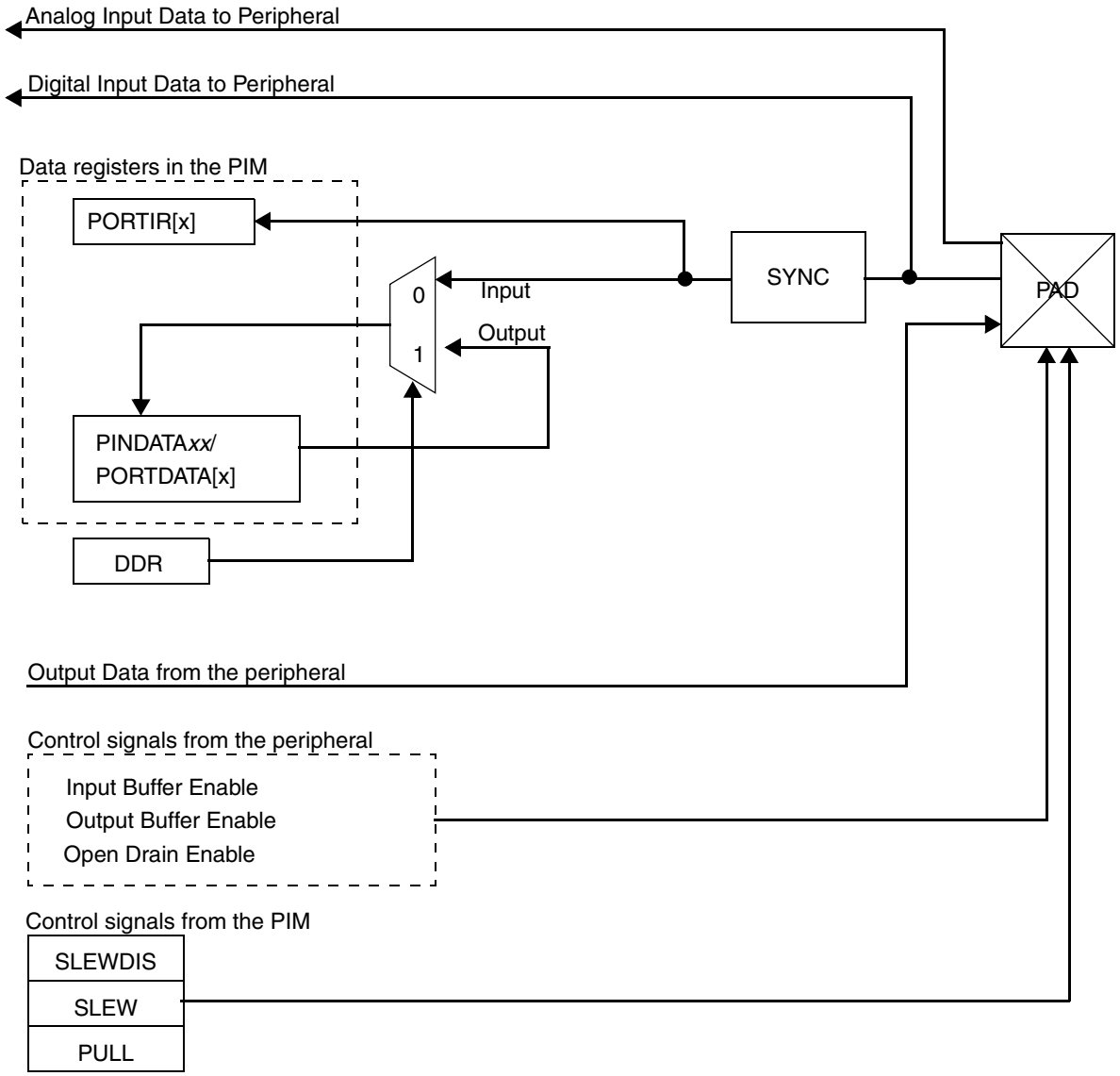


Figure 34-19. Pad in Peripheral Mode (including pad PD2)

NOTE

The DDR bit and PORTDATA register are not available on Port E.

34.7.3 Peripheral Pin Multiplexing

Peripheral muxing is a feature intended to increase the flexibility of the chip and broaden its possible application fields by allowing the user to define some peripheral functions to be made available at more than one pin location. To this effect every pin (besides Port E) has a mode field as described in Section 34.6.1.1, “Pin Configuration Register (Port A, B, C, D, F, G).” Table 34-47 lists the possible modes. These modes are software selectable by writing the mode bits with the desired value.

Table 34-47. MODE[1:0] Values

Value	Mode
00	GPIO Mode
01	Primary Peripheral Mode
10	Secondary Peripheral Mode
11	Tertiary Peripheral Mode

NOTE

Currently, only pins PA8 and PA9 allow usage of non-primary peripheral mode. For all other pins only MODE[1:0] values of 2’b00 or 2’b01 are allowed.

Table 34-48. Peripheral Pins that can be Multiplexed

Pin	Primary Peripheral Function	Secondary Peripheral Function	Tertiary Peripheral Function
PA8	FlexBus DATA[8]	DSPI_B CS4	None
PA9	FlexBus DATA[9]	DSPI_B CS3	None
PG14	DSPI_B CS4	None	None
PG15	DSPI_B CS3	None	None

This table shows that the chips select lines 3 and 4 from DSPI B are available on pins PG14 and PG15 (with MODE[1:0] = 2’b01) as well as on pins PA8 and PA9 if those are configured to use their secondary peripheral function (i.e. MODE[1:0] = 2’b10).

34.7.3.1 Driving Multiple Outputs

It is possible to drive the same peripheral outputs on multiple pins. If this is not desired, it is the responsibility of the user to ensure that the PIM is not configured in this manner. As an example, if pin PA8 was configured to be in Secondary Peripheral Mode, the DSPI_B CS3 would be driven out on both PA8 and PG15 simultaneously.

34.7.3.2 Input Multiplexing - Priorities

It is not possible to drive the same peripheral input from multiple pins. In the case where the PIM is configured in this manner, the following priority scheme will determine which pin will drive the input:

Table 34-49. Input Multiplexing Priority

Priority	Mode
highest	Primary Peripheral Mode
medium	Secondary Peripheral Mode
lowest	Tertiary Peripheral Mode

Within a Peripheral Mode (Primary, Secondary, Tertiary), the lowest port number has priority, from PA0...PA15, PB0...PB15, etc.

As an example, assuming that the IIC SDA functionality is available on the following pins (hypothetical only),

- PBO (Primary Peripheral Mode)
- PB5 (Secondary Peripheral Mode)
- PB10 (Secondary Peripheral Mode)
- PB11 (Tertiary Peripheral Mode)

then the selection of the SDA input to the IIC would be done as follows:

If PBO is configured as Primary Peripheral Mode, use PB0, else...
if PB5 is configured as Secondary Peripheral Mode, use PB5, else...
if PB10 is configured as Secondary Peripheral Mode, use PB10, else...
if PB11 is configured as Tertiary Peripheral Mode, use PB11, else...
drive the SDA input to the IIC to its off value

34.7.4 General Purpose Input mode

In General Purpose Input (GPI) mode, the pin is configured as an input, with full software control of the pad and external interrupt capability.

34.7.4.1 Overview

GPI mode is set on a particular pin by clearing the MODE and DDR bits in the pin's corresponding CONFIG*xx* register (See Section 34.8.2, "Using a Pin in GPIO Mode," for an example).

In GPI mode, the following register bits are unused:

- SLEW (CONFIG*xx*[4])
- SLEWDIS (CONFIG*xx*[15])

NOTE

Even though the above bits are unused in this mode, they can still be written to. The new value will take effect once the mode of the pin is switched to GPO mode.

While in GPI mode, the data registers PORTDATA, PORTIR and PINDATA*xx* may be used as follows:

Table 34-50. PIM Register Behavior in GPI Mode

Register	Write	Read
PORTDATA	Writing to the corresponding bit in the PORTDATA register will not drive the value onto the pin, but it will set/clear the bit in the register. When the pin is switched to General Purpose Output mode, the new value will be driven onto the pin.	Reading from the PORTDATA register will return the value driven on the pin, synchronized to the system clock. This is identical to reading the corresponding bit in the PORTIR register.
PORTIR	Writing to the PORTIR register will have no effect, as it is a read-only register in all modes.	Reading from the PORTIR register will return the value on the pin. Note that this value is internally synchronized to the system clock, and that there is a two clock cycle delay between the changing of the value on the pin, and the update of the PORTIR register.
PINDATA*xx*	Writing to the PINDATA*xx* register is identical to writing the corresponding bit in the PORTDATA register.	Reading from the PINDATA*xx* register is identical to reading the corresponding bit from the PORTDATA register.

Figure 34-20 illustrates the anatomy of a single PIM pad cell in GPI mode.

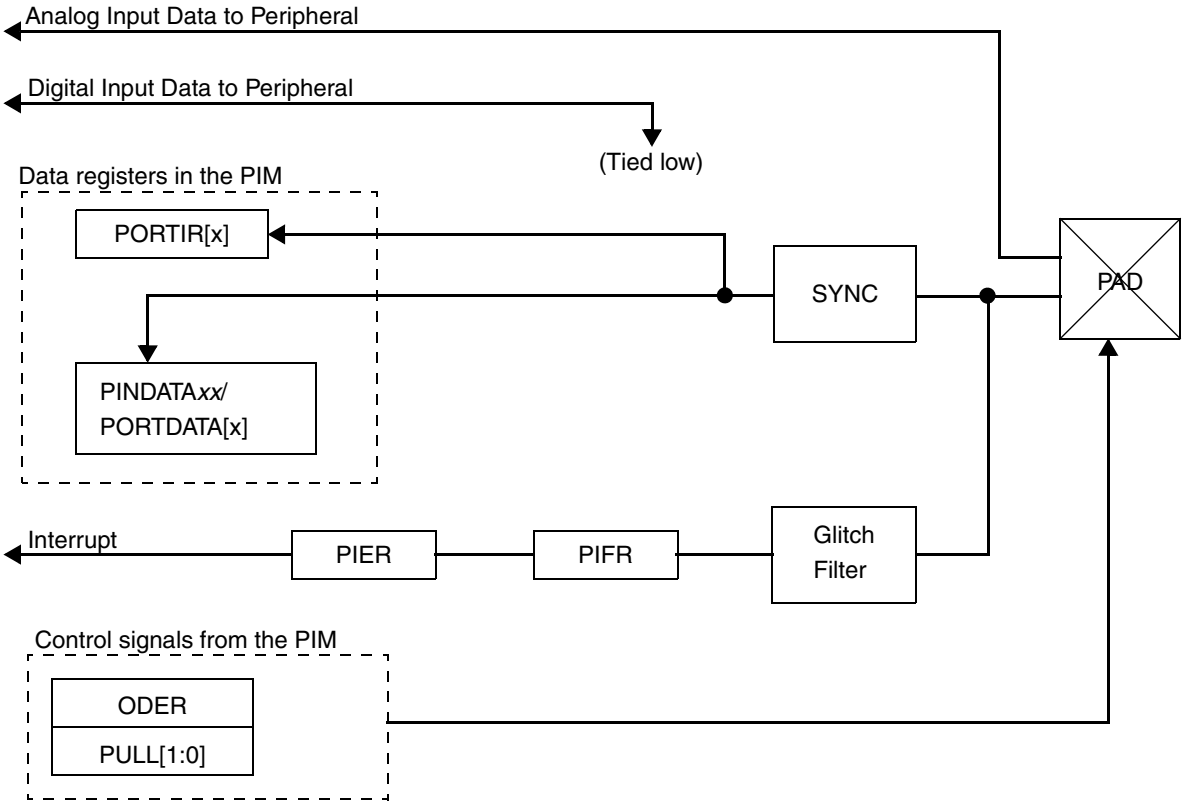


Figure 34-20. Pad in GPI Mode (including pad PD2)

NOTE

The ODER bit and PORTDATA register are not available on Port E.

34.7.4.2 Interrupts

When configured as a General Purpose Input (GPI), a pin may also be used to handle external edge sensitive interrupts. This section describes how to use this functionality.

A pin may be used as an external interrupt by configuring the PIER and PULL[1:0] bits in the appropriate CONFIG*xx* register as follows:

Table 34-51. Interrupt Polarity Configuration

Interrupt Type	PULL[1:0]
Active High (Rising Edge)	00
Active High with pull-down (Rising Edge)	10
Active Low (Falling Edge)	01
Active Low with pull-up (Falling Edge)	11

Detection of an external interrupt may be done in one of two ways:

- Polling - You may choose to detect an external interrupt by continuously reading the CONFIG*xx* register until the PIFR bit is set. Alternatively, you may also read the PORTIFR register to get all pending external interrupts for the entire port. To use this method, clear the PIER bit in the CONFIG*xx* register.
- Interrupt Driven - If this method is chosen, by setting the PIER bit in the CONFIG*xx* register, a system interrupt will be generated when an external interrupt occurs (with a minimum delay - See Table 34-52). If so configured, this system interrupt can force the processor to take an exception, where the interrupt may be cleared and serviced.

In either case, once an external interrupt has been recognized, it must be cleared before the next external interrupt (i.e.-the next edge) will be recognized on that pin. This is done by writing a ’1’ to the PIFR bit in the CONFIG*xx* register. Writing a ’0’ will not clear the interrupt flag.

There are two different ways in which interrupts can be organized in order to minimize the latency of the interrupt service routine.

1. Place all external interrupts on a single port. If all interrupts are on a single port, then the exact interrupt source can be determined by reading the Port Wide Interrupt Flag (PORTIFR) register for that port.
2. Use multiple ports, but place only a single external interrupt on each port. In this scenarios, the exact interrupt source can be determined by reading the Global Interrupt Status (GLBLINT) register.

Interrupts from all Ports are driven into a single interrupt on the Interrupt Controller (Interrupt Request #61).

All 16 pins in each port have a glitch filter, clocked by the system clock, to filter out spurious interrupt signals. Table 34-52 shows the minimum pulse width for an external interrupt/wakeup signal.

Table 34-52. Input Glitch Filter Requirements

System Mode	Minimum time negated before active edge (t_{setup})	Minimum time asserted after active edge (t_{hold})
Run (Normal)	4 x $T_{BusClock}$	4 x $T_{BusClock}$

As an example, in Normal mode, with a 20Mhz bus clock (50ns period), t_{setup} = 200ns and t_{hold} = 200ns. Figure 34-21 illustrates the timing parameters t_{setup} and t_{hold}. Six (6) bus clock cycles after the active edge (assuming no filtered glitches), the PIFR bit will be set in the corresponding CONFIG*xx* register.

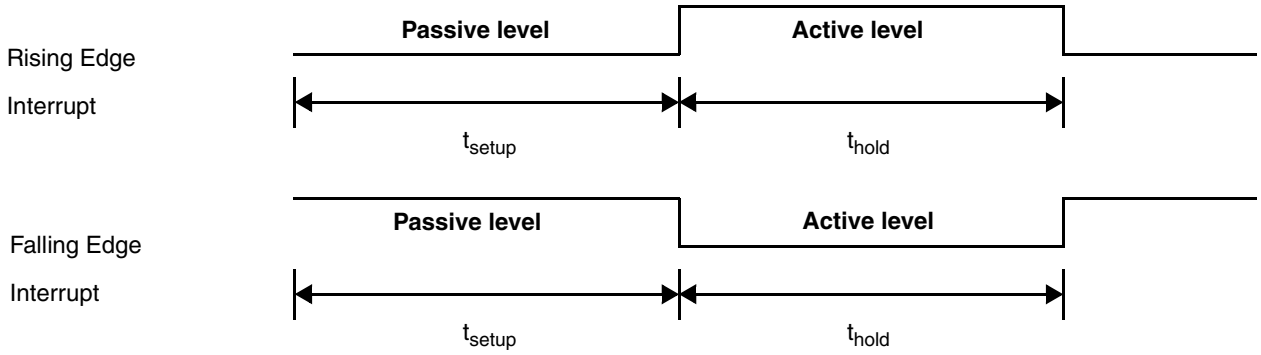


Figure 34-21. External Interrupt Timing Requirements

NOTE

The above illustration is for timing purposes only. Short (in duration) changes during setup or hold will be filtered out by the glitch filter.

34.7.5 General Purpose Output mode

In General Purpose Output (GPO) mode, the pin is configured as an output, with full software control of the pad.

34.7.5.1 Overview

GPO mode is set on a particular pin by clearing the MODE bit and setting the DDR bit in the pin's corresponding CONFIG*xx* register (See Section 34.8.2, "Using a Pin in GPIO Mode," for an example).

In GPO mode, the following register bits are unused:

- PIER (CONFIG*xx*[1])
- PIFR (CONFIG*xx*[0])

NOTE

Even though the above bits are unused in this mode, they can still be written to. The new values will take effect once the mode of the pin is switched to GPI mode.

While in GPO mode, the data registers PORTDATA**,** PORTIR and PINDATA*xx* may be used as follows:

Table 34-53. PIM Register Behavior in GPO Mode

Register	Write	Read
PORTDATA	Writing to the corresponding bit in the PORTDATA register will drive that value onto the pin.	Reading from the PORTDATA register will return the value in the PORTDATA register, which should also be the value driven onto the pin.
PORTIR	Writing to the PORTIR register will have no effect, as it is a read-only register in all modes.	Reading from the PORTIR register will return the value on the pin. Note that this value is internally synchronized to the system clock, and that there is a two clock cycle delay between the changing of the value on the pin, and the update of the PORTIR register.
PINDATA*xx*	Writing to the PINDATA*xx* register is identical to writing the corresponding bit in the PORTDATA register.	Reading from the PINDATA*xx* register is identical to reading the corresponding bit from the PORTDATA register. Note that for pad PD2 only, it is not possible to read back the value driven onto the pad using the PINDATA or PORTDATA register when in GPO mode.

Figure 34-22 illustrates the anatomy of a single PIM pad cell in GPO mode, while Figure 34-23 illustrates the anatomy of the pad PD2 cell in GPO mode.

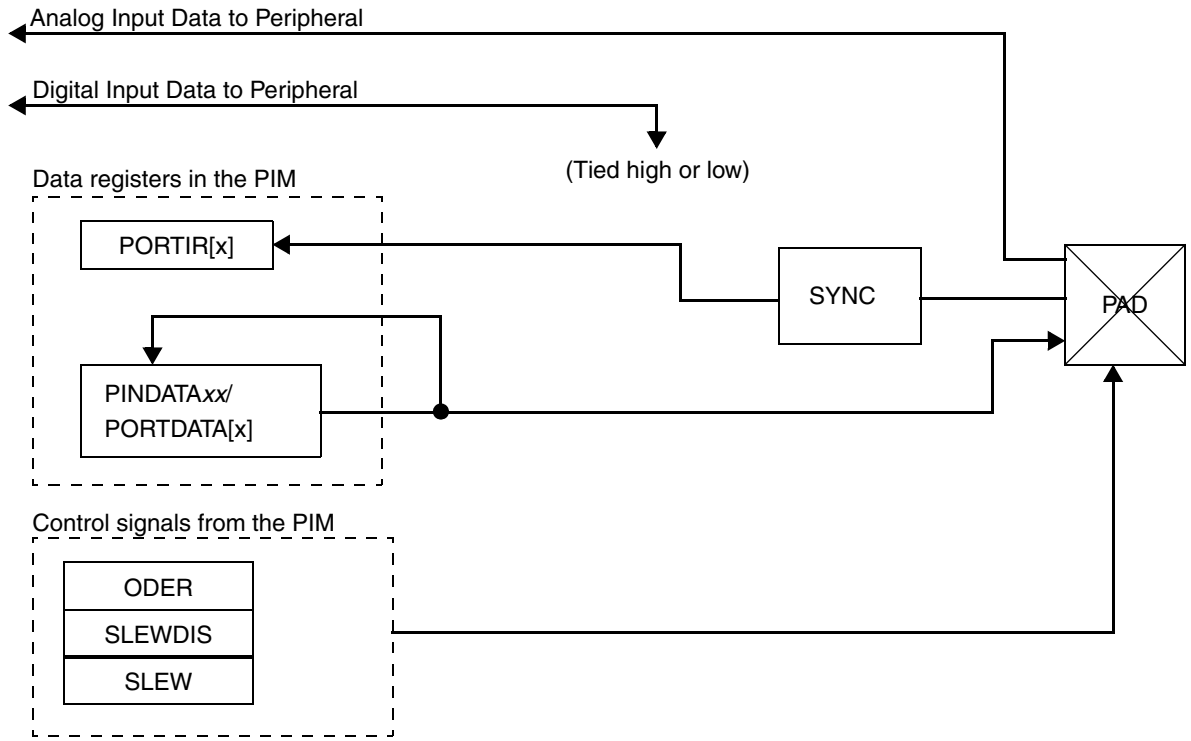


Figure 34-22. Pad in GPO Mode (except pad PD2)

NOTE

GPO Mode is not available on Port E.

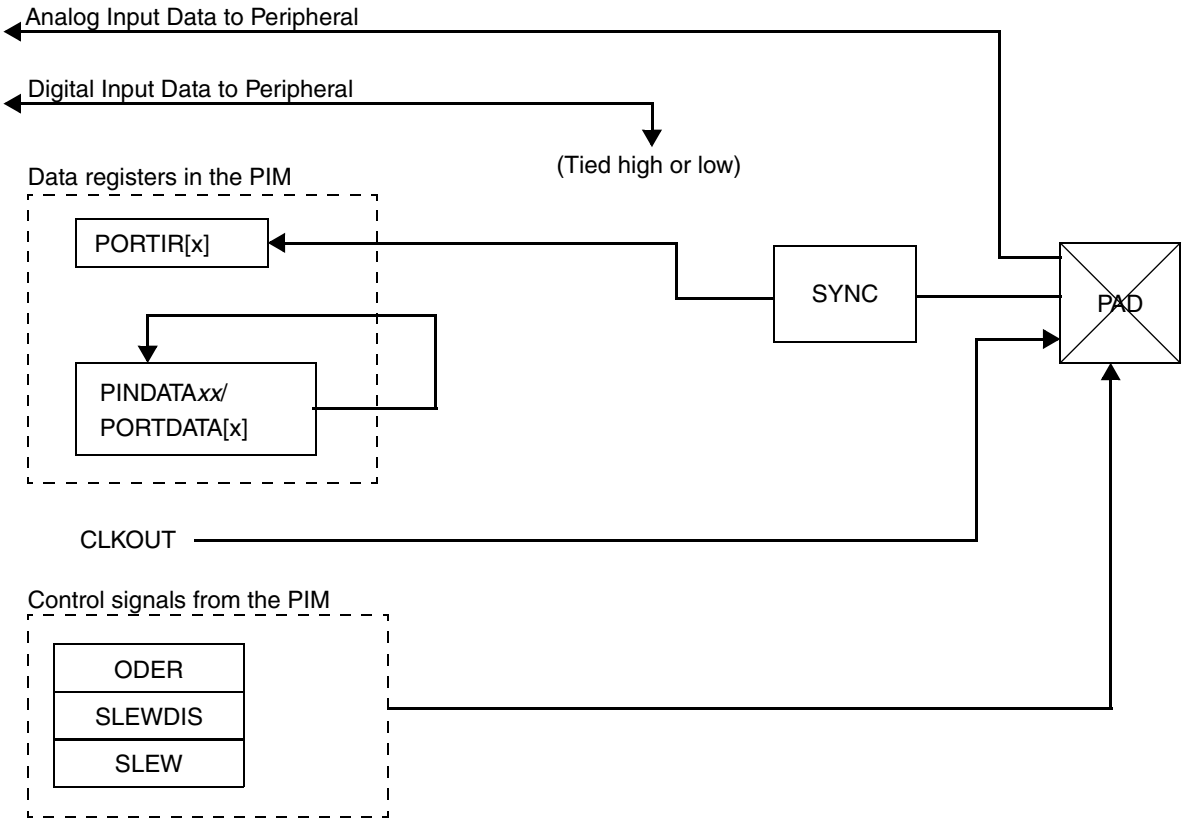


Figure 34-23. Pad PD2 in GPO Mode

34.7.6 PIM Integration Hints

The purpose of the Port Integration Module (PIM) is to implement the GPIO control registers, as well as select between the port controls of the peripheral (when in Peripheral Mode), and the port controls driven by the GPIO registers (when in GPIO Mode). The following figures illustrate this concept.

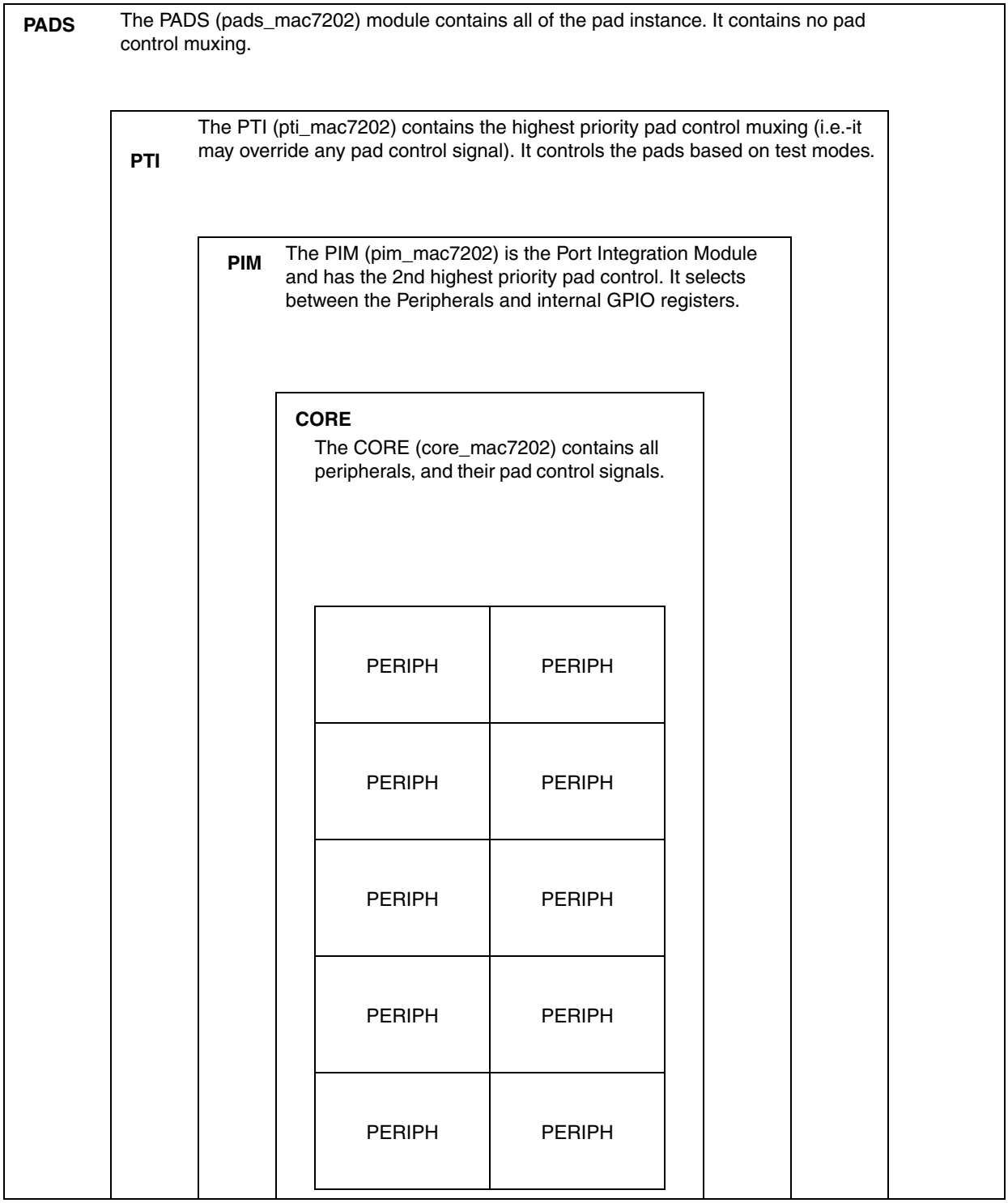


Figure 34-24. I/O Pad Control (Overview)

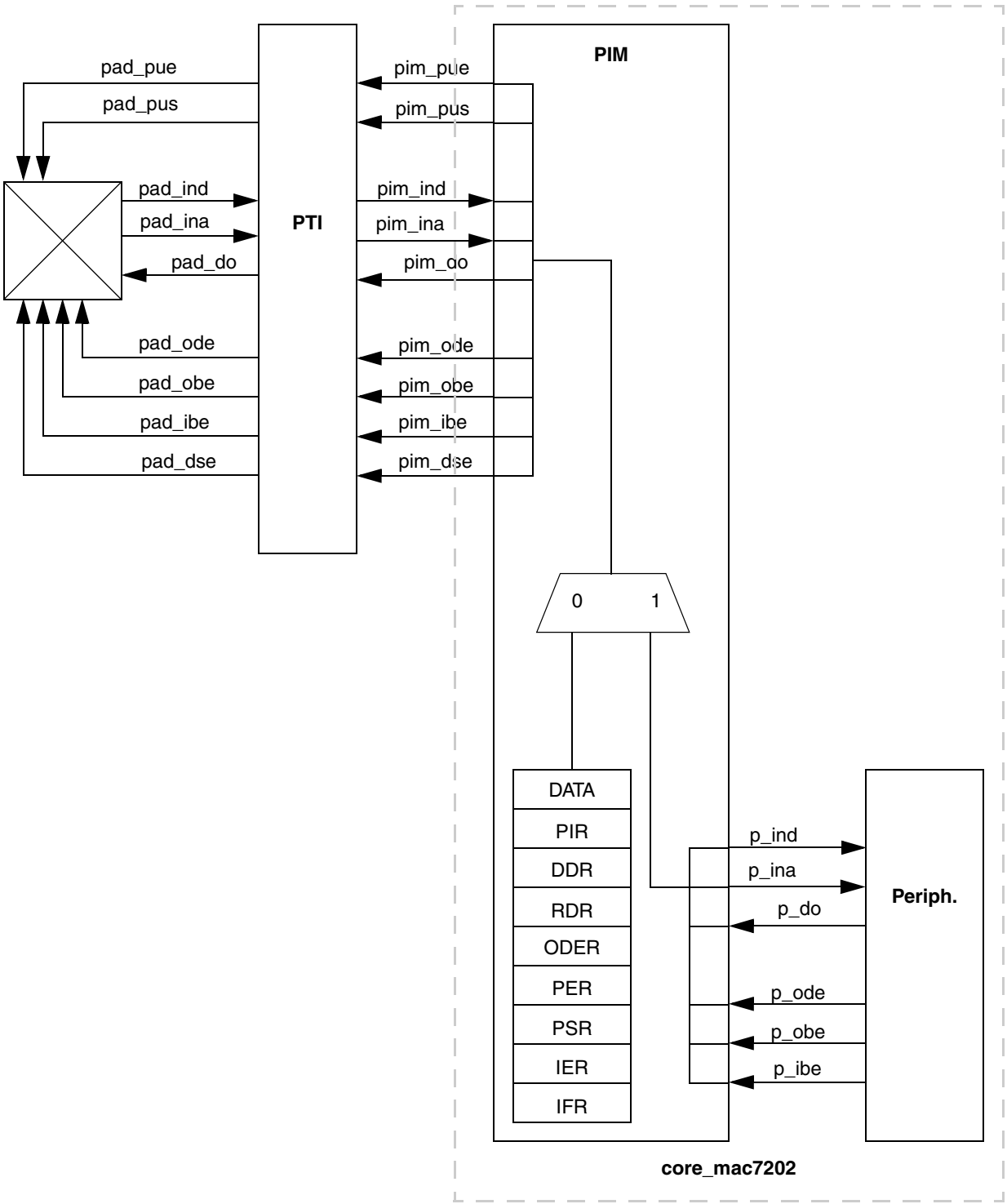


Figure 34-25. I/O Pad Control (Detailed View)

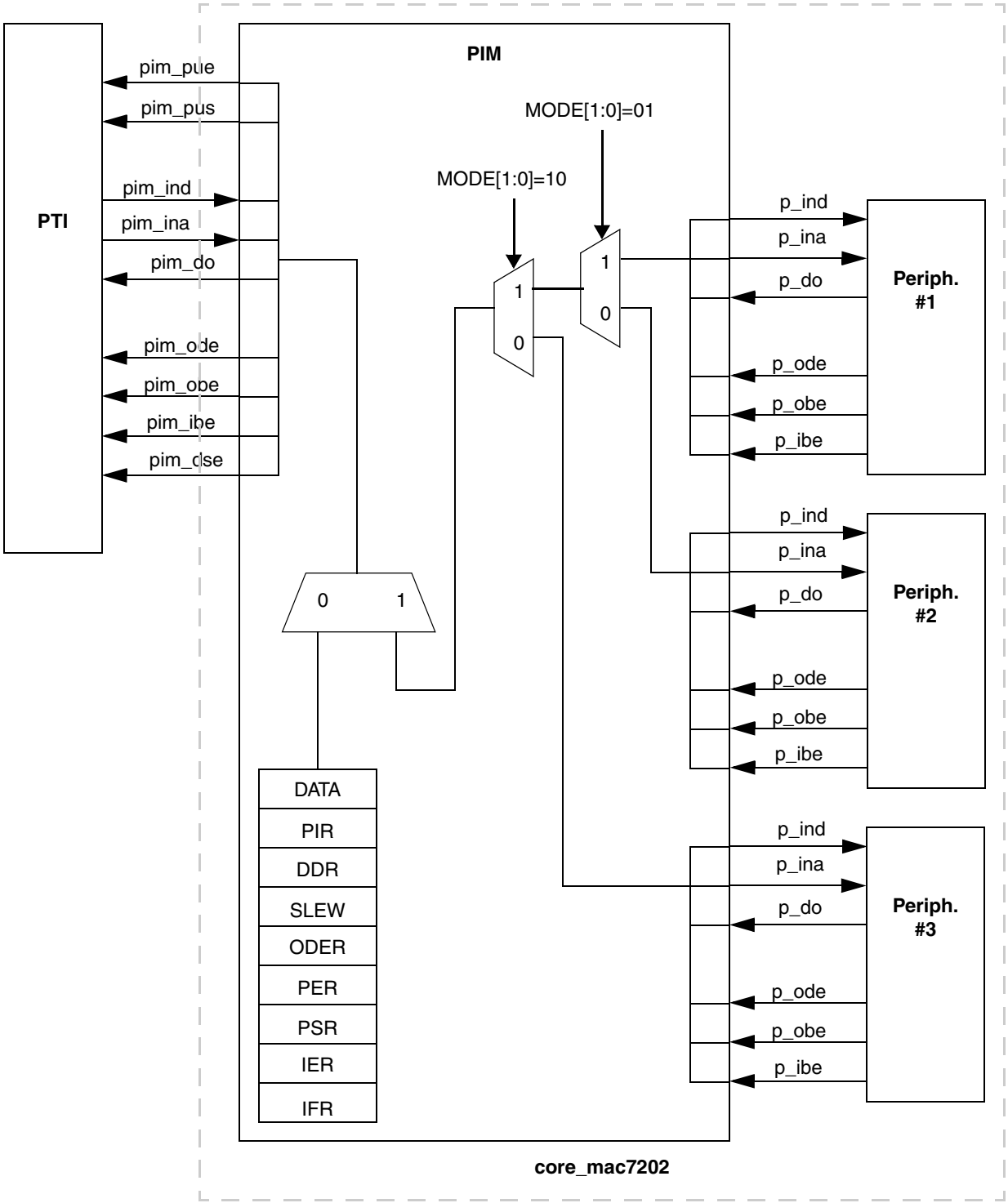


Figure 34-26. PIM Peripheral Muxing

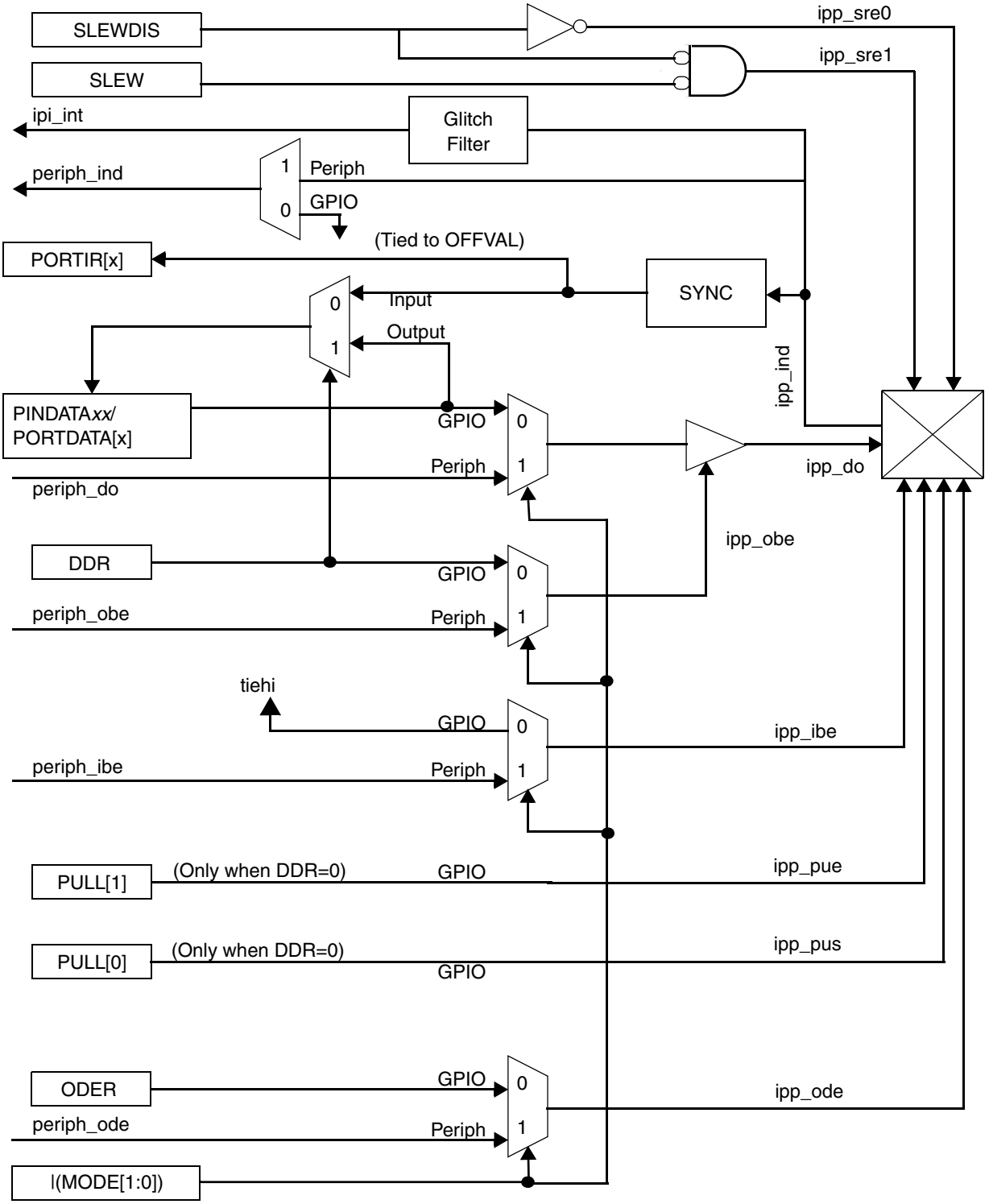


Figure 34-27. I/O Pad Control Cell Architecture (in PIM core)

34.8 Initialization/Application Information

34.8.1 Using a Pin in Peripheral Mode

To use a pin in Peripheral Mode, follow these steps:

1. Using Table 34-6, determine the Port number(s) for the desired peripheral pin(s). Note that, in most cases, *all* pins associated with a peripheral should be switched to Peripheral Mode for proper operation of the peripheral.
2. Using Table 34-11, determine the correct CONFIG register(s) for the selected port number(s).
3. Determine whether a pull-up or pull-down is required.
4. Write 0x80, 0x88 or 0x8C to the selected CONFIG register(s). Note that, in Peripheral Mode, the values of all bits except the MODE and PULL[1:0] bits are unused.
5. If required, enable the peripheral. See the documentation for the peripheral for more information on how to do this.

Example 34-1. Enable the IIC module

1. For the IIC module, enable both the SDA and SCL pins. Using Table 34-6, determine that these pins are located on Port B0 and B1, respectively.
2. Using Table 34-11, determine the addresses for Port BO and B1 to be ’Base + 0x0040’ and ’Base + 0x0042’, respectively.
3. Write 0x008C to ’Base + 0x0040’ and 0x008C to ’Base + 0x0042’. Alternatively, it is possible to write 0x0080 to both addresses (no Pull-up functionality). Use the following example code to do this:

```
In File registers.h:
    #define PIM_BASE_ADDRESS     0xFC0E8000/* Example only ! */
    /* Following example assumes short is 16-bits */
    volatile unsigned short *CONFIG_B0 = (volatile unsigned short *)
(PIM_BASE_ADDRESS+0x0040);
    volatile unsigned short *CONFIG_B1 = (volatile unsigned short *)
(PIM_BASE_ADDRESS+0x0042);

In File main.c:
    #include "registers.h"
        :
        :
    *CONFIG_B0 = 0x008C;  // Pull-up enabled
    *CONFIG_B1 = 0x008C;  // Pull-up enabled
```

4. If required, enable the IIC module by writing to the appropriate register in the IIC register map.

34.8.2 Using a Pin in GPIO Mode

34.8.2.1 Initialization

To use a pin in GPIO Mode, follow these steps:

1. Determine which peripheral(s) will not be used. The corresponding pins of the selected peripheral(s) may be used for GPIO. Although it is possible to dynamically switch between Peripheral and GPIO Modes, there is a 2 cycle latency that must be accounted for, and it is generally recommended that switching is done only during initialization of the part (i.e.-statically).
2. If required, disable the peripheral. See the documentation for the particular peripheral for more information on how to do this.
3. Using Table 34-6, determine the Port number(s) for the desired peripheral pin(s). Note that, in most cases, *all* pins associated with a peripheral should be switched to GPIO Mode to avoid any possible spurious inputs to the peripheral.
4. Using Table 34-11, determine the correct CONFIG register(s) for the selected port number(s).
5. Determine the proper configuration for the pin(s)

   Output Pin
   - Clear the MODE bit (Pin is in GPIO mode)
   - Set the DDR bit
   - Open drain required ? (Set the ODER bit accordingly)
   - Pull-up or Pull-down required ? (Set the PULL[1:0] bits accordingly)
   - What slew rate is required ? (Set the SLEWDIS and SLEW bits accordingly)

   Input Pin
   - Clear the MODE bit (Pin is in GPIO mode)
   - Clear the DDR bit
   - Pull-up or Pull-down required ? (Set the PULL[1:0] bits accordingly)
   - Is pin used as an interrupt or system wakeup ? (Set the PIER/PIFR bits accordingly)
6. Write the proper configuration to the selected CONFIG register(s). Note that, in GPIO Mode, all unreserved bits in the CONFIG register are used, and should be properly configured.

Example 34-2. Disable the IIC module and use the IIC pins to implement an output control signal and an external interrupt/wakeup.

1. For the IIC module, switch both the **SDA** and **SCL** pins to GPIO Mode. Using Table 34-6, determine that these pins are located on Port B0 and B1, respectively.
2. If required, disable the IIC module by writing to the appropriate register in the IIC register map.
3. Using Table 34-11, determine the addresses for Port BO and B1 to be ’Base + 0x040’ and ’Base + 0x042’, respectively.
4. Intended use for Port B0: an output control signal with the following characteristics:
   - No open drain
   - Fast slew rate

This gives a CONFIG register value of 0x0040

5. Intended use for Port B1: an input interrupt/wakeup signal with the following characteristics:

   — Active high interrupt with pull-down

   This gives us a CONFIG register value of 0x000B

   (Write ’1’ to the PIFR register to ensure that any pending interrupts are cleared)

6. Write 0x0040 to ’Base + 0x040’ and 0x000B to ’Base + 0x042’. Use the following example code to do this:

```
In File registers.h:
    #define PIM_BASE_ADDRESS     0xFC0E8000/* Example only ! */
    /* Following example assumes short is 16-bits */
    volatile unsigned short *CONFIG_B0 = (volatile unsigned short *)
(PIM_BASE_ADDRESS+0x0040);
    volatile unsigned short *CONFIG_B1 = (volatile unsigned short *)
(PIM_BASE_ADDRESS+0x0042);

In File main.c:
    #include "registers.h"
        :
        :
    *CONFIG_B0 = 0x0040;
    *CONFIG_B1 = 0x000B;
```

34.8.2.2 Accessing Data

For maximum flexibility, the value of signals may be driven or sampled both on a pin basis and a port (16-pin) basis, by the processor core or via a DMA. This section details how to drive and sample pins, as well as several examples.

34.8.2.2.1 Driving/Sampling an Entire Port

To drive or sample and entire port (16 pins), you must first configure the pins, following the steps outlined in Section 34.8.2.1, “Initialization.” Once this is done, you may drive a value onto the pins (output mode) or sample values from the pins (input mode) by writing or reading the appropriate PORTDATA register. Note that writing to pins that are specified as inputs will have no effect. Conversely, reading from pins that are specified as outputs will simply return the last value written into the PORTDATA register. In this manner, it is possible to use the PORTDATA register to drive or sample less than the full 16 pins in a port. The following example illustrates this point.

Example 34-3. Configure Port B to use the IIC Module, 10 outputs and 4 inputs. Sample the 4 inputs and drive 0x3FF onto the 10 outputs.

```
In File registers.h:
        #define PIM_BASE_ADDRESS     0xFC0E8000/* Example only ! */
        /* Following example assumes short is 16-bits */
        volatile unsigned short *CONFIG_B0 = (volatile unsigned short *) (PIM_BASE_ADDRESS+0x0040);
        volatile unsigned short *CONFIG_B1 = (volatile unsigned short *) (PIM_BASE_ADDRESS+0x0042);
        volatile unsigned short *CONFIG_B2 = (volatile unsigned short *) (PIM_BASE_ADDRESS+0x0044);
        volatile unsigned short *CONFIG_B3 = (volatile unsigned short *) (PIM_BASE_ADDRESS+0x0046);
        volatile unsigned short *CONFIG_B4 = (volatile unsigned short *) (PIM_BASE_ADDRESS+0x0048);
        volatile unsigned short *CONFIG_B5 = (volatile unsigned short *) (PIM_BASE_ADDRESS+0x004A);
```

```
volatile unsigned short *CONFIG_B6 = (volatile unsigned short *) (PIM_BASE_ADDRESS+0x004C);
volatile unsigned short *CONFIG_B7 = (volatile unsigned short *) (PIM_BASE_ADDRESS+0x004E);
volatile unsigned short *CONFIG_B8 = (volatile unsigned short *) (PIM_BASE_ADDRESS+0x0050);
volatile unsigned short *CONFIG_B9 = (volatile unsigned short *) (PIM_BASE_ADDRESS+0x0052);
volatile unsigned short *CONFIG_B10 = (volatile unsigned short *) (PIM_BASE_ADDRESS+0x0054);
volatile unsigned short *CONFIG_B11 = (volatile unsigned short *) (PIM_BASE_ADDRESS+0x0056);
volatile unsigned short *CONFIG_B12 = (volatile unsigned short *) (PIM_BASE_ADDRESS+0x0058);
volatile unsigned short *CONFIG_B13 = (volatile unsigned short *) (PIM_BASE_ADDRESS+0x005A);
volatile unsigned short *CONFIG_B14 = (volatile unsigned short *) (PIM_BASE_ADDRESS+0x005C);
volatile unsigned short *CONFIG_B15 = (volatile unsigned short *) (PIM_BASE_ADDRESS+0x005E);
volatile unsigned short *PORTDATA_B = (volatile unsigned short *) (PIM_BASE_ADDRESS+0x0060);
```

In File **main.c**:

```
#include "registers.h"
          :
          :
/* Initialize Port B */
*CONFIG_B0 = 0x0080;/* Peripheral Mode */
*CONFIG_B1 = 0x0080;/* Peripheral Mode */
*CONFIG_B2 = 0x0040;/* GPIO Mode: Output */
*CONFIG_B3 = 0x0040;/* GPIO Mode: Output */
*CONFIG_B4 = 0x0040;/* GPIO Mode: Output */
*CONFIG_B5 = 0x0040;/* GPIO Mode: Output */
*CONFIG_B6 = 0x0040;/* GPIO Mode: Output */
*CONFIG_B7 = 0x0040;/* GPIO Mode: Output */
*CONFIG_B8 = 0x0040;/* GPIO Mode: Output */
*CONFIG_B9 = 0x0040;/* GPIO Mode: Output */
*CONFIG_B10 = 0x0040;/* GPIO Mode: Output */
*CONFIG_B11 = 0x0040;/* GPIO Mode: Output */
*CONFIG_B12 = 0x0000;/* GPIO Mode: Input, No interrupt */
*CONFIG_B13 = 0x0000;/* GPIO Mode: Input, No interrupt */
*CONFIG_B14 = 0x0000;/* GPIO Mode: Input, No interrupt */
*CONFIG_B15 = 0x0000;/* GPIO Mode: Input, No interrupt */
/* Technically speaking, the above four lines of code are
   not necessary because the reset value of all CONFIG
   registers is 0x0000. They are shown here for the purposes
   of illustration. */

/* Sample the 4 input pins */
unsigned int value;
/* Mask out bits 0-11 and shift into bit positions 0-4.
   It is necessary to mask bits 0-11 because a read from
   PORTDATA on output ports will return the value of the
   bit in the PORTDATA register */
value = (*PORTDATA_B & 0xF000) >> 11;

/* drive the 10 output pins */
/* Shift our desired value '0x3FF' into bit positions 2-11.
   It is not necessary to mask out any other bits (i.e.-bits
   0-1 and 12-15) because writes to pins in GPIO Input or
   Peripheral Mode have no effect. However, these writes do
   change the value in the PORTDATA register, and therefore
   will be reflected in the next read of the PORTDATA register. */
*PORTDATA_B = 0x3FF < 2;
```

34.8.2.2.2 Driving/Sampling Individual Pins

As can be seen from the above example, when using the PORTDATA register, it becomes necessary to perform mask and shift operations when sampling or driving data. For smaller numbers of pins, this can

be rather inefficient, and can not easily be programmed into a DMA. The next example illustrates how to use the Pin Data Register (PINDATA*xx*) to overcome these two disadvantages.

NOTE

The PINDATA*xx* register is simply a "mirror" of the corresponding bit in the PORTDATA register. Therefore writes to the PORTDATA register will change the read value of a PINDATA*xx* register, and vice-versa.

Example 34-4. Disable the IIC and use its two pins as general purpose inputs

```
In File registers.h:
          #define PIM_BASE_ADDRESS     0xFC0E8000/* Example only ! */
          /* Following example assumes 'char' is 8-bits and 'short' is 16-bits */
          volatile unsigned short *CONFIG_B0 = (volatile unsigned short *) (PIM_BASE_ADDRESS+0x0040);
          volatile unsigned short *CONFIG_B1 = (volatile unsigned short *) (PIM_BASE_ADDRESS+0x0042);
                    :
                    :
          volatile unsigned char *PINDATA_B0 = (volatile unsigned char *) (PIM_BASE_ADDRESS+0x0064);
          volatile unsigned char *PINDATA_B1 = (volatile unsigned char *) (PIM_BASE_ADDRESS+0x0065);


In File main.c:
          #include "registers.h"
                    :
                    :
          /* Initialize Port B */
          *CONFIG_B0 = 0x0000;/* GPIO Mode: Input, No interrupt */
          *CONFIG_B1 = 0x0000;/* GPIO Mode: Input, No interrupt */

          /* Sample Ports B0 & B1 */
          unsigned char value_B0;
          unsigned char value_B1;
          value_B0 = *PINDATA_B0;/* Equivalent to (*PORTDATA_B & 0xF001) >> 0 */
          value_B1 = *PINDATA_B1;/* Equivalent to (*PORTDATA_B & 0x0002) >> 1 */
```

Like the PORTDATA register, writes to the PINDATA registers for input pins have no effect (other than to set the corresponding bit in the PORTDATA register). Therefore, it is possible to use a DMA to drive multiple pins, even though the pins may not be contiguous (i.e.-Pins B0-B3 and B5-B8). Conversely, reads from the PINDATA registers for output pins will simply return the value of the corresponding bit in the PORTDATA register. The next section illustrates how to use a DMA with trigger capability to implement periodic protocols (such as PWM) on the GPIO ports.

34.8.2.2.3 Using a DMA

As seen in the previous section, it is relatively easy to drive or sample multiple pins without the need for mask or shift operations, using the PINDATA*xx* registers. This allows the use of a DMA to copy entire tables of data from/to several pins at a time. In addition to standard DMA operations, the Port Integration Module also supports the use of DMA triggers to perform periodic copies of data, allowing you to easily implement protocols such as Pulse Width Modulation. For more details on how to setup and configure DMA triggers, please refer to the section covering the DMA Channel Mux. In general, the procedure is as follows:

1. Configure the Port Integration Module
2. Configure the DMA to copy tables of data from memory to the Port Integration Module

3. Configure DMA triggers in the Periodic Interrupt Timer
4. In the DMA Channel Mux, configure one of the available DMA channels to be sourced from an "always enabled" source

Example 34-5. Drive Ports G0 - G2 every 3µs with data from memory for 129µs

1. Ensure that ESCI_A and ESCI_B are disabled
2. Configure Ports G0,G1 and G2 as outputs (**CONFIG*xx***=0x0040)
3. Configure the DMA to copy 3-byte tables from memory. This can be done by using a minor loop count of 3 and a major loop count equal to the number of tables of data to be driven. Data is to be driven for 129µs, with a 3µs delay between each table, so 44 tables of data are required. Therefore our major counter will be 44.
4. Configure the Periodic Interrupt Timer to set a period of 3µs
5. Configure the DMA Channel Mux to connect an open "always enabled" source and to enable the DMA trigger functionality for that channel.

34.8.2.3 Using Interrupts

To use a pin as an external interrupt, follow these steps:

1. Determine which peripheral(s) will not be used. The corresponding pins of the selected peripheral(s) may be used as external interrupts. Although it is possible to dynamically switch between Peripheral and GPIO Modes, there is a 2 cycle latency that must be accounted for, and it is generally recommended that switching is done only during initialization of the part (i.e.-statically).
2. If required, disable the peripheral. See the documentation for the particular peripheral for more information on how to do this.
3. Using Table 34-6, determine the Port number(s) for the desired peripheral pin(s). Note that, in most cases, *all* pins associated with a peripheral should be switched to GPIO Mode to avoid any possible spurious inputs to the peripheral.
4. Using Table 34-11, determine the correct CONFIG register(s) for the selected port number(s).
5. Determine the proper configuration for the pin(s)
   - Clear the MODE bit (Pin is in GPIO mode)
   - Clear the DDR bit (Pin is an input)
   - Set the PIER bit (Interrupt is not masked) if you want the processor to be interrupted
   - Interrupt polarity (See Table 34-12, Table 34-14 and Table 34-51)
6. Write the proper configuration to the selected CONFIG register(s). Note that, in GPIO Mode, all unreserved bits in the CONFIG register are used, and should be properly configured.
7. Once an interrupt occurs on *any* pin enabled as an interrupt, the processor will receive an interrupt (if the PIER bit was set for that particular pin). The Interrupt Service Routine (ISR) should determine the exact source of the interrupt as follows:

— Read the IPR registers in the system interrupt controller to determine that the Port Integration Module caused the interrupt. If IPRH[29] is set, then there is an interrupt pending on one or more Ports.

— Read the GLBLINT register to determine which Ports have interrupts pending.

— Read the PORTIFR register *for each port* to determine which pin(s) have pending interrupts. Note that there may be multiple interrupts pending at any given time.

8. After the exact source(s) of the interrupt have been determined, the ISR should clear all pending interrupts by writing 0xFF to the PORTIFR register for each Port. In order to prevent any lost interrupts, you should do the read and clear of the PORTIFR register (or PIFR bit in the CONFIG register) for each port one port at a time (versus reading all PORTIFR registers, and then writing all PORTIFR registers).

Example 34-6. Disable the eMIOS module and use all eMIOS pins as external interrupts.

1. Using Table 34-6, determine that the eMIOS pins are located on Port F0 to F7.
2. If required, disable the eMIOS module by writing to the appropriate register in the eMIOS register map (typically the MDIS bit).
3. Using Table 34-11, determine the addresses for Port F0 through F7 CONFIG registers to be ’Base + 0x140’ to ’Base + 0x14E’.
4. Use all pins as an input interrupt/wakeup signal with the following characteristics:

   — Active low interrupt with pull-up

   This gives a CONFIG register value of 0x000F

   (Write ’1’ to the PIFR register to ensure that any pending interrupts are cleared)
5. Write 0x000F to ’Base + 0x100’ through ’Base + 0x11E’. Use the following example code to do this:

```
In File registers.h:
#define PIM_BASE_ADDRESS     0xFC0E8000/* Example only ! */
/* Following example assumes short is 16-bits */
volatile unsigned short *CONFIG_F0 = (volatile unsigned short *) (PIM_BASE_ADDRESS+0x0140);
volatile unsigned short *CONFIG_F1 = (volatile unsigned short *) (PIM_BASE_ADDRESS+0x0142);
volatile unsigned short *CONFIG_F2 = (volatile unsigned short *) (PIM_BASE_ADDRESS+0x0144);
volatile unsigned short *CONFIG_F3 = (volatile unsigned short *) (PIM_BASE_ADDRESS+0x0146);
volatile unsigned short *CONFIG_F4 = (volatile unsigned short *) (PIM_BASE_ADDRESS+0x0148);
volatile unsigned short *CONFIG_F5 = (volatile unsigned short *) (PIM_BASE_ADDRESS+0x014A);
volatile unsigned short *CONFIG_F6 = (volatile unsigned short *) (PIM_BASE_ADDRESS+0x014C);
volatile unsigned short *CONFIG_F7 = (volatile unsigned short *) (PIM_BASE_ADDRESS+0x014E);


In File main.c:
#include "registers.h"
:
:
*CONFIG_F0 = 0x000F;
*CONFIG_F1 = 0x000F;
*CONFIG_F2 = 0x000F;
*CONFIG_F3 = 0x000F;
*CONFIG_F4 = 0x000F;
*CONFIG_F5 = 0x000F;
*CONFIG_F6 = 0x000F;
*CONFIG_F7 = 0x000F;
```

6. When receiving an interrupt, determine if the interrupt was generated by the Port Integration Module. In this case read and clear the PORTIFR registers for each port to determine which interrupts are pending. The following example code illustrates this *assuming all pins on all ports are used as external interrupts*:

```
In File registers.h:
#define PIM_BASE_ADDRESS     0xFC0E8000/* Example only ! */
/* Following example assumes short is 16-bits */
volatile unsigned short *PORTIFR_A = (volatile unsigned short *) (PIM_BASE_ADDRESS+0x0020);
volatile unsigned short *PORTIFR_B = (volatile unsigned short *) (PIM_BASE_ADDRESS+0x0060);
volatile unsigned short *PORTIFR_C = (volatile unsigned short *) (PIM_BASE_ADDRESS+0x00A0);
volatile unsigned short *PORTIFR_D = (volatile unsigned short *) (PIM_BASE_ADDRESS+0x00E0);
volatile unsigned short *PORTIFR_F = (volatile unsigned short *) (PIM_BASE_ADDRESS+0x0160);
volatile unsigned short *PORTIFR_G = (volatile unsigned short *) (PIM_BASE_ADDRESS+0x01A0);


In File main.c:
#include "registers.h"
:
:
unsigned short ifr_port_a;
unsigned short ifr_port_b;
unsigned short ifr_port_c;
unsigned short ifr_port_d;
unsigned short ifr_port_f;
unsigned short ifr_port_g;

ifr_port_a = *PORTIFR_A;/* Determine which interrupt sources are pending on Port A*/
*PORTIFR_A = 0xFFFF;/* Clear all pending interrupts on Port A */

ifr_port_b = *PORTIFR_B;/* Determine which interrupt sources are pending on Port B*/
*PORTIFR_B = 0xFFFF;/* Clear all pending interrupts on Port B */

ifr_port_c = *PORTIFR_C;/* Determine which interrupt sources are pending on Port C*/
*PORTIFR_C = 0xFFFF;/* Clear all pending interrupts on Port C */

ifr_port_d = *PORTIFR_D;/* Determine which interrupt sources are pending on Port D*/
*PORTIFR_D = 0xFFFF;/* Clear all pending interrupts on Port D */

ifr_port_f = *PORTIFR_F;/* Determine which interrupt sources are pending on Port F*/
*PORTIFR_F = 0xFFFF;/* Clear all pending interrupts on Port F */

ifr_port_g = *PORTIFR_G;/* Determine which interrupt sources are pending on Port G*/
*PORTIFR_G = 0xFFFF;/* Clear all pending interrupts on Port G */
```

34.8.3 Using the PD2 pad (CLKOUT)

34.8.3.1 Enabling the CLKOUT output

To enable the CLKOUT clock on pad PD2, do the following:

1. Put pin PD2 into Peripheral Mode (Set the MODE bit in the CONFIG2 register for Port D)

   or

   Put pin PD2 into General Purpose Output (GPO) Mode (Clear the MODE bit and set the DDR bit in the CONFIG2 register for Port D)

2. Select the desired slew rate by setting or clearing the SLEWDIS and SLEW bits in the CONFIG2 register for Port D. The reset (default) value is fast slew rate.

Note that, when the external bus is used to boot the device (i.e.-the System is in Expanded Mode), the CLKOUT clock output is automatically enabled. Otherwise, the CLKOUT clock output is disabled, but may be manually enabled via the above steps.

34.8.3.2 Disabling the CLKOUT output

To disable the CLKOUT clock on pad PD2, simply configure pad PD2 to be a General Purpose Input, as described in Section 34.8.3.3, "Using pad PD2 as a General Purpose Input."

34.8.3.3 Using pad PD2 as a General Purpose Input

To enable pin PD2 as a General Purpose Input (and therefore disable the CLKOUT clock), do the following:

- Put pin PD2 into GPIO Mode (Clear the MODE bit in the CONFIG2 register for Port D)
- Put pin PD2 into Input Mode (Clear the DDR bit in the CONFIG2 register for Port D)

Note that, when the external bus is not used to boot the device (i.e.-the System is not in Expanded Mode), the PD2 pin is automatically configured as a General Purpose Input.

34.8.3.4 Using pad PD2 as a General Purpose Output

It is not possible to use pad PD2 as a General Purpose Output. Pad PD2 behaves virtually identically if it is configured as a General Purpose Output or in Peripheral Mode. In Peripheral Mode, the open drain functionality is disabled; In GPO Mode, the open drain functionality may be enabled/disabled via the ODER bit.

34.8.4 Unbonded Pins

All port pins (Port A, B, C, D, F, G) that are not bonded out on a particular package should be placed in GPO mode to reduce power consumption. Otherwise the floating inputs might result in unneeded current draw.

Chapter 35
Test Controller (PTI)

35.1 Test Controller Introduction

The MAC72xx Test Controller (PTI) performs the following tasks:

- Implementation of JTAG Test Register 4
- Control of Lockout Recovery sequence

35.1.1 JTAG Test Register (SC4)

The SC4 Test Register is directly accessible from the JTAG interface, and includes the following functionality:

- JTAG Lockout Recovery
- Device ID
- Debug Reset
- Chip Status
- Core Run Control
- This register is customer visible

The SC4 Test Register can be accessed as follows:

Perform a JTAG SCAN_N instruction with the following values:

— ID Number = 4

— Register Length = 23 bits

— For writes to SC4, the WRITE bit (bit 22) must be set.

— For reads from SC4, the WRITE bit (bit 22) must be cleared.

The SC4 Test Register contains the following bits:

Table 35-1. SC4 Test Register Field Definitions

Field	Reset Value	Function
0	1	Core is running
4:1	n/a	Device ID (Read only)
5	0	Reset debug logic
7:6	0	(reserved)
8	n/a	Flash Lockout Recovery is complete (Read only)

Table 35-1. SC4 Test Register Field Definitions (Continued)

Field	Reset Value	Function
9	0	Start Flash Lockout Recovery
18:10	0	(reserved)
19	n/a	Flash Security Request (Read only)
21:20	n/a	Chip mode (Read only)
22	0	Write

Table 35-2. SC4 Test Register Field Descriptions

Field	Description
0	Core is running When this bit is set, the core is running. When it is cleared, the core is suspended by negating the **CLKEN** input of the ARM7 core. This bit may be written or read in any mode, with TEST = 0 or 1. It is used primarily in MBIST or NVMBIST modes to prevent any possible interaction between the core and the BIST engines.
1–4	Device ID (Read only) These four bits contain the last four digits of the device ID.
5	Reset debug logic When this bit is set, it causes the system debug reset to be asserted. This resets the following logic: • EICE (**DBGnTRST** input on the ARM7 core) • Test Registers SC4, SC5, SC6 and SC7 and associated logic • JTAG synchronization logic Since setting this bit in JTAG register SC4 causes JTAG Register SC4 itself to be cleared, you must not perform any other operations during this write. This bit may be written while $\overline{\text{RESET}}$ is asserted, but the value will not take effect until after $\overline{\text{RESET}}$ has been negated. Resetting of the Nexus registers is not done via this bit. In order to reset the Nexus registers, you must perform either a Power On Reset (POR) or cycle the ARM7 TAP State Machine through the Test Reset state.
6–7	(reserved)
8	Flash Lockout Recovery is complete (Read only) This read-only bit will be set once the JTAG Lockout Recovery sequence has completed. It does not indicate whether the sequence was successful (this is indicated by bit 19), but is used to determine when the sequence has completed and the status of the chip security may be read out. After the JTAG Lockout Recovery has completed a reset must be issued to reload the security of the device from the shadow.
9	Start Flash Lockout Recovery Setting this bit will start the JTAG Lockout Recovery sequence. Note that it is necessary to manually clear this bit once the sequence has started or completed.
10–18	(reserved)
19	Flash Security Request (Read only)
20–21	Chip mode (Read only)
22	Write

35.1.2 JTAG Lockout Recovery

JTAG Lockout Recovery is performed solely in hardware on the MAC72xx. Please refer to Chapter 20, “Boot Assist Module (BAM)” for more details on JTAG Lockout Recovery.

35.2 Test Controller External Pins

Table 35-3. Test Controller External Pins

Signal	Description
TEST	This input only pin is reserved for test. The pin must be tied to VSS in all applications. If this pin is shorted to a VDD supply or left open, the operation of the device will be unpredictable.

35.3 Test Controller Application Usage

No further information is required to use the Test Controller.

Appendix A
Electrical Characteristics

This section contains electrical information for MAC7200 family microcontrollers. The information is preliminary and subject to change without notice.

A.1 Parameter Classification

The electrical parameters shown in this appendix are derived by various methods. To provide a better understanding to the designer, the following classification is used. Parameters are tagged accordingly in the column labeled “C” of the parametric tables, as appropriate.

Table A-1. Parametric Value Classification

P	Parameters guaranteed during production testing on each individual device.
C	Parameters derived by the design characterization and by measuring a statistically relevant sample size across process variations.
T	Parameters derived by design characterization on a small sample size from typical devices under typical conditions (unless otherwise noted). All values shown in the typical column are within this classification, even if not so tagged.
D	Parameters derived mainly from simulations.

A.2 Absolute Maximum Ratings

Absolute maximum ratings are stress ratings only. Functional operation outside these maximums is not guaranteed. Stress beyond these limits may affect reliability or cause permanent damage to the device.

MAC7200 family devices contain circuitry protecting against damage due to high static voltage or electrical fields; however, it is advised that normal precautions be taken to avoid application of any voltages higher than maximum-rated voltages to this high-impedance circuit. Reliability of operation is enhanced if unused inputs are tied to an appropriate logic voltage level (for example, either $V_{SS}5$ or $V_{DD}5$).

Table A-2. Absolute Maximum Ratings

Num	Rating	Symbol	Min	Max	Unit
A1	I/O, Regulator and Analog Supply Voltage	$V_{DD}5$	–0.3	+5.5	V
A2	Digital Logic Supply Voltage[1]	$V_{DD}1.5$	–0.3	+1.65	V
A3	PLL Supply Voltage [1]	$V_{DD}PLL$	–0.3	+3.6	V
A4	Analog Supply Voltage (reference to $V_{SS}A$)	$V_{DD}A$	–0.3	+5.5	V
A5	Analog Reference High Voltage (reference to V_{RL})	V_{RH}	–0.3	+5.5	V

Table A-2. Absolute Maximum Ratings (Continued)

Num	Rating	Symbol	Min	Max	Unit
A6	Voltage difference $V_{DD}X$ to $V_{DD}A$	Δ_{VDDX}	–0.1	+0.1	V
A7	Voltage difference $V_{SS}X$ to $V_{SS}A$	Δ_{VDDX}	–0.1	+0.1	V
A8	V_{REF} Differential Voltage	$V_{RH} - V_{RL}$	-0.3	5.5	V
A9	V_{RH} to V_{DDA} Differential Voltage	$V_{RH} - V_{DD}A$	-5.5	5.5	V
A10	V_{RL} to V_{SSA} Differential Voltage	$V_{RL} - V_{SSA}$	-0.3	0.3	V
A11	Digital I/O Input Voltage	V_{IN}	–0.3	+5.5	V
A12	XFC, EXTAL, XTAL inputs	V_{ILV}	–0.3	+3.6	V
A13	TEST input	V_{TEST}	–0.3	+5.5	V
A14	Instantaneous Maximum Current[2]				
A15	Storage Temperature Range	T_{stg}	–55	+150	°C

1. The device contains an internal voltage regulator to generate the logic and PLL supply from the I/O supply. The absolute maximum ratings apply when the device is powered from an external source.
2. Input must be current limited to the value specified. To determine the value of the required current-limiting resistor, use the larger of the calculated values using $V_{POSCLAMP} = V_{DD}A + 0.3V$ and $V_{NEGCLAMP} = –0.3$ V.

A.3 ESD Protection and Latch-up Immunity

All ESD testing is in conformity with CDF-AEC-Q100 Stress test qualification for Automotive Grade Integrated Circuits. During the device qualification ESD stresses were performed for the Human Body Model (HBM), the Machine Model (MM) and the Charge Device Model (CDM).

A device is defined as a failure if after exposure to ESD pulses the device no longer meets the device specification. Complete DC parametric and functional testing is performed per the applicable device specification at room temperature followed by hot temperature, unless specified otherwise.

Table A-3. ESD and Latch-up Test Conditions

Model	Description	Symbol	Value	Unit
Human Body	Series Resistance	R1	1500	Ohm
	Storage Capacitance	C	100	pF
	Number of Pulses per pin positive negative	—	— 3 3	

Table A-3. ESD and Latch-up Test Conditions (Continued)

Model	Description	Symbol	Value	Unit
Latch-up	Minimum input voltage limit		–2.5	V
	Minimum current limit		I_{nom}+100 or 1.5 x I_{nom}[1]	mA
	Maximum input voltage limit		7.5	V
	Maximum current limit		-100 or -0.5 x I_{nom}[1]	mA

1. Whichever value is the greater. I_{nom} is the nominal value of current.

Table A-4. ESD and Latch-Up Protection Characteristics

Num	C	Rating	Symbol	Min	Typ	Max	Unit
B1	C	Human Body Model (HBM)	V_{HBM}	2000	—	—	V
B2	C	Charge Device Model (CDM)	V_{CDM}	500 (all pins)	—	—	V
				750 (corner pins)			
B3	C	Latch-up Current at T_A = 125°C positive negative	I_{LAT}	I_{nom} + 10 mA or 1.4 x I_{nom}[1]	—	—	mA

1. Whichever value is the greater. I_{nom} is the nominal value of current.

A.4 Operating Conditions

Unless otherwise noted, the following conditions apply to all parametric data. Refer to the temperature rating of the device (C, V, M) with respect to ambient temperature (T_A) and junction temperature (T_J). For power dissipation calculations refer to Section A.5, “Power Dissipation and Thermal Characteristics”

Table A-5. MAC7200 Family Device Operating Conditions

Num	C	Rating	Symbol	Min	Typ	Max	Unit
C1	D	I/O, Regulator and Analog Supply Voltage	$V_{DD}5$	4.5	5.0	5.5	V
C2	D	Digital Logic Supply Voltage[1]	$V_{DD}15$	1.45	1.5	1.65	V
C3	D	PLL Supply Voltage [1]	$V_{DD}PLL$	3.0	3.3	3.6	V
C4	D	Voltage Difference $V_{DD}X$ to $V_{DD}A$	$\Delta_{VDD}X$	–0.1	0	0.1	V
C5	D	Voltage Difference $V_{SS}X$ to $V_{SS}A$	$\Delta_{VSS}X$	–0.1	0	0.1	V
C6a	C	Oscillator Frequency (ALC Mode)	f_{OSC}	4.0	—	20 (MAC72x2) 40 (MAC72x1)	MHz
C6b	C	Oscillator Frequency (External Clock Mode)	f_{OSC}	0.5	—	70	MHz

Table A-5. MAC7200 Family Device Operating Conditions (Continued)

Num	C	Rating		Symbol	Min	Typ	Max	Unit
C7	D	System Clock Frequency		f_{SYS}	0.5	—	70	MHz
C8	D	Peripheral Bus Frequency		f_{PERIPH}	0.25	—	35	MHz
C9a	D	MAC72xx**C**	Operating Junction Temperature Range[2]	T_J	–40	—	110	°C
C9b	D		Operating Ambient Temperature Range [2]	T_A	–40	25	85	°C
C10a	D	MAC72xx**V**	Operating Junction Temperature Range [2]	T_J	–40	—	130	°C
C10b	D		Operating Ambient Temperature Range [2]	T_A	–40	25	105	°C
C11a	D	MAC72xx**M**	Operating Junction Temperature Range [2]	T_J	–40	—	150	°C
C11b	D		Operating Ambient Temperature Range [2]	T_A	–40	25	125	°C

1. The device contains an internal voltage regulator to generate the logic and PLL supply from the I/O supply. The absolute maximum ratings apply when this regulator is disabled and the device is powered from an external source.
2. Please refer to Section A.5, “Power Dissipation and Thermal Characteristics” for more details about the relation between ambient temperature T_A and device junction temperature T_J.

A.4.1 Input/Output Pins

The I/O pins operate at a nominal level of 5V. This class of pins is comprised of the clocks, RESET, TEST, JTAG interface, and general purpose/peripheral pins (i.e.-Port A-G). The internal structure of these pins is identical; however, some functionality may be disabled (for example, on all ATD analog inputs on Port E, the output drivers are permanently disabled).

This section describes the characteristics of all I/O pins under 5V operating conditions. All parameters are not always applicable; for example, not all pins feature pull up/down resistances.

Table A-6. 5V I/O Characteristics

Conditions shown in Table A-5 unless otherwise noted							
Num	**C**	**Rating**	**Symbol**	**Min**	**Typ**	**Max**	**Unit**
D1	P	Input High Voltage	V_{IH}	$0.65 \times V_{DD}5$[1]	—	$V_{DD}5 + 0.3$	V
D2	P	Input Low Voltage	V_{IL}	$V_{SS}5 - 0.3$	—	$0.35 \times V_{DD}5$	V
D3	D	Input Hysteresis	V_{HYS}	No Hysteresis			V
D4	P	Input Leakage Current (pins in high impedance input mode)[2] $V_{in} = V_{DD}5$ or $V_{SS}5$	I_{in}	–2.5	—	2.5	µA
D5	P	Output High Voltage (pins in output mode) Drive I_{OH} = –2mA	V_{OH}	$0.8 \times V_{DD}5$	—	—	V
D6	P	Output Low Voltage (pins in output mode) Drive I_{OL} = +2mA	V_{OL}	—	—	$0.2 \times V_{DD}5$	V
D7	P	Internal Pull Up Device Current, tested at V_{IL} Max.	I_{PUL}	—	—	150	µA

Table A-6. 5V I/O Characteristics (Continued)

Conditions shown in Table A-5 unless otherwise noted							
D8	P	Internal Pull Up Device Current, tested at V_{IH} Min.	I_{PUH}	15	—	—	µA
D9	P	Internal Pull Down Device Current, tested at V_{IH} Min.	I_{PDH}	—	—	150	µA
D10	P	Internal Pull Down Device Current, tested at V_{IL} Max.	I_{PDL}	15	—	—	µA
D11	D	Input Capacitance	C_{in}	—	2	—	pF

1. Refer to Section A.6, “Power Supply” for definition of $V_{SS}5$ and $V_{DD}5$.
2. Maximum leakage current occurs at maximum operating temperature. Current decreases by approximately one-half for each 8°C to 12°C in the temperature range from 50°C to 125°C.

A.4.2 Oscillator Pins

The pins EXTAL and XTAL are dedicated to the oscillator and operate from V_{DD}PLL at a nominal level of 3.3V.

Table A-7. Oscillator Characteristics

Num	C	Rating	Symbol	Min	Typ	Max	Unit
E1	D	Input Capacitance (EXTAL, XTAL pins)	C_{IN}	—	—	5	pF
E2	D	EXTAL Pin Input High Voltage[1]	$V_{IH,EXTAL}$	0.75 x V_{DD}PLL	—	V_{DD}PLL+ 0.4	V
E3	D	EXTAL Pin Input Low Voltage[1]	$V_{IL,EXTAL}$	V_{DD}PLL - 0.4	—	0.25 x V_{DD}PLL	V
E4	D	EXTAL Pin Input Hysteresis[1]	$V_{HYS,EXTAL}$		200		mV

1. In External Clock Model.

A.4.3 PLL Pins

The pin XFC is dedicated to the PLL and operates at a nominal level of 3.3V.

A.5 Power Dissipation and Thermal Characteristics

Power dissipation and thermal characteristics are closely related. The user must assure that the maximum operating junction temperature is not exceeded.

Note that the JEDEC specification reserves the symbol $R_{\theta JA}$ or θ_{JA} (Theta-JA) strictly for junction-to-ambient thermal resistance on a 1s test board in natural convection environment. $R_{\theta JMA}$ or θ_{JMA} (Theta-JMA) will be used for both junction-to-ambient on a 2s2p test board in natural convection and for junction-to-ambient with forced convection on both 1s and 2s2p test boards. It is anticipated that the generic name, θ_{JA}, will continue to be commonly used.

The average chip-junction temperature (T_J) in °C is obtained from:

$$T_J = T_A + P_D \cdot \Theta_{JA} \qquad \textit{Eqn. A-1}$$

T_J = Junction Temperature (°C)

T_A = Ambient Temperature (°C)

P_D = Total Chip Power Dissipation (W)

Θ_{JA} = Package Thermal Resistance (°C/W)

The total power dissipation is calculated from:

$$P_D = P_{INT} + P_{IO} \quad \textit{Eqn. A-2}$$

P_{INT} = Chip Internal Power Dissipation (W)

P_{IO} = Input / Output Power Dissipation (W)

Two cases must be considered for P_{INT}:

1. Internal Voltage Regulator enabled:

$$P_{INT} = (I_{DD}R \times V_{DD}R) + (I_{DD}A \times V_{DD}A) \quad \textit{Eqn. A-3}$$

2. Internal voltage regulator disabled ($V_{DD}R = V_{SS}R$ = system ground):

$$P_{INT} = (I_{DD}1.5 \times V_{DD}1.5) + (I_{DD}3.3 \times V_{DD}3.3) + (I_{DD}PLL \times V_{DD}PLL) + (I_{DD}A \times V_{DD}A) \quad \textit{Eqn. A-4}$$

P_{IO} is the sum of all output currents on I/O ports associated with $V_{DD}X$ and $V_{DD}R$.:

$$P_{IO} = \sum_i R_{DSON} \cdot (I_{IO_i})^2 \quad \textit{Eqn. A-5}$$

where

$$R_{DSON} = \frac{V_{OL}}{I_{OL}} \text{ (for outputs driven low)} \quad \textit{Eqn. A-6}$$

or

$$R_{DSON} = \frac{V_{DD}X - V_{OH}}{I_{OL}} \text{ (for outputs driven high)} \quad \textit{Eqn. A-7}$$

$I_{DD}R$ is the current shown in Table A-11 and not the overall current flowing into $V_{DD}R$, which additionally contains the current flowing into the external loads with output high.

A.5.1 Thermal Resistance Simulation Details

Table A-8. Thermal Resistance for 100 lead 14x14 mm LQFP, 0.5 mm Pitch[1]

Rating			Value	Unit
Junction to Ambient (Natural Convection)[2, 3]	Single layer board (1s)	$R_{\theta JA}$	55	°C/W
Junction to Ambient (Natural Convection)[2, 4]	Four layer board (2s2p)	$R_{\theta JMA}$	42	°C/W
Junction to Ambient (@ 200 ft./min.)[2, 4]	Single layer board (1s)	$R_{\theta JMA}$	45	°C/W
Junction to Ambient (@ 200 ft./min.)[2, 4]	Four layer board (2s2p)	$R_{\theta JMA}$	36	°C/W
Junction to Board[5]		$R_{\theta JB}$	28	°C/W
Junction to Case[6]		$R_{\theta JC}$	11	°C/W
Junction to Package Top[7]	Natural Convection	Ψ_{JT}	2	°C/W

1. 100 LQFP, Case Outline: 983–02
2. Junction temperature is a function of die size, on-chip power dissipation, package thermal resistance, mounting site (board) temperature, ambient temperature, air flow, power dissipation of other components on the board, and board thermal resistance.
3. Per SEMI G38-87 and JEDEC JESD51-2 with the single layer board (JESD51-3) horizontal.
4. Per JEDEC JESD51-6 with the board (JESD51-7) horizontal.
5. Thermal resistance between the die and the printed circuit board per JEDEC JESD51-8. Board temperature is measured on the top surface of the board at the center lead. For fused lead packages, the adjacent lead is used.
6. Thermal resistance between the die and the case top surface as measured by the cold plate method (MIL SPEC-883 Method 1012.1).
7. Thermal characterization parameter indicating the temperature difference between package top and junction temperature per JEDEC JESD51-2. When Greek letters are not available, the thermal characterization parameter is written as Psi-JT.

Table A-9. Thermal Resistance for 144 lead 20x20 mm LQFP, 0.5 mm Pitch[1]

Rating			Value	Unit
Junction to Ambient (Natural Convection)[2, 3]	Single layer board (1s)	$R_{\theta JA}$	TBD	°C/W
Junction to Ambient (Natural Convection)[2, 4]	Four layer board (2s2p)	$R_{\theta JMA}$	TBD	°C/W
Junction to Ambient (@ 200 ft./min.)[2, 4]	Single layer board (1s)	$R_{\theta JMA}$	TBD	°C/W
Junction to Ambient (@ 200 ft./min.)[2, 4]	Four layer board (2s2p)	$R_{\theta JMA}$	TBD	°C/W
Junction to Board[5]		$R_{\theta JB}$	TBD	°C/W
Junction to Case[6]		$R_{\theta JC}$	TBD	°C/W
Junction to Package Top[7]	Natural Convection	Ψ_{JT}	TBD	°C/W

1. 144 LQFP, Case Outline: 918–03
2. Junction temperature is a function of die size, on-chip power dissipation, package thermal resistance, mounting site (board) temperature, ambient temperature, air flow, power dissipation of other components on the board, and board thermal resistance.
3. Per SEMI G38-87 and JEDEC JESD51-2 with the single layer board (JESD51-3) horizontal.
4. Per JEDEC JESD51-6 with the board (JESD51-7) horizontal.
5. Thermal resistance between the die and the printed circuit board per JEDEC JESD51-8. Board temperature is measured on the top surface of the board at the center lead. For fused lead packages, the adjacent lead is used.
6. Thermal resistance between the die and the case top surface as measured by the cold plate method (MIL SPEC-883 Method 1012.1).
7. Thermal characterization parameter indicating the temperature difference between package top and junction temperature per JEDEC JESD51-2. When Greek letters are not available, the thermal characterization parameter is written as Psi-JT.

NOTE

The 144LQFP packages are not currently qualified; any future qualification will be based upon customer demand and will be subject to specified leadtimes. This package option is supported for limited samples only, and does not include burn-in testing.

Table A-10. Power Dissipation 1/8 Simulation Model Packaging Parameters

Component	Conductivity
Mold Compound	0.96 W/m K
Die Attach	1.5 W/m K

A.6 Power Supply

The MAC7200 family utilizes several pins to supply power to the oscillator, PLL, digital core, I/O ports and ATD. In the context of this section, the following conventions apply:

- $V_{DD}5$ is used for $V_{DD}A$, $V_{DD}R$ or $V_{DD}X$; $V_{SS}5$ is used for $V_{SS}A$, $V_{SS}R$ or $V_{SS}X$ unless otherwise noted. $I_{DD}5$ denotes the sum of the currents flowing into the $V_{DD}A$, $V_{DD}X$, and $V_{DD}R$.
- $V_{DD}33$ is used for $V_{DD}3.3$, and $V_{DD}PLL$, $V_{SS}3$ is used for $V_{SS}3.3$ and $V_{SS}PLL$. $I_{DD}33$ is used for the sum of the currents flowing into $V_{DD}3.3$ and $V_{DD}PLL$.
- V_{DD} is used for $V_{DD}1.5$, V_{SS} is used for $V_{SS}1.5$. I_{DD} is used for the current flowing into $V_{DD}1.5$.

A.6.1 Current Injection

The power supply must maintain regulation within the $V_{DD}5$, $V_{DD}33$ or V_{DD} operating range during instantaneous and operating maximum current conditions. If positive injection current ($V_{in} > V_{DD}5$) is greater than $I_{DD}5$, the injection current may flow out of V_{DD5} and could result in the external power supply going out of regulation. It is important to ensure that the external $V_{DD}5$ load will shunt current greater than the maximum injection current. The greatest risk will be when the MCU is consuming very little power (for example, if no system clock is present, or if the clock rate is very low).

A.6.2 Power Supply Pins

The device has the following power supply pins.

- The $V_{DD}R$ – $V_{SS}R$ pair supplies the internal voltage regulator.
- The $V_{DD}A$ – $V_{SS}A$ pair supplies the ATD converter and the reference circuit of the internal voltage regulator.
- The $V_{DD}X$ – $V_{SS}X$ pairs supply the I/O pins. The following $V_{DD}X$ and $V_{SS}X$ pins are internally connected by metal
  - $V_{DD}X3$, $V_{DD}X5$, $V_{DD}X8$, $V_{DD}X9$
  - $V_{DD}X1$, $V_{DD}X2$, $V_{DD}X4$, $V_{DD}X6$, $V_{DD}X7$, $V_{DD}X10$, $V_{DD}X11$
  - $V_{SS}X3$, $V_{SS}X5$, $V_{SS}X8$, $V_{SS}X9$
  - $V_{SS}X1$, $V_{SS}X2$, $V_{SS}X4$, $V_{SS}X6$, $V_{SS}X7$, $V_{SS}X10$, $V_{SS}X11$
- The $V_{DD}PLL$ – $V_{SS}PLL$ pair supplies the oscillator and PLL.
- The $V_{DD}3.3$ – $V_{SS}3.3$ pair supplies the pre-drivers for the I/O pins and the on-chip Flash.
- The $V_{DD}1.5$ – $V_{SS}1.5$ pairs supply the core logic. All $V_{DD}1.5$ pins are internally connected by metal. All $V_{SS}1.5$ pins are internally connected by metal.

$V_{DD}A$, $V_{DD}X$ and $V_{DD}R$ as well as $V_{SS}A$, $V_{SS}X$ and $V_{SS}R$ are connected by ESD protection diodes.

A.6.3 Supply Current Characteristics

Table A-11 lists the supply current characteristics for MAC72xx devices operating at 70 MHz.

All current measurements are without output loads. Unless otherwise noted the currents are measured in single chip mode, internal voltage regulator enabled and at 70MHz bus frequency. Production testing isperformed using a square wave signal at the EXTAL input.

In expanded modes, the currents flowing in the system are highly dependent on the load at the address, data and control signals as well as on the duty cycle of those signals. No generally applicable numbers can be given. A good estimate is to take the single chip currents and add the currents due to the external loads.

Table A-11. Supply Current Characteristics

Conditions shown in Table A-5, with f_{SYS} = 70 MHz.							
Num	**C**	**Rating**		**Symbol**	**Typ**	**Max**	**Unit**
F1a	C	**Run Supply Current** Single Chip	**Core** –40° C	$I_{DD}R_{core}$	75	100	mA
			25° C		75	100	mA
			85° C[1]		75	100	mA
			105° C[1]		75	100	mA
			125° C[1]		75	100	mA
F1b	P		**Regulator** (if enabled) –40° C	$I_{DD}R_{reg}$	70	95	mA
			25° C		70	95	mA
			85° C[1]		70	95	mA
			105° C[1]		70	95	mA
			125° C[1]		70	95	mA
F1c	C		**Pins** –40° C	$I_{DD}R_{pins}$	Determined by the activity on the bus and external pins. Should be characterised by the customer.		mA
			25° C				mA
			85° C[1]				mA
			105° C[1]				mA
			125° C[1]				mA

Table A-11. Supply Current Characteristics (Continued)

Conditions shown in Table A-5, with f_{SYS} = 70 MHz.							
Num	**C**	**Rating**		**Symbol**	**Typ**	**Max**	**Unit**
F2a	?	**Doze Supply Current** Single Chip	**Core** –40° C	$I_{DD}D_{core}$	<=75mA, depending on the customer configuration of modules in DOZE mode and speed of operation.	<=100mA, depending on the customer configuration of modules in DOZE mode and speed of operation.	mA
			Core 25° C				mA
			Core 85° C[1]				mA
			Core 105° C[1]				mA
			Core 125° C[1]				mA
F2b	?		**Regulator** (if enabled) –40° C	$I_{DD}D_{reg}$	<=70mA, depending on the customer configuration of modules in DOZE mode and speed of operation.	<=95mA, depending on the customer configuration of modules in DOZE mode and speed of operation.	mA
			Regulator (if enabled) 25° C				mA
			Regulator (if enabled) 85° C[1]				mA
			Regulator (if enabled) 105° C[1]				mA
			Regulator (if enabled) 125° C[1]				mA
F2c	?		**Pins** –40° C	$I_{DD}D_{pins}$	Determined by the activity on the bus and external pins. Should be characterised by the customer.		mA
			Pins 25° C				mA
			Pins 85° C[1]				mA
			Pins 105° C[1]				mA
			Pins 125° C[1]				mA

1. 85°C, 105°C, and 125°C refer to the "C", "V", and "M" Temperature Options, respectively.

A.7 Voltage Regulator Characteristics

Table A-12. VREG Operating Conditions

Num	C	Characteristic	Symbol	Min	Typical	Max	Unit
G1	D	Input Voltages	V_{VDDRA}	4.5	5.0	5.5	V
G2	D	Output Voltage Core (1.5V)[1]	$V_{DD}15$	1.45	1.5	1.65	V
G3	D	Output Voltage Core (3.3V)[1]	$V_{DD}33$	3.0	3.3	3.6	V

Table A-12. VREG Operating Conditions (Continued)

Num	C	Characteristic	Symbol	Min	Typical	Max	Unit
G4	D	Output Voltage PLL	$V_{DD}PLL$	3.0	3.3	3.6	V
G5	C	Low Voltage Reset 1.5V[2] Assert Level De-assert Level	 $V_{LVR1V5A}$ $V_{LVR1V5D}$	 1.35 1.35	 — —	 — —	V
G6	C	Low Voltage Reset 3.3V[3] Assert Level De-assert Level	 $V_{LVR3V3A}$ $V_{LVR3V3D}$	 3.0 3.0	 — —	 — —	V
G7	C	Low Voltage Reset PLL[4] Assert Level De-assert Level	 $V_{LVRPLLA}$ $V_{LVRPLLD}$	 3.0 3.0	 — —	 — —	V
G8	C	Power On Reset Assert Level De-assert Level	 V_{PORA} V_{PORD}	 0.5 0.5	 — —	 0.9 0.9	 V V
G9	C	Power On Reset 5V Assert Level De-assert Level	 V_{POR5VA} V_{POR5VD}	 2.2 2.2	 — —	 4.0 4.0	 V V

1. 220 nF to 470 nF bypass capacitors should be used on 1.5V and 3.3V outputs.
2. Monitors V_{DD15}
3. Monitors V_{DD33}
4. Monitors V_{DDPLL}

A.7.1 Output Loads

The on-chip voltage regulator is intended to supply the internal logic and oscillator circuits. No external DC load is allowed. Capacitive loads are specified in Table A-13. Capacitors with X7R dielectricum are required.

Table A-13. VREG Recommended Load Capacitances

Rating	Symbol	Min	Typ	Max	Unit
Load Capacitance on each $V_{DD}1.5$ pins[1]	C_{LVDD15}	220	—	660	nF
Load Capacitance on each $V_{DD}3.3$ pins[2]	C_{LVDD33}	220	470	660	nF
Load Capacitance on $V_{DD}PLL$ pin	$C_{LVDDPLL}$	220	470	660	nF
Load Capacitance on $V_{DD}5$ pin	C_{LVDD5}	220	470	660	nF
Ramp rate for the supply voltage $V_{DD}5$	T_{rise}	2.5	—	100	V/ms

1. Refer to the package information for the specific number of $V_{DD}1.5$ pins on various packages.
2. Refer to the package information for the specific number of $V_{DD}3.3$ pins on various packages.

A.8 Oscillator Characteristics

The MAC7200 family features an internal automatic level control (ALC) oscillator with an External Clock Mode. The selection of ALC Mode or External Clock Mode on the oscillator depends on the level of the

XCLKS signal at the rising edge of the RESET signal. Before asserting the oscillator to the internal system clock distribution subsystem, the quality of the oscillator output clock is checked for each start from either power on reset (POR) or reset after a Clock Monitor Failure. t_{CQOUT} specifies the maximum time before switching to the internal self clock mode after POR if a proper oscillator clock output is not detected. The quality check also determines the minimum oscillator start-up time t_{UPOSC}. The device also features a Crystal Monitor. A Crystal Monitor Failure is asserted if the frequency of the incoming clock signal is below the Crystal Monitor Assert Frequency f_{CMFA}.

NOTE

The use of overtone resonators (Ceramic or Crystal) is not recommended.

Table A-14. Oscillator Characteristics

Num	C	Rating	Symbol	Min	Typ	Max	Unit
H1a	C	Crystal oscillator frequency (ALC Mode)[1]	f_{OSC}[2]	4.0	—	20 (MAC72x2) 40 (MAC72x1)	MHz
H1b	C	External Clock Frequency (Oscillator bypassed)[3]	f_{OSC}	0.5	—	70	MHz
H2	C	Crystal oscillator reference duty cycle	t_{DUTY}	40	—	60	%
H3	C	Startup Current	I_{OSC}	1.2	—	—	mA
H4	D	Supply Voltage	V_{DD}PLL	3.0	3.3	3.6	V
H5	D	Supply Current	I_{OSC}	—	—	3	mA
H6	C	Oscillator start-up time (ALC Mode)	t_{UPOSC}	—	5[4]	10[5]	ms
H7	D	Clock Quality check time-out	t_{CQOUT}	0.45	—	2.5	s
H8	D	Clock Monitor Failure Assert Frequency	f_{CMFA}	100	200	400	KHz
H9	D	External Clock Duty Cycle	—	40/60			—
H10	D	External Clock rise time	t_{EXTR}	—	—	1	ns
H11	D	External Clock fall time	t_{EXTF}	—	—	1	ns

1. XCLKS negated (1) during reset
2. If CLKSEL[PLLSEL] is clear then the system clock (f_{SYS}) is equal to f_{OSC}, otherwise it is equal to f_{VCO}. Throughout this document, t_{SYS} is used to specify a unit of time equal to 1 / f_{SYS}.
3. XCLKS asserted (0) during reset
4. f_{osc} = 8 MHz, C_{LOAD} = 16.5 pF, C_{EXTAL} = 33 pF, C_{XTAL} = 33pF.
5. Maximum value is for extreme cases using high Q, low frequency crystals

NOTE

The oscillators on the MAC71xx and MAC72xx are different designs. You should characterize your oscillator design if you switch between the two devices.

A.9 PLL Characteristics

A.9.1 PLL Filter Characteristics

The oscillator provides the reference clock for the PLL. The voltage controlled oscillator (VCO) of the PLL is also the system clock source in self clock mode. In order to operate reliably, care must be taken to select proper values for external loop filter components.

Figure A-1. Basic PLL Functional Diagram

The procedure described below can be used to calculate the resistance and capacitance values using typical values for K_V (VCO Gain) and i_{ch} from Table A-15.

The phase detector relationship is given by:

$$K_\Phi = -|i_{ch}| \cdot K_V \qquad \textit{Eqn. A-8}$$

i_{ch} is the current in tracking mode. The loop bandwidth f_C should be chosen to fulfill the Gardner's stability criteria by at least a factor of 10, a typical value for the stability factor is 50. $\zeta = 0.9$ ensures a good transient response.

$$f_C < \frac{2 \cdot \zeta \cdot f_{REF}}{\pi \cdot (\zeta + \sqrt{1 + \zeta^2})} \frac{1}{10} \rightarrow f_C < \frac{f_{REF}}{4 \cdot 10}; (\zeta = 0.9) \qquad \textit{Eqn. A-9}$$

And finally the frequency relationship is defined as

$$n = \frac{f_{VCO}}{f_{REF}} = 2 \cdot (SYNR + 1) \qquad \textit{Eqn. A-10}$$

With the above inputs the resistance can be calculated as:

$$R = \frac{2 \cdot \pi \cdot n \cdot f_C}{K_\Phi} \qquad \textit{Eqn. A-11}$$

The capacitance C_S can now be calculated as:

$$C_S = \frac{2 \cdot \zeta^2}{\pi \cdot f_C \cdot R} \approx \frac{0.516}{f_C \cdot R}; (\zeta = 0.9) \qquad \textit{Eqn. A-12}$$

The capacitance C_P should be chosen in the range of:

$$C_S \div 20 \leq C_P \leq C_S \div 10 \qquad \textit{Eqn. A-13}$$

The stabilization delays shown in Table A-15 are dependent on PLL operational settings and external component selection (for example, the crystal and XFC filter).

Example A-1. PLL calculations

$$K_V = -140 \frac{MHz}{V}$$

$$K_\Phi = -|5 \cdot 10^{-6}| \cdot \left(-140 \frac{MHz}{V}\right) = 700 \frac{Hz}{V}$$

Using f_{REF} = 4 Mhz and f_{VCO} = 64Mhz, f_C must be smaller than 100 kHz. A value of f_C = 20 kHz is chosen.

SYNR is calculated to be 7.

$$R = \frac{2 \cdot 3.14 \cdot 20 \cdot 20kHz}{700} = 3.59k\Omega$$

A standard value of R=3.3kΩ is chosen.

$$C_S \approx \frac{0.516}{(20 \cdot 10^3) \cdot 3.3 \cdot 10^3} = 7.82nF$$

A value of C_δ = 8.2 nF is chosen.

A.9.2 PLL Characteristics

Table A-15. PLL Characteristics

Num	C	Rating	Symbol	Min	Typ	Max	Unit
I1	D	PLL reference frequency, crystal oscillator range (ALC Mode)[1]	f_{REF}	0.5	—	20 (MAC72x2)	MHz
				0.5	—	40 (MAC72x1)	
I2	D	PLL reference frequency (External Clock Mode)[1]	f_{REF}	0.5	—	35	MHz
I3	P	Self Clock Mode Frequency	f_{SCM}	1	2.5	10	MHz
I4	D	PLL frequency range	f_{PLL}	30	—	70	MHz
I5	D	VCO frequency range	f_{VCO}	30	—	70	MHz
I6	D	Lock Detector transition from Acquisition to Tracking mode	$\|\Delta_{trk}\|$	3	—	4	%[2]
I7	D	Lock Detection	$\|\Delta_{Lock}\|$	0	—	1.5	%[2]
I8	D	Un-Lock Detection	$\|\Delta_{unl}\|$	0.5	—	2.5	%[2]
I9	D	Lock Detector transition from Tracking to Acquisition mode	$\|\Delta_{unt}\|$	6	—	8	%[2]
I10	C	Charge pump current acquisition mode	$\| i_{ch} \|$	—	50	—	µA
I11	C	Charge pump current tracking mode	$\| i_{ch} \|$	—	5	—	µA

1. Min value only applies when the PLL is bypassed. When the PLL is enabled, the PLL frequency range specification is in effect.
2. Percentage deviation from target frequency

A.9.3 Crystal Monitor Time-out

The time-out Table A-16 shows the delay for the crystal monitor to trigger when the clock stops, either at the high or at the low level. The corresponding clock period with an ideal 50% duty cycle is twice this time-out value.

Table A-16. Crystal Monitor Characteristics

Num	C	Rating	Symbol	Min	Typ	Max	Unit
J1	C	Timeout period	t_{CMTO}	1.25	2.5	5.0	µs

A.9.4 Clock Quality Checker

The timing for the clock quality check is derived from the oscillator and the VCO frequency range in Table A-15. These numbers define the upper time limit for the individual check windows to complete.

Table A-17. Clock Quality Checker Characteristics

Num	C	Rating	Symbol	Min	Typ	Max	Unit
K1	D	Check Window	t_{CQCCW}	50000	—	50000	t_{SCM} [1]
				9.1	—	25	ms
K2	D	Timeout Window	t_{CQCTO}	50	—	50	t_{CQCCW} [2]
				0.46	—	1.25	s

1. See parameter I3
2. See parameter K1

A.9.5 Startup

Table A-18 summarizes several startup characteristics explained in this section. Refer to the *MAC7200 Microcontroller Family Reference Manual* (MAC7200RM/D) for a detailed description of the startup behavior.

Table A-18. System Reset Characteristics

Num	C	Rating	Symbol	Min	Typ	Max	Unit
L1	D	Reset input pulse width, minimum input time	PW_{RSTL}	2	—	—	t_{osc} [1]
L2	D	Startup from Reset [2]	t_{RST}	t_{UPOSC} + 195*t_{OSC} + t_{RCU}	—	t_{UPOSC} + 198*t_{OSC} + t_{RCU}	

1. $t_{OSC} = 1/f_{OSC}$. See parameters H1a and H1b.
2. Normal operating conditions, does not apply in SCM mode.

A.10 General Purpose I/O (PIM) Timing

A.10.1 General Purpose Input (GPI)

Table A-19 lists General Purpose Input timings, which are shown in Figure A-2.

These timings are required if the GPIs are used as synchronous inputs in this case the input setup/hold must be met with respect to the rising edge of the reference clock. The reference clock is the CLKOUT output.

For use as asynchronous inputs the setup and hold constraints don't need to be maintained, but signals must be held for a minimum of two system clock cycles.

Table A-19. General Purpose Input Timing Specifications

Num	C	Rating[1]	Symbol	Min	Typ	Max	Unit
M1	D	CLKOUT	t_{CYC}	28	—	2000	ns
Data Inputs							
M2	D	General Purpose Input valid to CLKOUT high (Setup time)	t_{PVCH}	12	—	—	ns
M3	D	CLKOUT high to General Purpose Input invalid (Hold time)	t_{CHPI}	2	—	—	ns

1. Timing specifications have been indicated taking into account the full slew rate for the pads.

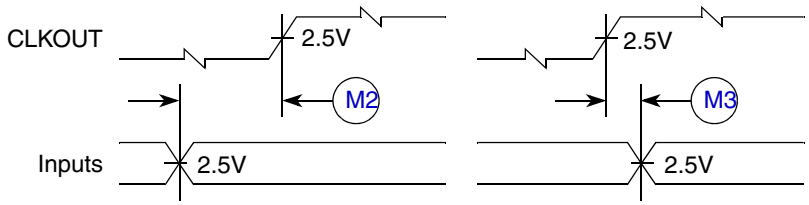


Figure A-2. General Purpose Input Timing Specifications

A.10.2 General Purpose Output (GPO)

Table A-20 lists General Purpose Output timings, which are shown in Figure A-3.

NOTE

All GPO timings are synchronous; the output delay is specified with respect to the CLKOUT output.

All other timing relationships can be derived from these values.

Table A-20. General Purpose Output Timing Specifications

Num	C	Rating[1]	Symbol	Min	Typ	Max	Unit
N1	D	CLKOUT	t_{CYC}	28	—	2000	ns
Data Inputs							

Table A-20. General Purpose Output Timing Specifications (Continued)

Num	C	Rating[1]	Symbol	Min	Typ	Max	Unit
N2	D	CLKOUT high to General Purpose Output valid	t_{CHPOV}	—	—	11	ns
N3	D	CLKOUT high to General Purpose Output invalid	t_{CHPOI}	–2.0	—	—	ns

1. Timing specifications apply to the “full slew rate” configuration for the pads.

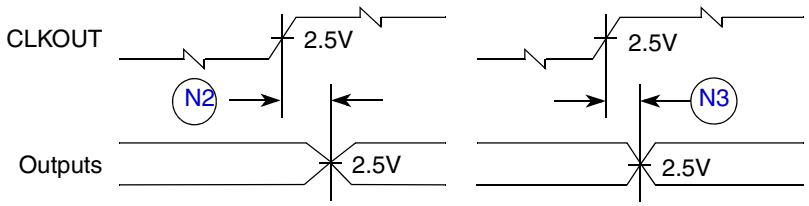


Figure A-3. General Purpose Output Timing Specifications

A.11 Nexus Timing Specifications

A.11.1 Nexus Inputs

Note that both the sole Nexus input, EVTI, is synchronized internally, so no setup or hold timings is required. Consequently, the EVTI signal must be held for a minimum of two system clock cycles.

Table A-21. Nexus Input Timing Specifications

Num	C	Rating	Symbol	Min	Typ	Max	Unit
O1	D	EVTI pulse width	t_{CYC}	2	—	—	t_{SYS}[1]

1. $t_{SYS} = 1/f_{SYS}$. See parameter C7.

A.11.2 Nexus Outputs

Table A-22 lists Nexus Output timings, which are shown in Figure A-4.

NOTE

All Nexus output timings are synchronous; that is, input setup/hold and output delay with respect to the falling edge of a reference clock. The reference clock is the MCKO output. The RDY signal can be asserted while MCKO is inactive.

All other timing relationships can be derived from these values.

Table A-22. Nexus Output Timing Specifications

Num	C	Rating[1]	Symbol	Min	Typ	Max	Unit
P1	D	MCKO	t_{CYC}	25	—	—	ns
Data Outputs							
P2	D	MCKO low to Nexus Output valid	t_{CLPOV}	—	—	3.5	ns
P3	D	MCKO low to Nexus Output invalid	t_{CLPOI}	–5.5	—	—	ns

1. Timing specifications have been indicated taking into account the full slew rate for the pads and 30 pF output load.

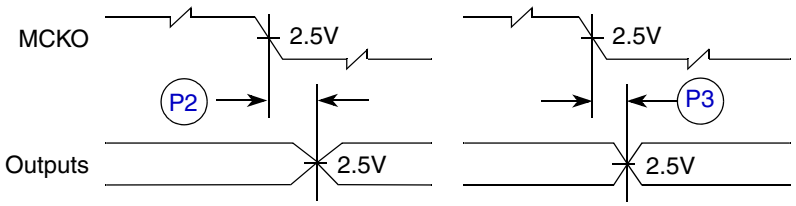


Figure A-4. Nexus Output Timing Specifications

A.12 External Interrupt Inputs (IRQ/XIRQ)

Note that both the IRQ and XIRQ inputs are synchronized internally, so no setup or hold timings are required. Consequently, the input signal must be held long enough to be synchronized.

Table A-23. External Interrupt Characteristics

Num	C	Rating	Symbol	Min	Typ	Max	Unit
Q1	D	Interrupt pulse width, $\overline{IRQ}$ edge-sensitive mode	PW_{IRQ}	4	—	—	t_{SYS}[1]

1. $t_{SYS} = 1/f_{SYS}$. See parameter C7.

A.13 JTAG Port Timing

Table A-24 lists JTAG Port timings, which are shown in Figure A-5.

NOTE

All JTAG input and output timings are synchronous; that is, input setup/hold and output delay with respect to the rising edge of a reference clock. The reference clock is the TCK input.

All other timing relationships can be derived from these values.

Table A-24. JTAG Port Timing

Num	C	Rating[1]	Symbol	Min	Typ	Max	Unit
R1	D	TCK Frequency of Operation	f_{JCYC}	DC	—	f_{SYS} / 8	MHz[2]
R2	D	TCK Cycle Period	t_{JCYC}	8	—	—	t_{SYS}[3]

Table A-24. JTAG Port Timing (Continued)

Num	C	Rating[1]	Symbol	Min	Typ	Max	Unit
R3	D	TCK Clock Pulse Width	t_{JCW}	2	—	—	t_{SYS}[3]
R4	D	TMS, TDI Input Data Setup Time to TCK Rise	t_{TAPBST}	1	—	—	t_{SYS}[3]
R5	D	TMS, TDI Input Data Hold Time after TCK Rise	t_{TAPBHT}	1	—	—	t_{SYS}[3]
R6	D	TCK Fall to TDO Data Valid	t_{TDODV}	—	—	1	t_{SYS}[3]
R7	D	TCK Fall to TDO Data Invalid	t_{TDODI}	1	—	—	t_{SYS}[3]

1. Timing specifications have been indicated taking into account the full slew rate for the pads.
2. See parameter C7.
3. $t_{SYS} = 1/f_{SYS}$. See parameter C7.

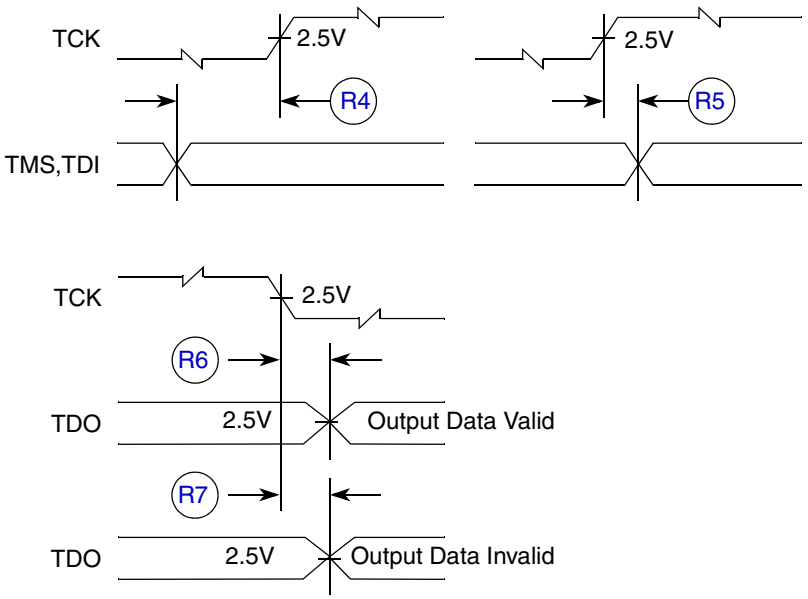


Figure A-5. JTAG Port Timing Specifications

A.14 DSPI Timing

A.14.1 Master Mode

Master mode timing values are shown in Table A-25 and illustrated in Figure A-6 and Figure A-7.

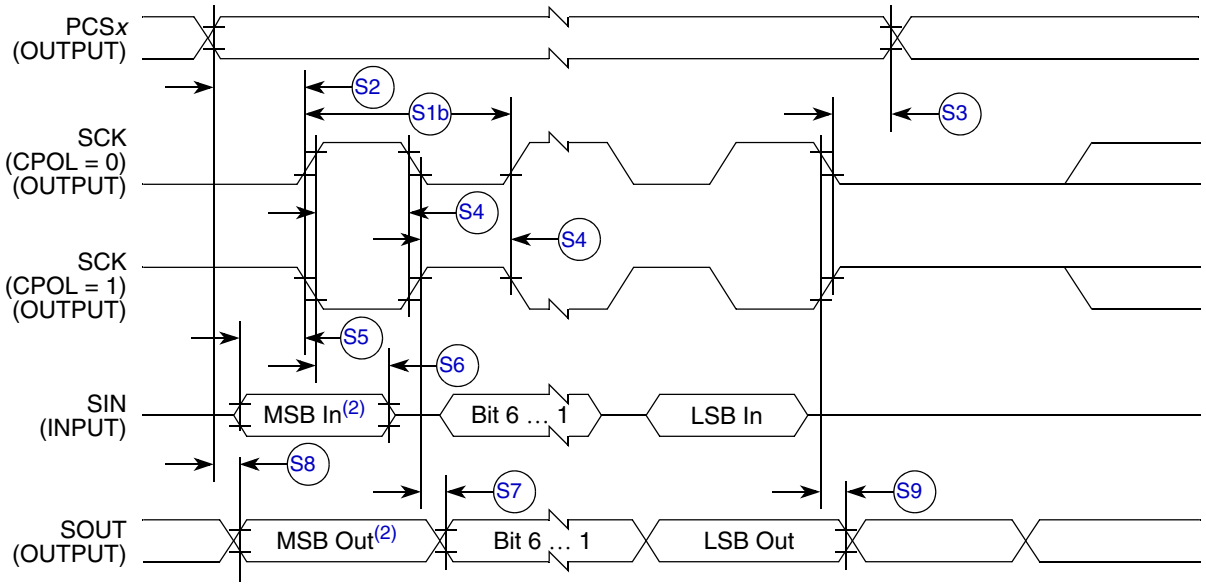
Table A-25. SPI Master Mode Timing Characteristics

Conditions are shown in Table A-5 unless otherwise noted, C_{LOAD} = 30 pF on all outputs							
Num	**C**	**Rating**	**Symbol**	**Min**	**Typ**	**Max**	**Unit**
S1a	D	Operating Frequency (baud rate)	f_{OP}[1]	—	—	18	MHz
S1b	D	SCK Period	t_{SCK}	$t_{SCK} = 1 / f_{OP}$			

Table A-25. SPI Master Mode Timing Characteristics (Continued)

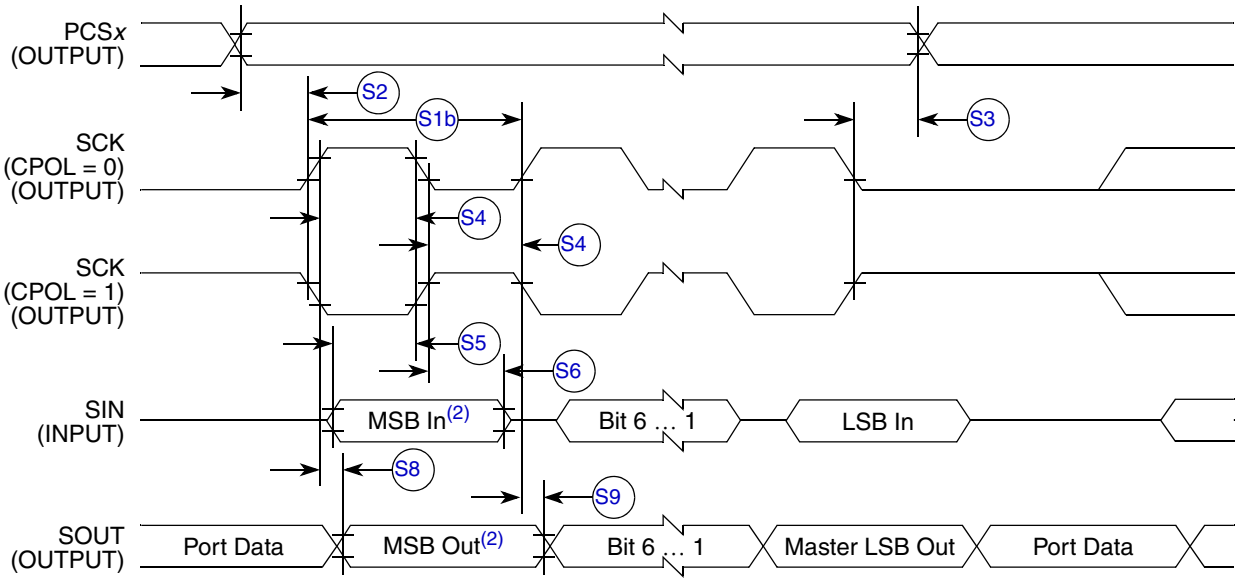
Conditions are shown in Table A-5 unless otherwise noted, C_{LOAD} = 30 pF on all outputs							
S2	D	Enable Lead Time	t_{LEAD}	0.5	—	—	t_{SCK}[2]
S3	D	Enable Lag Time	t_{LAG}	0.5	—	—	t_{SCK}[2]
S4	D	Clock (SCK) Duty Cycle	t_{DUTY}	40	—	60	%
S5	D	Data Setup Time (Inputs)	t_{su}	18	—	—	ns
S6	D	Data Hold Time (Inputs)	t_{hi}	-6	—	—	ns
S7	D	Data Valid (after Enable Edge)	t_{v}	—	—	6	ns
S8	D	Data Valid (after ChipSelect Edge)	t_{v}	—	—	6	ns
S9	D	Data Hold Time (Outputs)	t_{ho}	-3	—	—	ns

1. Refer to Chapter 30, “Deserial Serial Peripheral Interface (DSPI)” for all available baud rates.
2. See parameter S1b.



1 If configured as output.
2 LSBFE = 0. For LSBFE = 1, bit order is LSB, bit 1, … bit 6, MSB.

Figure A-6. SPI Master Timing (CPHA = 0)



1 If configured as output.

2 LSBFE = 0. For LSBFE = 1, bit order is LSB, bit 1, … bit 6, MSB.

Figure A-7. SPI Master Timing (CPHA = 1)

A.14.2 Slave Mode

Slave mode timing values are shown in Table A-26 and illustrated in Figure A-8 and Figure A-9.

Table A-26. SPI Slave Mode Timing Characteristics

Conditions are shown in Table A-5 unless otherwise noted, C_{LOAD} = 30 pF on all outputs							
Num	**C**	**Rating**	**Symbol**	**Min**	**Typ**	**Max**	**Unit**
T1a	D	Operating Frequency	f_{OP}[1]	see S1a			
T1b	D	SCK Period (t_{SCK} = 1 / f_{OP})	t_{SCK}	see S1b			
T2	D	Enable Lead Time	t_{lead}	1	—	—	t_{SYS}[3]
T3	D	Enable Lag Time	t_{lag}	1	—	—	t_{SYS}[3]
T4	D	Clock (SCK) Duty Cycle	t_{DUTY}	40	—	60	%
T5a	D	Data Setup Time (Inputs) (CPHA=0)	t_{su}	12	—	—	ns
T5b	D	Data Setup Time (Inputs) (CPHA=1)	t_{su}	18	—	—	ns
T6a	D	Data Hold Time (Inputs) (CPHA=0)[2]	t_{hi}	10	—	—	ns
T6b	D	Data Hold Time (Inputs) (CPHA=1)	t_{hi}	-6	—	—	ns
T7	D	Slave Access Time	t_a	—	—	1	t_{SYS}[3]
T8	D	Slave SIN Disable Time	t_{dis}	—	—	1	t_{SYS}[3]
T9a	D	Data Valid (after SCK Edge) (CPHA=0)	t_v	—	—	20	ns
T9b	D	Data Valid (after SCK Edge) (CPHA=1)	t_v	—	—	17	ns

Table A-26. SPI Slave Mode Timing Characteristics (Continued)

Conditions are shown in Table A-5 unless otherwise noted, C_{LOAD} = 30 pF on all outputs							
T10a	D	Data Hold Time (Outputs) (CPHA=0)	t_{ho}	5	—	—	ns
T10b	D	Data Hold Time (Outputs) (CPHA=1)	t_{ho}	-3	—	—	ns

1. Refer to *MAC7200 Microcontroller Family Reference Manual* (MAC7200RM) Chapter 22 for all available baud rates.
2. This number is calculated assuming the SMPL_PT bitfield in DSPI_MCR is set to 0b10.

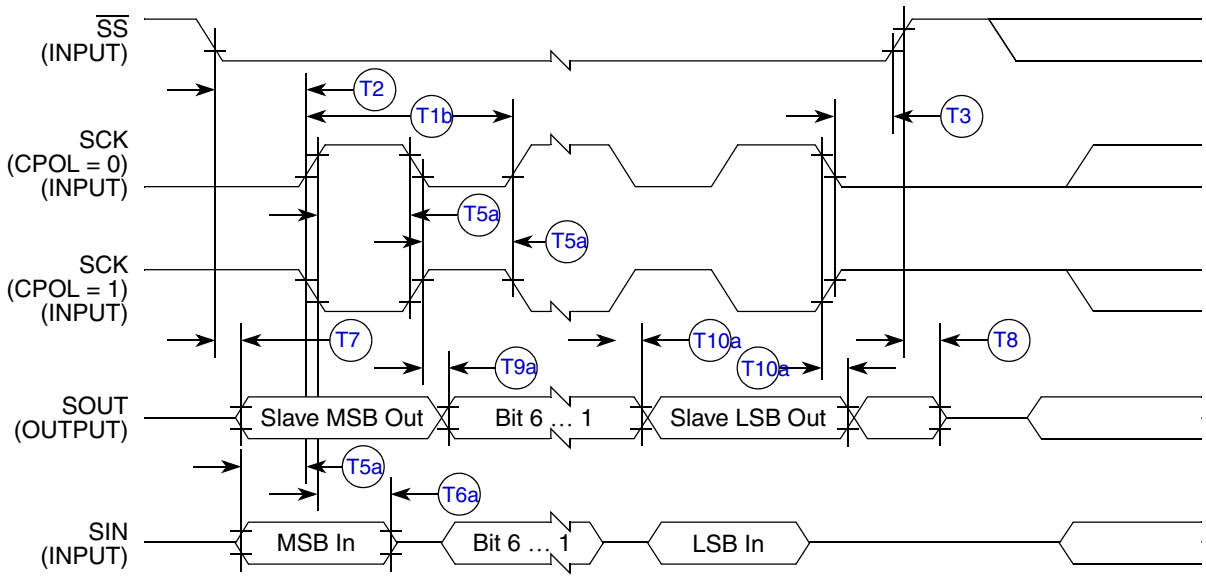


Figure A-8. SPI Slave Timing (CPHA = 0)

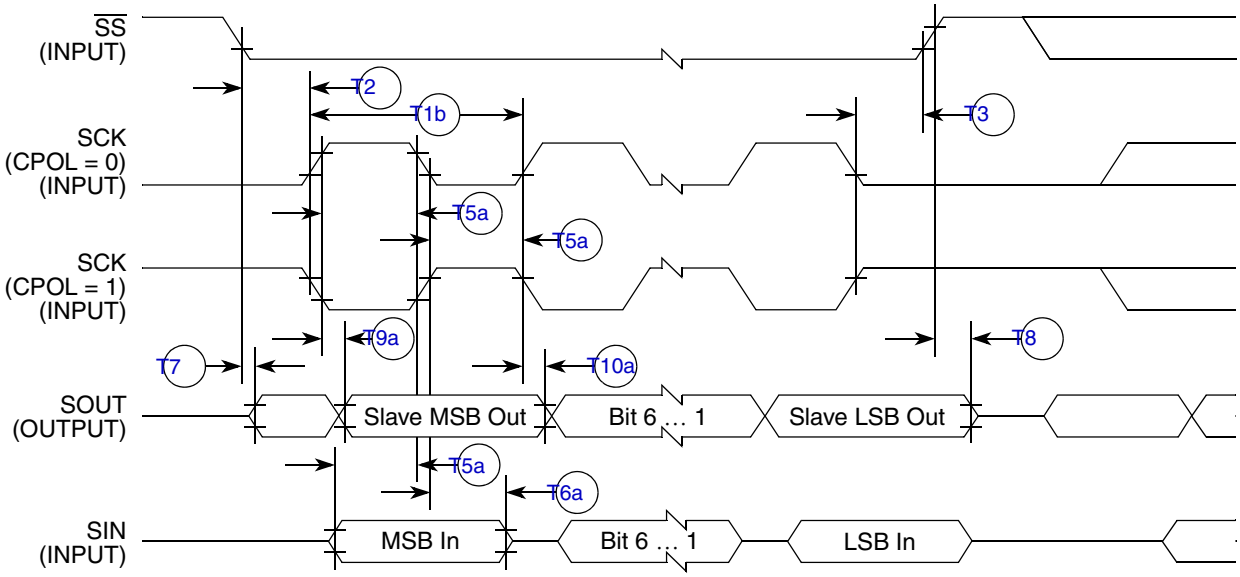


Figure A-9. SPI Slave Timing (CPHA = 1)

A.15 External Bus (FlexBus) Timing Specifications

A multi-function external bus interface called FlexBus is provided on the MAC72xx with basic functionality to interface to slave-only devices up to a maximum bus frequency of 35 MHz. It can be directly connected to asynchronous or synchronous devices such as external boot ROMs, flash memories, gate-array logic, or other simple target (slave) devices with little or no additional circuitry. For asynchronous devices, a simple chip-select based interface can be used. The FlexBus interface has three general purpose chip-selects ($\overline{CS}$[2:0]). Chip-select $\overline{CS0}$ can be dedicated to boot ROM access and can be programmed to be byte (8 bits) or half-word (16 bits) wide. Control signal timing is compatible with common ROM / flash memories.

A.15.1 FlexBus Inputs

Table A-27 lists external bus input timings, which are shown in Figure A-10, Figure A-11 and Figure A-12.

NOTE

All external bus timings are synchronous; that is, input setup/hold and output delay with respect to the rising edge of a reference clock. The reference clock is the CLKOUT output. It has the same duty cycle as the Peripheral Bus Clock.

All other timing relationships can be derived from these values.

Table A-27. External Bus Input Timing Specifications

Num	C	Rating[1]	Symbol	Min	Typ	Max	Unit
U1	D	CLKOUT	t_{CYC}	28	—	4000	ns
Control Inputs							
U2a	D	Control input valid to CLKOUT high[2]	t_{CVCH}	13	—	—	ns
U3a	D	CLKOUT high to control inputs invalid[2]	t_{CHCII}	0	—	—	ns
Data Inputs							
U4	D	Data input (DATA[15:0]) valid to CLKOUT high	t_{DIVCH}	10	—	—	ns
U5	D	CLKOUT high to data input (DATA[15:0]) invalid	t_{CHDII}	0	—	—	ns

1. Timing specifications have been indicated taking into account the full slew rate for the pads.
2. The $\overline{TA}$ pin is the only control input on MAC7200 family devices.

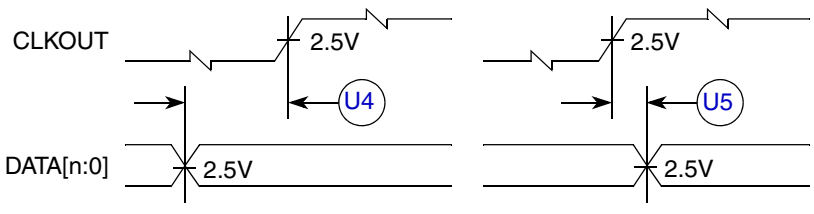


Figure A-10. External Bus Input Timing Specifications

A.15.2 FlexBus Outputs

Table A-28 lists external bus output timings. Read/write bus timings listed in Table A-28 are shown in Figure A-11 and Figure A-12.

Table A-28. External Bus Output Timing Specifications

Num	C	Rating	Symbol	Min	Max	Unit
Control Outputs						
U6a	D	CLKOUT high to chip selects valid [1]	t_{CHCV}	—	8	ns
U6b	D	CLKOUT high to byte select ($\overline{BS}$[1:0]) valid[2]	t_{CHBV}	—	8	ns
U6c	D	CLKOUT high to output select ($\overline{OE}$) valid[3]	t_{CHOV}	—	8	ns
U6d	D	CLKOUT high to output select ($\overline{TBST}$) valid[4]	t_{CHOV}	—	8	ns
U7a	D	CLKOUT high to control output ($\overline{BS}$[1:0], $\overline{OE}$) invalid	t_{CHCOI}	0	—	ns
U7b	D	CLKOUT high to chip selects invalid	t_{CHCI}	0	—	ns
Address and Attribute Outputs						
U8	D	CLKOUT high to address (ADDR[21:0]) and control (R/$\overline{W}$) valid	t_{CHAV}	—	8	ns
U9	D	CLKOUT high to address (ADDR[21:0]) and control (R/$\overline{W}$) invalid	t_{CHAI}	0	—	ns
Data Outputs						
U10	D	CLKOUT high to data output (DATA[15:0]) valid	t_{CHDOV}	—	8	ns
U11	D	CLKOUT high to data output (DATA[15:0]) invalid	t_{CHDOI}	0	—	ns
U12	D	CLKOUT high to data output (DATA[15:0]) high impedance	t_{CHDOZ}	—	9	ns

1. $\overline{CS}n$ transitions after the falling edge of CLKOUT.
2. $\overline{BS}n$ transitions after the falling edge of CLKOUT.
3. $\overline{OE}$ transitions after the falling edge of CLKOUT.
4. $\overline{TBST}$ transitions after the falling edge of CLKOUT.

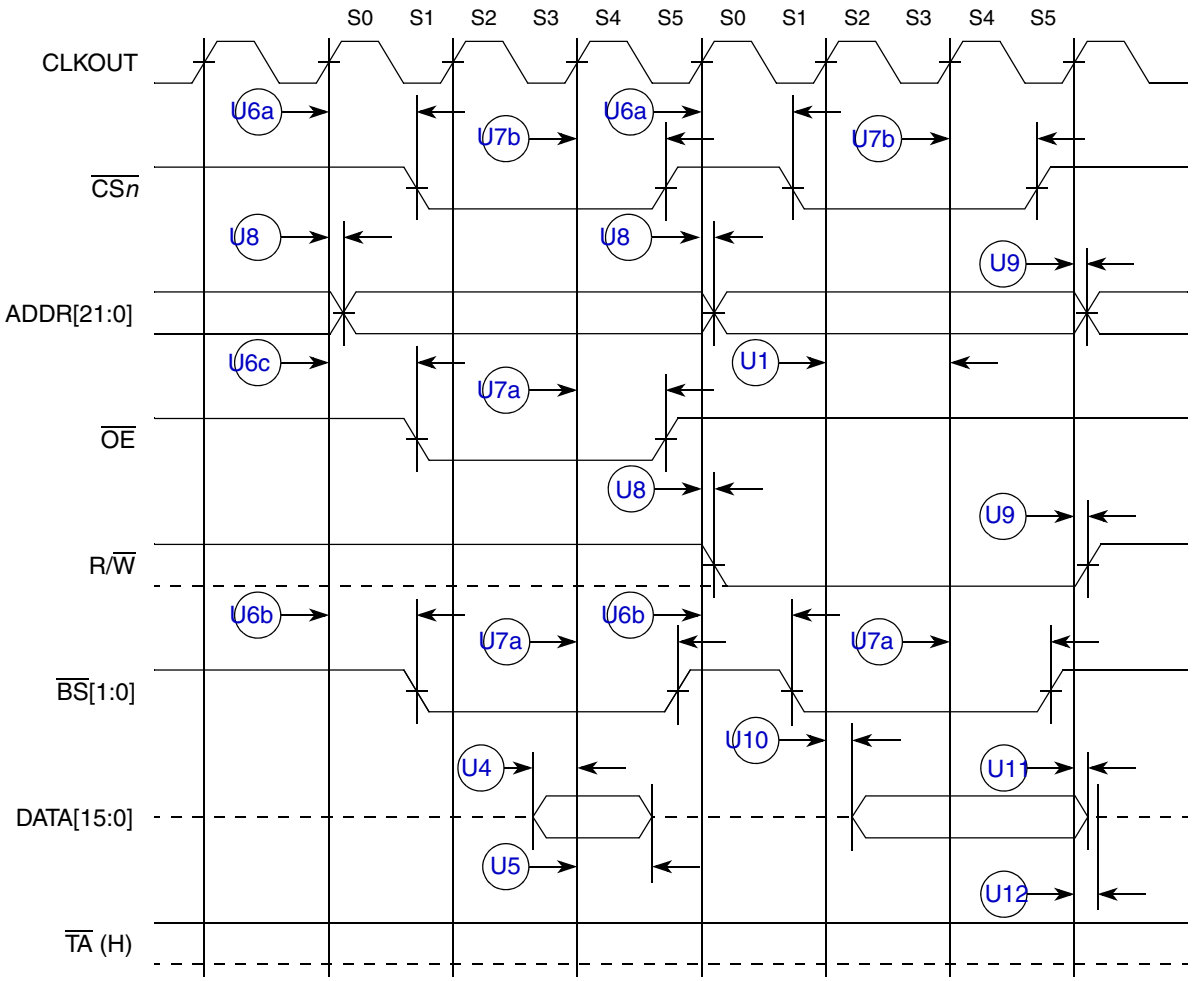


Figure A-11. Read/Write (Internally Terminated) Bus Timing

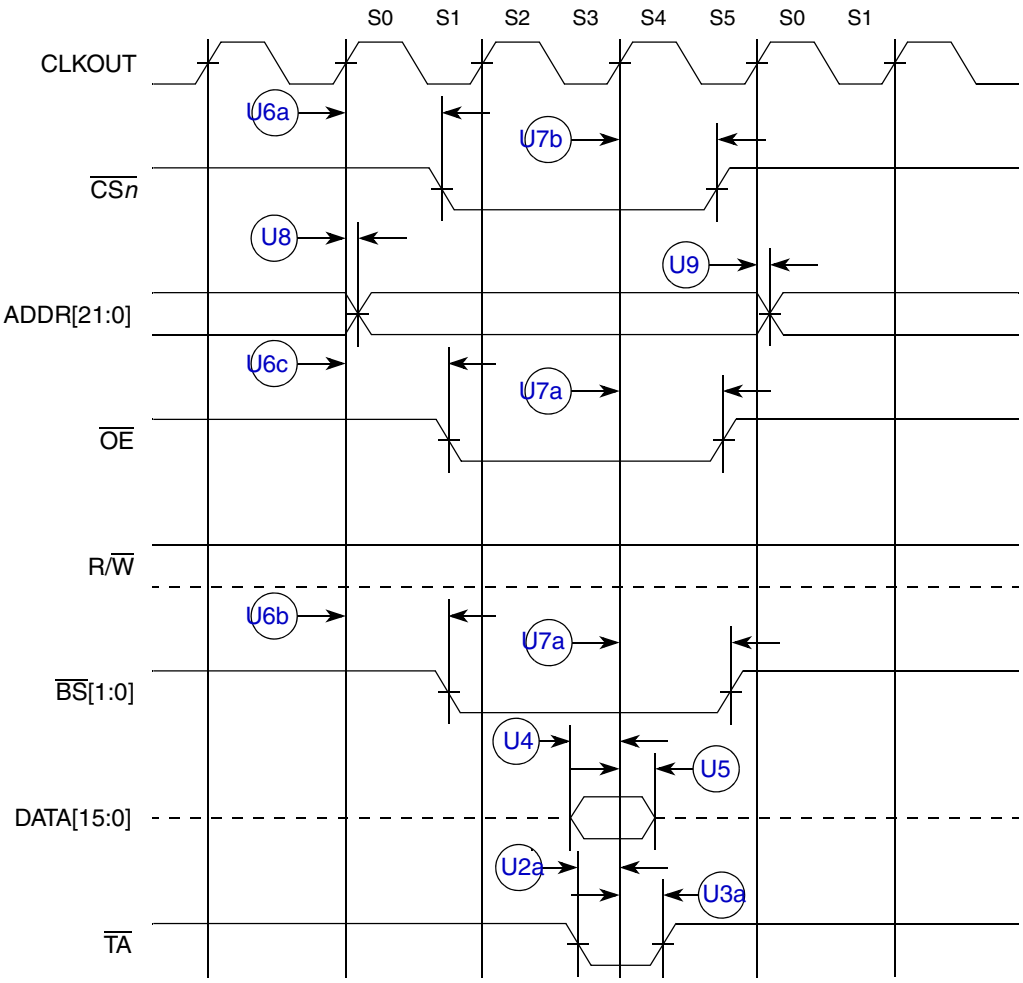


Figure A-12. Read Bus Cycle Terminated by TA

A.16 Analog-to-Digital Converter Characteristics

Table A-29 shows conditions under which the ATD operates. The following constraints exist to obtain full-scale, full range results: $V_{SS}A \le V_{RL} \le V_{IN} \le V_{RH} \le V_{DD}A$. This constraint exists because the sample buffer amplifier cannot drive beyond the ATD power supply levels. If the input level V_{IN} goes outside of this range it will effectively be clipped.

Table A-29. ATD Electrical Characteristics (Operating)[1]

Num	C	Rating	Symbol	Min	Typ	Max	Unit
V1	D	Analog Reference Low Voltage	V_{RL}	$V_{SS}A$ - 0.1	—	$V_{SS}A$ + 0.1	V
V2	D	V_{RH} Differential Voltage	V_{RL} - $V_{SS}A$	-100	—	100	mV
V3	D	Analog Reference High Voltage	V_{RH}	$V_{DD}A$ - 0.1	—	$V_{DD}A$ + 0.1	V
V4	D	V_{REF} Differential Voltage	$V_{RH} - V_{RL}$	4.5	—	5.25	V
V5	D	Analog Input Voltage	V_{IN}	$V_{SS}A$ - 0.3	—	$V_{DD}A$ + 0.3	V

Table A-29. ATD Electrical Characteristics (Operating)[1] (Continued)

Num	C	Rating	Symbol	Min	Typ	Max	Unit
V6	C	Analog Reference Supply Current (V_{RH}, V_{RL})	I_{REF}	—	—	1	mA
V7	C	Disruptive Input Injection Current[2 3 4]	I_{NA}	–1.0	—	1.0	mA

1. All voltages referred to $V_{SS}A$, –40 to 125°C, $V_{DD}A$ = 5.0 V ±10% and 2.0 MHz conversion rate unless otherwise noted. Refer to Table A-5 for additional operating conditions.
2. Below disruptive current conditions, the channel being stressed has conversion values of 0x3FF for analog inputs greater than V_{RH} and 0x000 for values less than V_{RL}. This assumes that V_{RH} <= VDDA and V_{RL} >= VSSA due to the presence of the sample amplifier. Other channels are not affected by non-disruptive conditions.
3. Exceeding limit may cause conversion error on stressed channels and on unstressed channels. Transitions within the limit do not affect device reliability or cause permanent damage.
4. Condition applies to two adjacent pins.

A.16.1 Factors Influencing Accuracy

The main factors influencing the ATD accuracy are the supply voltages (V_{RL}, V_{RH}, $V_{SS}A$ and $V_{DD}A$) and the ATD conversion clock frequency f_{ATDCLK}. Please see Table A-31 for details.

Table A-30. ATD Electrical Characteristics

Conditions are shown in Table A-5 unless otherwise noted							
Num	**C**	**Rating**	**Symbol**	**Min**	**Typ**	**Max**	**Unit**
W1	T	Total Input Capacitance (Non Sampling)	C_{INN}	—	—	10	pF
W2		Stop Mode Recovery Time[1]	T_{REC}	—	—	10	µs

1. Stop mode recovery time is the time from the setting of either of the enable bits in the ATD Control Register to the time that the ATD is ready to perform conversions.

A.16.2 ATD Accuracy

Table A-31 specifies the ATD conversion performance excluding any errors due to current injection, input capacitance and source resistance.

Table A-31. ATD Conversion Performance in 5 V Range

Conditions shown in Table A-5 unless otherwise noted. $V_{REF} = V_{RH} - V_{RL}$ = 5.12 V, resulting in one 9 bit count = 10 mV and one 12 bit count = 1.25 mV f_{ATDCLK} = 12 MHz, $V_{DD}A$ -0.1V ≤ V_{RH} ≤ $V_{DD}A$ + 0.1V, with calibration, a source impedance of < 150 ohm, no precondition, WARP on.							
Num	**C**	**Rating**	**Symbol**	**Min**	**Typ**	**Max**	**Unit**
X1	P	Resolution[1]	LSB	1.25	—	—	mV
X2	P	Differential Nonlinearity @12MHz ATD clock and 12-bit resolution	DNL	-8	—	+8	Counts

Table A-31. ATD Conversion Performance in 5 V Range (Continued)

Conditions shown in Table A-5 unless otherwise noted. $V_{REF} = V_{RH} - V_{RL} = 5.12$ V, resulting in one 9 bit count = 10 mV and one 12 bit count = 1.25 mV $f_{ATDCLK} = 12$ MHz, $V_{DD}A$ -0.1V ≤ V_{RH} ≤ $V_{DD}A$ + 0.1V, with calibration, a source impedance of < 150 ohm, no precondition, WARP on.							
X3	P	Integral Nonlinearity @12MHz ATD clock and 12-bit resolution	INL	-8	—	+8	Counts
X4	P	Total unadjusted Error[2] @12MHz ATD clock and 12-bit resolution[3]: V_{RL} + 100mV <= V_{in} <= V_{RH} - 100mV	TUE	-8	—	+8	Counts
		0 <= V_{in} < V_{RL} + 100mV and V_{RH} - 100mV <= V_{in} < V_{RH}		-16	—	+16	
X5	C	Source Impedance[4]	R_{Source}	0	—	100	kohm

1. At V_{RH} - V_{RL} = 5.12V, one LSB = 1.25mV = one count
2. These values include the quantization error which is inherently 1/2 count for any A/D converter.
3. The TUE specification will always be better than the sum of the INL and DNL errors due to cancelling errors.
4. If using a source impedance greater than 150 ohm, then the customer should characterize the ATD sample clock value.

For the following definitions see also Figure A-13.

DNL — Differential Non Linearity Error

The measured device output codes are normalized, and the difference of each code width from 1 LSB is measured. The largest difference is reported as the DNL.

$$\text{MAX}[\text{ABS}(\text{ViStart} - \text{ViEnd}) - \text{Bitweight}]_{i=x}^{y} \qquad \textit{Eqn. A-14}$$

where

- x = location in the normalized results array where a voltage ramp begins
- y = location in the normalized results array where a voltage ramp ends
- ViStart = the input voltage when the observed normalized output changes
- ViEnd = the input voltage when the normalized output changed previously
- Bitweight = volts per LSB step

INL — Integral (Cumulative) Non Linearity Error

The measured output codes are normalized and compared against the codes that would be produced by an ideal ATD. The difference is taken between each measured transition point and the corresponding transition point for the ideal ATD. The largest difference is reported as INL.

$$\text{MAX}[(\text{ViActual} - \text{ViIdeal})]_{i=x}^{y} \qquad \textit{Eqn. A-15}$$

where

- x = location in the normalized results array where a voltage ramp begins
- y = location in the normalized results array where a voltage ramp ends
- ViActual = the input voltage when the observed normalized output changes
- ViIdeal = the input voltage that would ideally cause the observed output

TUE — Total Unadjusted Error

The difference between each observed transition point and the corresponding ideal transition point is measured using non-normalized data. The largest difference is reported as the total unadjusted error.

$$\max_{i=x}^{y}[(ViActual - ViIdeal)] \qquad \textit{Eqn. A-16}$$

where

- x = location in the actual results array where a voltage ramp begins
- y = location in the actual results array where a voltage ramp ends
- ViActual = the input voltage when the observed actual output changes
- ViIdeal = the input voltage that would ideally cause the observed output

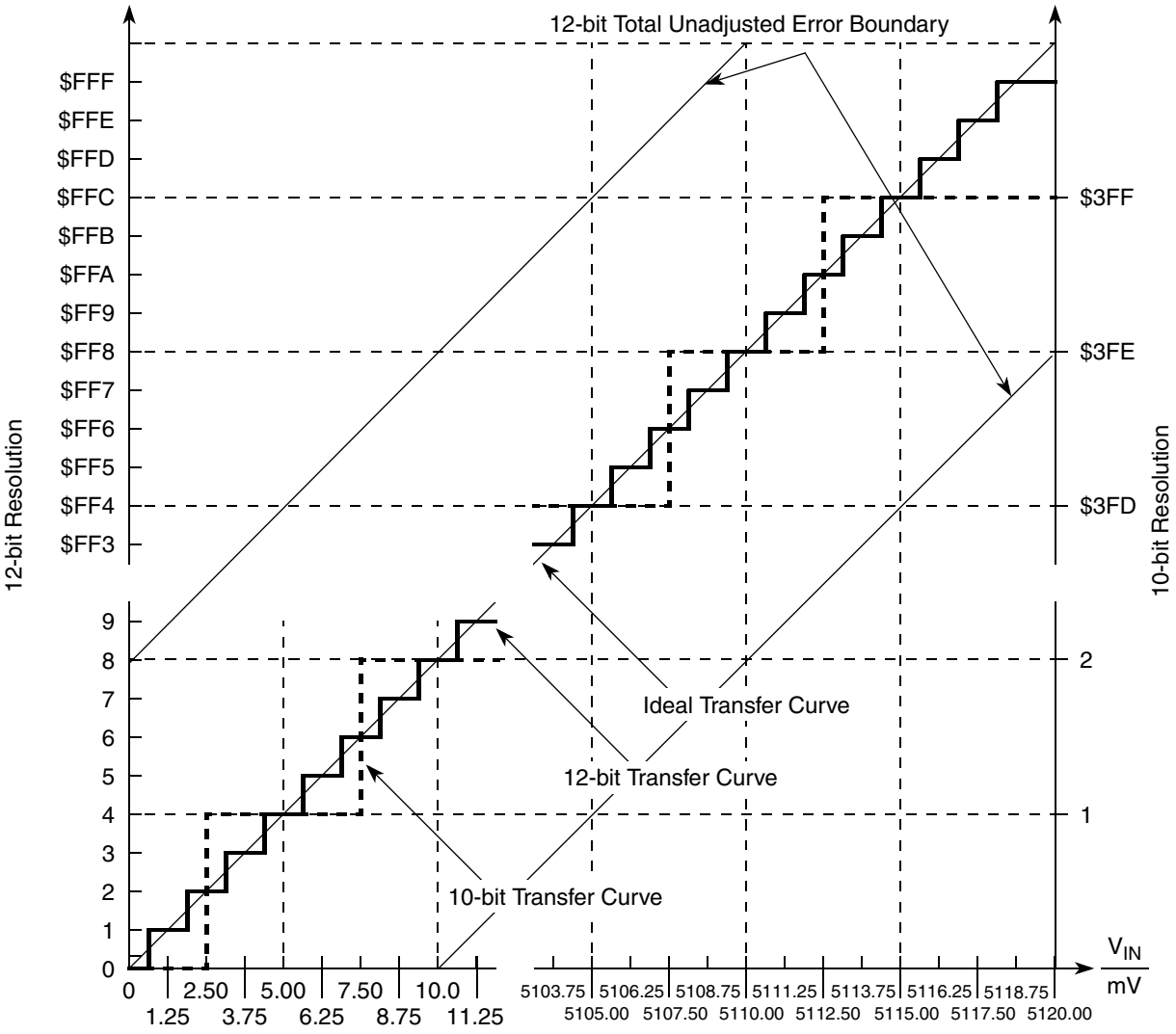


Figure A-13. ATD Accuracy Definitions

NOTE

Figure A-13 shows only definitions. For specification values refer to Table A-31.

A.16.3 ATD Timing Specifications

Table A-32. ATD Timing Specifications

Num	C	Parameter	Symbol	Min	Typ	Max	Unit
Y1	D	ATD Module Clock Frequency	f_{MODCLK}	f_{SYS}[1] (min)	—	f_{SYS}[1] (max)	MHz
Y2	D	ATD Conversion Clock Frequency	f_{ATDCLK}	0.5	—	12	MHz
Y3	D	ATD 12-bit Conversion Time for Single-ended Conversions Without Pre-discharge[2]	t_{CONV12}	17.5 x t_{ATDCLK} + 2 x t_{SYS} or[3] 17 x t_{ATDCLK} + 7 x t_{SYS}	—	143.5 x t_{ATDCLK} + 2 x t_{SYS} or[4] 143 x t_{ATDCLK} + 7 x t_{SYS}	µs

1. $t_{SYS} = 1/f_{SYS}$. See parameter C7.
2. The conversion time consists of the sampling time (minimum 2, maximum 128), the time to convert the sampled voltage, the result adjustment and the time needed for internal processing
3. For the min value, use whichever total time is shorter
4. For the max value, use whichever total time is longer

Table A-33. ATD External Trigger Timing Specifications

Num	C	Rating	Symbol	Min	Typ	Max	Unit
Z1	D	ETRIG Minimum Period	T_{PERIOD}	—	—	4	t_{SYS}[1]
Z2	D	ETRIG Minimum Pulse Width	t_{PW}	2	—	—	t_{SYS}[1]
Z3	D	ETRIG Level Recovery[2]	t_{LR}	2	—	—	t_{SYS}[1]
Z4	D	Conversion Start Delay	t_{DLY}	—	—	4	t_{SYS}[1]

1. $t_{SYS} = 1/f_{SYS}$. See parameter C7.
2. Time prior to end of conversion that the ETRIG pin must be deactivated so that another conversion sequence does not start.

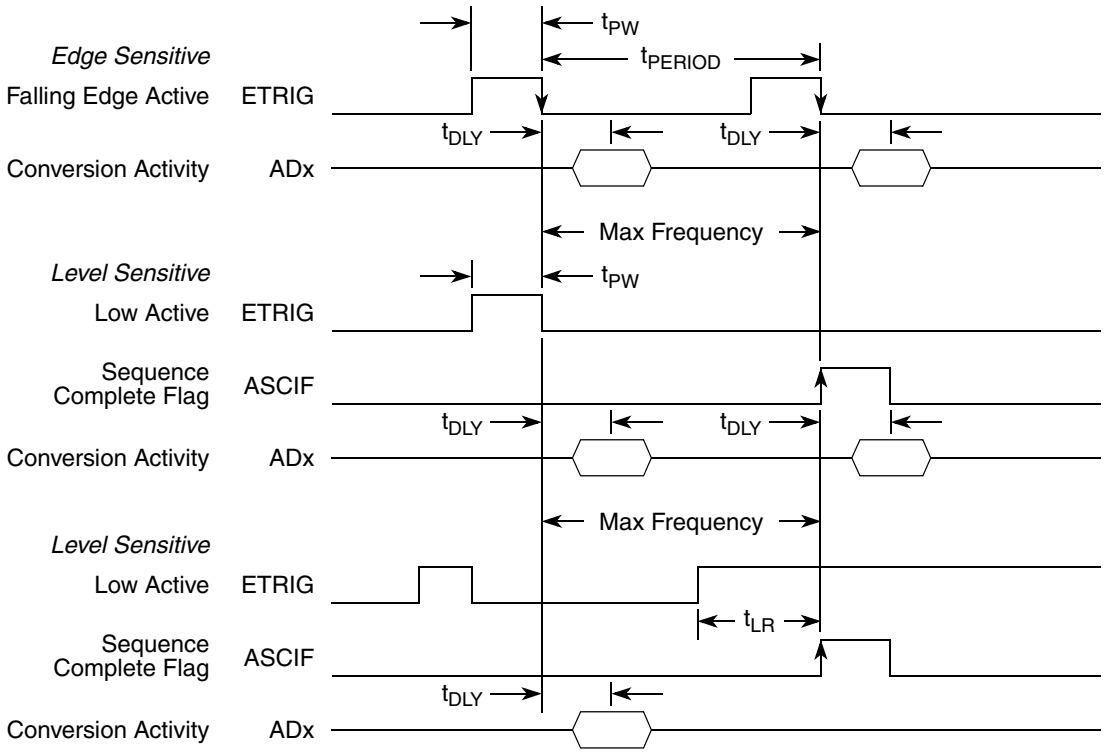


Figure A-14. ATD External Trigger Timing Diagram

A.17 I2C Timing Specifications

Table A-34 lists specifications for the I^2C input timing parameters shown in Figure A-15.

Table A-34. I^2C Input Timing Specifications between SCL and SDA

Num	C	Rating	Min	Typ	Max	Units
AA1	D	Start condition hold time	2	—	—	t_{PERIPH}[1]
AA2	D	Clock low period	8	—	—	t_{PERIPH}[1]
AA3	D	SCL/SDA rise time (V_{IL} = 0.5 V to V_{IH} = 2.4 V)	—	—	1	ms
AA4	D	Data hold time	0	—	—	ns
AA5	D	SCL/SDA fall time (V_{IH} = 2.4 V to V_{IL} = 0.5 V)	—	—	1	ms
AA6	D	Clock high time	4	—	—	t_{PERIPH}[1]
AA7	D	Data setup time	0	—	—	ns
AA8	D	Start condition setup time (for repeated start condition only)	2	—	—	t_{PERIPH}[1]
AA9	D	Stop condition setup time	2	—	—	t_{PERIPH}[1]

1. $t_{PERIPH} = 1/f_{PERIPH}$. See parameter C8.

Table A-35 lists specifications for the I^2C output timing parameters shown in Figure A-15.

Table A-35. I²C Output Timing Specifications between SCL and SDA

Num	C	Rating	Min	Typ	Max	Units
AA1[1]	D	Start condition hold time	6	—	—	t_{PERIPH}[2]
AA2 [1]	D	Clock low period	10	—	—	t_{PERIPH}[2]
AA3 [3]	D	SCL/SDA rise time (V_{IL} = 0.5 V to V_{IH} = 2.4 V)	—	—	3	nμs
AA4 [1]	D	Data hold time	7	—	—	t_{PERIPH}[2]
AA5 [4]	D	SCL/SDA fall time (V_{IH} = 2.4 V to V_{IL} = 0.5 V)	—	—	3	ns
AA6 [1]	D	Clock high time	10	—	—	t_{PERIPH}[2]
AA7 [1]	D	Data setup time	2	—	—	t_{PERIPH}[2]
AA8 [1]	D	Start condition setup time (for repeated start condition only)	20	—	—	t_{PERIPH}[2]
AA9 [1]	D	Stop condition setup time	10	—	—	t_{PERIPH}[2]

1. Output numbers depend on the value programmed into the IFDR; an IFDR programmed with the maximum frequency (IFDR = $20) results in minimum output timings as shown in Table A-35. The I²C interface is designed to scale the actual data transition time to move it to the middle of the SCL low period. The actual position is affected by the prescale and division values programmed into the IFDR; however, the numbers given in Table A-35 are minimum values.
2. $t_{PERIPH} = 1/f_{PERIPH}$. See parameter C8.
3. Because SCL and SDA are open-collector-type outputs, which the processor can only actively drive low, the time SCL or SDA take to reach a high level depends on external signal capacitance and pull-up resistor values.
4. Specified at a nominal 50-pF load.

Figure A-15 shows timing for the values in Table A-34 and Table A-35.

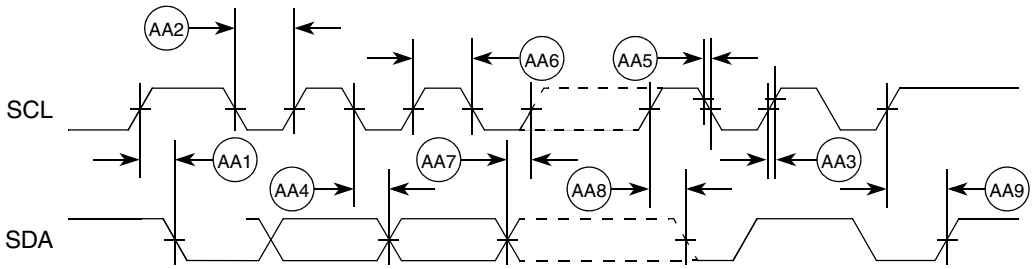


Figure A-15. I²C Input/Output Timings

A.18 Flash Characteristics

NOTE

Unless otherwise noted the abbreviation NVM (Non-Volatile Memory) is used for both the Flash Main Array and the Shadow Block.

A.18.1 NVM Timing Specifications

The Flash Main Array and Shadow Block program and erase operations are timed using an internally generated clock.

Table A-36. NVM Program/Erase Times

Symbol	Characteristic	Min	Typ[1]	Initial Max[2]	Max[3]	Units
$t_{dwprogram}$	Double Word (64 bits) Program Time[4]	—	10	—	500	µs
$t_{pprogram}$	Page (128 bits) Program Time[5]	—	15	44	500	µs
$t_{16kpperase}$	16KB Block Pre-program and Erase Time	—	325	525	5,000	ms
$t_{48kpperase}$	48KB Block Pre-program and Erase Time	—	435	525	5,000	ms
$t_{64kpperase}$	64KB Block Pre-program and Erase Time	—	525	675	5,000	ms
$t_{128kpperase}$	128KB Block Pre-program and Erase Time	—	675	1,800	15,000	ms

1. Typical program and erase times assume nominal supply values and operation at 25C.
2. Initial factory condition: < 100 program/erase cycles, nominal supply values and operation at 25C.
3. The Maximum characteristic is at worst case conditions after the specified number of program erase cycles. This maximum value is characterized but not guaranteed.
4. Actual hardware programming times. This does not include software overhead.
5. Actual hardware programming times. This does not include software overhead.

A.18.2 NVM Reliability

The reliability of the NVM blocks is guaranteed by stress test during qualification, constant process monitors and burn-in to screen early life failures. The failure rates for data retention and program/erase cycling are specified at the operating conditions noted. The program/erase cycle count on the sector is incremented every time a sector or erase event is executed.

Table A-37. NVM Module Life

Symbol	Characteristic	Min	Typical	Max	Units
Array P/E Cycles	P/E Cycles for 16KB, 48KB and 64KB blocks, across full operating temperature range. (-40C <= Tj <= 150C)	100,000	—	—	P/E Cycles
	P/E Cycles for 128KB blocks, across full operating temperature range. (-40C <= Tj <= 150C)	10,000	100,000[1]	—	P/E Cycles
Data Retention	Data Retention for blocks with 0 – 1,000 P/E cycles. (Maximum temperature Tj = 150C)	20	—	—	Years
	Data Retention for blocks with 1,001 – 100,000 P/E cycles. (Maximum temperature Tj = 150C)	5	—	—	Years

1. Typical P/E Cycles is 100,000 cycles for 128KB blocks. For additional information on the Freescale definition of typical endurance, please refer to Engineering Bulletin EB619 “Typical Endurance for Nonvolatile Memory”

NOTES

Only one model may be used to determine the endurance and data retention lifetime for a particular program/erase sector. Thus, the Shadow Block and Main Array may use different models, as may the multiple sectors in the Main Array.

For Flash cycling performance, each Program operation must be preceded by an Erase.

Appendix B
Mechanical Information

B.1 General

This section provides the physical dimensions of the MAC72xx packages.

B.2 100-pin LQFP Package

Please refer to document 98ASS23308W, Issue F (Case 983-02) for the mechanical outline of this package.

This package option is supported for production devices.

B.3 144-pin QFP Package

Please refer to document 98ASS23177W, Issue D (Case 918-03) for the mechanical outline of this package.

NOTE

The 144LQFP packages are not currently qualified; any future qualification will be based upon customer demand and will be subject to specified leadtimes. This package option is supported for limited samples only, and does not include burn-in testing.

THIS PAGE INTENTIONALLY LEFT BLANK

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 26668334
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

LDCForFreescaleSemiconductor@hibbertgroup.com

freescale™
semiconductor

Document Number: MAC7200RM
Rev. 2
04/2007