DSP56F800

User Manual

56F800
16-bit Digital Signal Controllers

DSP56F801-7UM
Rev. 8
03/2007

freescale.com

freescale™
semiconductor

This manual is one of a set of three documents. You need the following manuals to have complete product information: Family Manual, User's Manual, and Technical Data Sheet.

Note: *With the exception of errata documents, if any other Freescale document contains information that conflicts with the information in the device data sheet, the data sheet should be considered to have the most current and correct data.*

Order this document by **DSP56F801-7UM/D - Rev. 8**
March 2007

Summary of Changes and Updates:

Added "List of Figures" and "List of Tables" sections after the Table of Contents.

See the "Document Revision History" section of each chapter for chapter-specific changes.

TABLE OF CONTENTS

Chapter 1
56F800 Family

Chapter 2
Pin Descriptions

Chapter 3
Memory and Operating Modes

Chapter 4
Interrupt Controller (ITCN)

Chapter 5
Flash Memory Interface

Chapter 6
External Memory Interface (EMI)

Chapter 7
General Purpose Input/Output (GPIO)

Chapter 8
Controller Area Network (CAN)

Chapter 9
Analog-to-Digital Converter (ADC)

Chapter 10
Quadrature Decoder

Chapter 11
Pulse Width Modulator Module (PWM)

Chapter 12
Serial Communications Interface (SCI)

Chapter 13
Serial Peripheral Interface (SPI)

Chapter 14
Quad Timer Module (TMR)

Chapter 15
On-Chip Clock Synthesis (OCCS)

Chapter 16
Reset, Low Voltage, Stop and Wait Operations

Chapter 17
OnCE Module

Chapter 18
JTAG Port

Appendix A
Programmer's Sheets

LIST OF FIGURES

LIST OF TABLES

Chapter 1
56F800 Family

1.1 Introduction

Differing in size of memories and choice of peripherals, these multi-functional chips offer exceptional application control using a patented synchronization of the pulse width modulator and analog-to-digital converter. The core provides *more processing power than any other control chip while saving design space and money*. The 56F801, 56F802, 56F803, 56F805, and 56F807 are members of the 56800 core-based family of Digital Signal Controllers (DSCs). Combined on a single chip are the processing power of a controller and the functionality of a microcontroller and a flexible set of peripherals. The chip design creates an extremely cost-effective and compact solution for a number of uses. The family of chips provide control for such applications as:

- AC induction motors (industrial and appliance—washing machines, HVAC, fans, vacuums, and motor drives
- Brush DC motors—car window lifters and electric antennas, toys, cordless tools
- Brushless DC motors (automotive and appliance)—PC fans, ceiling fans, blowers, washing machines, electric power steering systems
- Switched and variable reluctance motors—washing machines, electric power steering, dynamic body control, refrigerator compressors, HVAC, fans, vacuums

and for the power control of:

- General converter/inverter applications
- Uninterruptable power systems—in-line, line interactive, and standby
- Inverter output stages—push-pull, half-bridge, and full bridge

Each of the five chips may be designed, at the minimum, into the following applications:

- Automotive control
- Power line modem
- Uninterruptable power supplies
- Telephony system implementation
- Noise cancellation applications
- Home security
- Temperature regulation

- HVAC applications
- Remote monitoring and control
- Digital telephone answering machine
- Fuel management systems
- Voice enabled appliances
- Cable test equipment
- Electric energy meter with embedded power line modem
- Underwater acoustics
- Glass breakage detection and security systems
- Traffic light control
- Identification tag readers

In addition to the above *general applications* of each chip, the following are unique to a specific chip:

56F801 unique applications include:

- Pumps
- Industrial fans
- Exercise equipment
- Smart appliances
- Compressors
- Noise cancellation
- HVAC
- Remote monitoring
- Tachometers
- Cable test equipment
- General purpose devices

56F802 unique applications include:

- Consumer electronics
- Motion control
- Limit switches
- HVAC

- Home appliances
- Encoders
- Engine management
- Tachometers
- Power supply and control
- Motor control
- Industrial control including lighting, power, automation, and HVAC
- General purpose devices

56F803 unique applications include:

- Steppers and encoders
- Engine management
- HVAC
- UPS
- Home security
- Automotive control
- Temperature control
- Compressors
- General purpose devices

56F805 unique applications include:

- Regenerative drives
- Glass breakage detection
- Critical reliability drives
- Cable test equipment
- Remote monitoring
- Automotive control
- Winder and pullers
- General purpose devices

56F807 unique applications include:

- Conveyors
- UPS

- Servo drives
- Fuel management systems
- Underwater acoustics
- Industrial frequency inverters
- Noise cancellation
- Lifts, elevators and cranes
- General purpose devices

1.2 DSP56800 Family Description

The 56800 core is based on a Harvard architecture consisting of three execution units operating in parallel. The MCU-style programming model and optimized instruction set allow straightforward generation of efficient, compact controller and control code. The instruction set is highly efficient for C-compilers, enabling rapid development of optimized control applications.

The 56F80x chip set supports program execution from either internal or external memories, providing two-external dedicated interrupt lines and up to 32 General-Purpose Input/Output (GPIO) lines. The controller includes Program Flash and Data Flash, each programmable through the Joint Test Action Group (JTAG) port, with program RAM and data RAM. With the exception of the 56F801 and 56F802, these controllers also supports program execution for external memory. The 56800 core is capable of accessing two data operands from the on-chip data RAM per instruction cycle.

A Boot Flash is incorporated for easy customer-inclusion of field-programmable software routines and can be used to program the main program and Data Flash memory areas. Both Program and Data Flash memories can be both independently bulk and page erased; however, while erasure is being performed on the main Program and/or Data Flash memories, the Boot Flash should not be erased. All Flash memories can be erased at once, if executing from a programmable RAM.

A key application-specific feature of the device is the inclusion of Pulse Width Modulator (PWM) modules. These modules each incorporate up to six complementary, individually programmable PWM signal outputs. To enhance motor control functionality, each module, except in the 56F801 and 56F802, is also capable of supporting six independent PWM functions, for a total of 12 PWM outputs. Complementary operation permits programmable deadtime insertion, distortion correction through current sensing by software, and separate top and bottom output polarity control. The up-counter value is programmable to support a continuously variable PWM frequency. Both edge- and center-aligned synchronous pulse width-control, full 0 to 100 percent modulation, are supported. The device is capable of controlling most motor

types: AC Induction Motors (ACIM), both Brushless (BLDC) and Brush DC motors (BDC), Switched (SRM) and Variable Reluctance Motors (VRM), and stepper motors.

Fault protection and cycle-by-cycle current limiting are incorporated in PWMs with sufficient output drive capability to directly drive standard opto-isolators. A *smoke-inhibit* write-once protection feature for key parameters is also included. A patented PWM waveform distortion correction circuit is provided as well. Each PWM is double-buffered and includes interrupt controls permitting integral reload rates programmable from one to 16. The PWM modules provide a reference output to synchronize the Analog-to-Digital Converters (ADC). Included are several four-input multiplexed 12-bit Analog-to-Digital Converters (ADC) and Quadrature Decoders. Each is capable of capturing transitions on the two-phase inputs, permitting generation of a number proportional to actual position. Speed computation capabilities accommodate both fast and slow moving shafts. The integrated Watchdog Timer in the Quadrature Decoder can be programmed with a time-out value to alarm when no shaft motion is detected. Each input is filtered ensuring only true transitions are recorded. If any Quadrature Decoder is not required, it can be programmed to provide a Quad Timer alternate function using four 16-bit independent timers. This controller also provides a full set of standard programmable peripherals, including:

- Serial Communications Interface (SCI)
- Serial Peripheral Interface (SPI)
- Additional Quad Timers (TMR)

Any of these interfaces can be used as GPIOs if that function is not required. A Controller Area Network (CAN) interface, CAN version 2.0 A/B-compliant, an internal interrupt controller and dedicated GPIO, are also included on some of the parts.

1.3 Manual Organization

This manual is arranged in the following sections:

- **Chapter 1, “56F800 Family”**—provides a brief overview of each of the 56F80x devices. The chapter describes the structure of this document, and lists other documentation necessary to use these chips.
- **Chapter 2, “Pin Descriptions”**—presents a description of the pins on the 56F801/803/805/807 chips and how the pins are grouped into the various interfaces.
- **Chapter 3, “Memory and Operating Modes”**—delineates the on-chip memory, structures, registers, and interfaces.
- **Chapter 4, “Interrupt Controller (ITCN)”**—describes how the IPBus interrupt controller accepts interrupt requests from IPBus-based peripherals and presents them to the 56800 core.

- **Chapter 5, "Flash Memory Interface"**—presents the Program Flash, Data Flash, and Boot Flash features and registers.
- **Chapter 6, "External Memory Interface (EMI)"**—describes the external memory interface available on the 56F803, 56F805, and 56F807.
- **Chapter 7, "General Purpose Input/Output (GPIO)"**—specifies how the GPIO shares package pins with other peripherals on the chip.
- **Chapter 8, "Controller Area Network (CAN)"**—describes the capabilities of CAN as a communication controller.
- **Chapter 9, "Analog-to-Digital Converter (ADC)"**—presents features, functions and registers of the Analog-to-Digital Converter.
- **Chapter 10, "Quadrature Decoder"**—delineates the features and registers of the Quadrature Decoder.
- **Chapter 11, "Pulse Width Modulator Module (PWM)"**—describes the function, configuration and registers of the PWM.
- **Chapter 12, "Serial Communications Interface (SCI)"**—depicts the Serial Communications Interface (SCI), communicating with devices such as codecs, other controllers, microprocessors, and peripherals to provide the primary data input path.
- **Chapter 13, "Serial Peripheral Interface (SPI)"**—relates the Serial Peripheral Interface (SPI), communicating with external devices, such as Liquid Crystal Displays (LCDs) and Micro Controller Units (MCUs).
- **Chapter 14, "Quad Timer Module (TMR)"**—describes the available internal Quad Timer devices, including features and registers.
- **Chapter 15, "On-Chip Clock Synthesis (OCCS)"**—delineates the Internal Oscillator, Phase Lock Loop (PLL), and timer distribution chain for the 56F801/803/805/807.
- **Chapter 16, "Reset, Low Voltage, Stop and Wait Operations"**—provides data about the On-Chip Watchdog Timer and the real-time interrupt generator, and modes of operation.
- **Chapter 17, "OnCE Module"**—describes the specifics of the 56F801/803/805/807 On-Chip Emulation (OnCE™) module, accessed through the Joint Test Action Group (JTAG) port.
- **Chapter 18, "JTAG Port"**—relates specifics of the 56F801/803/805/807 JTAG port.
- **Appendix A, Programmer's Sheets**—is a compilation set of reference tables and programming sheets intended to simplify programming for the 56F801/803/805/807.
- **Glossary**—provides definitions of terms, acronyms, and register names used in this manual.

1.4 Additional information:

- See http://www.freescale.com/ for the most current BSDL listings.
- See device Technical Data Sheet for package and pin-out information.

Note: With the exception of errata documents, if any other Freescale document contains information that conflicts with the information in the device data sheet, the data sheet should be considered to have the most current and correct data.

1.5 Manual Conventions

The following conventions are used in this manual:

- Bits within registers are always listed from Most Significant Bit (MSB) to Least Significant Bit (LSB). Other manuals use the opposite convention, with bits listed from LSB to MSB.
- Bits within a register are indicated AA[n:0] when more than one bit is involved in a description. For purposes of description, the bits are presented as if they are contiguous within a register. However, this is not always the case. Refer to the programming model diagrams or to the programmer's sheets to see the exact location of bits within a register.
- When a bit is described as *set*, its value is set to 1. When a bit is described as *cleared*, its value is set to 0.
- Pins or signals asserted as low, made active when pulled to ground, have an overbar above their name. For example, the $\overline{SS0}$ pin is asserted low.
- Hex values are indicated with a dollar sign ($) preceding the hex value, as follows: $FFFB is the X-memory address for the Interrupt Priority Register (IPR).
- Code examples are displayed in a mono-spaced font, illustrated in **Example 1-1.**

Example 1-1. Sample Code Listing

```
BFSET  #$0007,X:PCC ; Configure:                          line 1
                    ; MISO0, MOSI0, SCK0 for SPI master   line 2
                    ; ~SS0 as PC3 for GPIO                line 3
```

- Pins or signals listed in code examples asserted as low have a tilde in front of their names. In the previous example, line three above refers to the $\overline{SS0}$ pin (shown as ~SS0).
- The word *reset* is used in three different contexts in this manual.

— 1) There is a *reset* pin. It is always written as $\overline{\text{RESET}}$.

— 2) The processor state occurring when the $\overline{\text{RESET}}$ pin is asserted is always written as *Reset.*

— 3) Power and COP reset

- The word *reset* referring to the *function* is written in lower case with a leading capital letter as grammar dictates.
- The word *pin* is a generic term for any pin on the chip.
- The word *assert* means a high true (active high) signal is pulled high to V_{DD} or a low true (active low) signal is pulled low to ground.
- The word *deassert* means a high true signal is pulled low to ground, or a low true signal is pulled high to $V_{DD,}$ illustrated in **Table 1-1.**
- Shaded areas in registers represent reserved bits. They are written as zero, ensuring future compatibility.
- Throughout this manual, data memory locations are noted as X:$0000 and program memory locations are noted as P:$0000, where $ represents a memory location in hex.
- The PWM value registers are buffered. The value written does not take effect until the LDOK bit is set and the next PWM load cycle begins. Reading PWMVALx reads the value in a buffer and not necessarily the value the PWM generator is currently using.

Table 1-1. Pin Conventions

Signal/Symbol	Logic State	Signal State	Voltage
$\overline{\text{PIN}}$	True	Asserted	Ground[1]
$\overline{\text{PIN}}$	False	Deasserted	V_{DD}[2]
PIN	True	Asserted	V_{DD}
PIN	False	Deasserted	Ground

1. Ground is an acceptable low voltage level. See the appropriate data sheet for the range of acceptable low voltage levels (typically a TTL logic low).
2. V_{DD} is an acceptable high voltage level. See the appropriate data sheet for the range of acceptable high voltage levels (typically a TTL logic high).

1.6 Architectural Overview

The 56F801/803/805/807 family consists of the DSP56800 core, program and data memory, and peripherals useful for embedded control applications. Block diagrams for each chip describing the differences in available peripheral sets and memory illustrated in the following figures: **Figure 1-1**, **Figure 1-2**, **Figure 1-3**, **Figure 1-4**, and **Figure 1-5**.

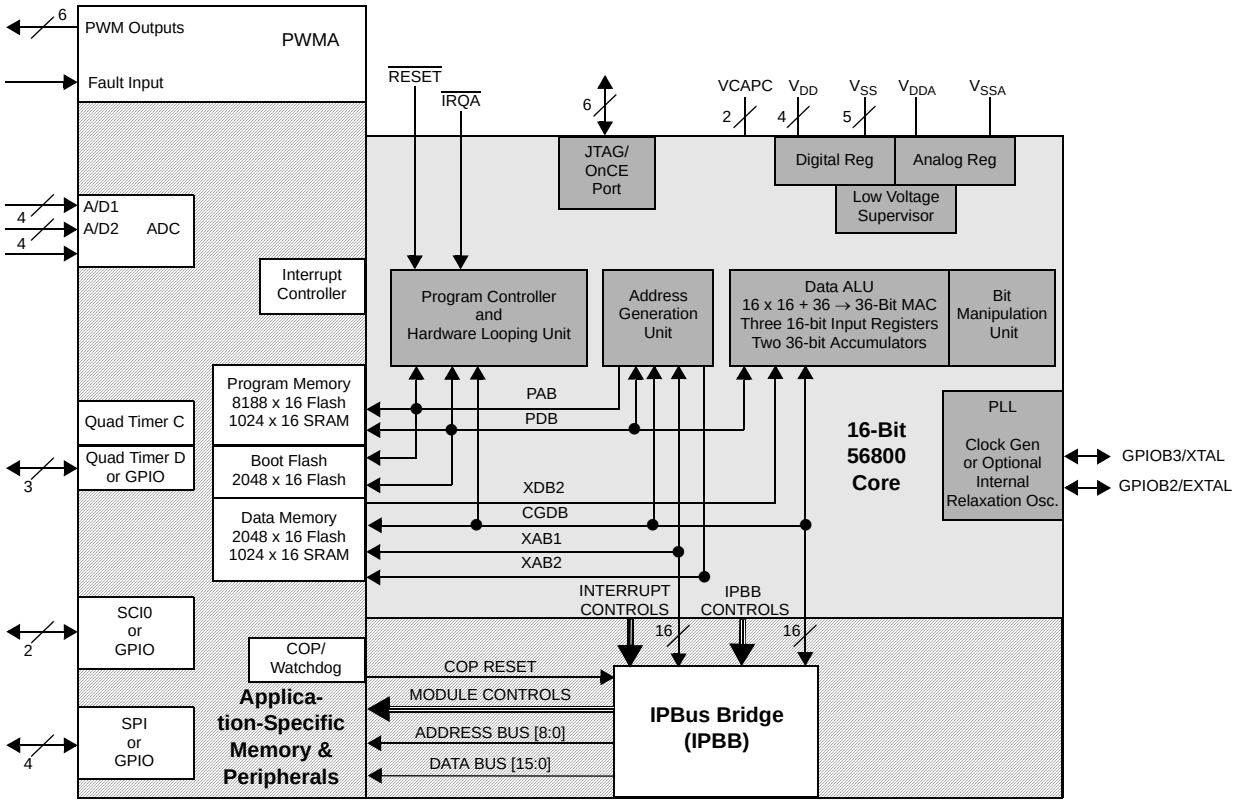
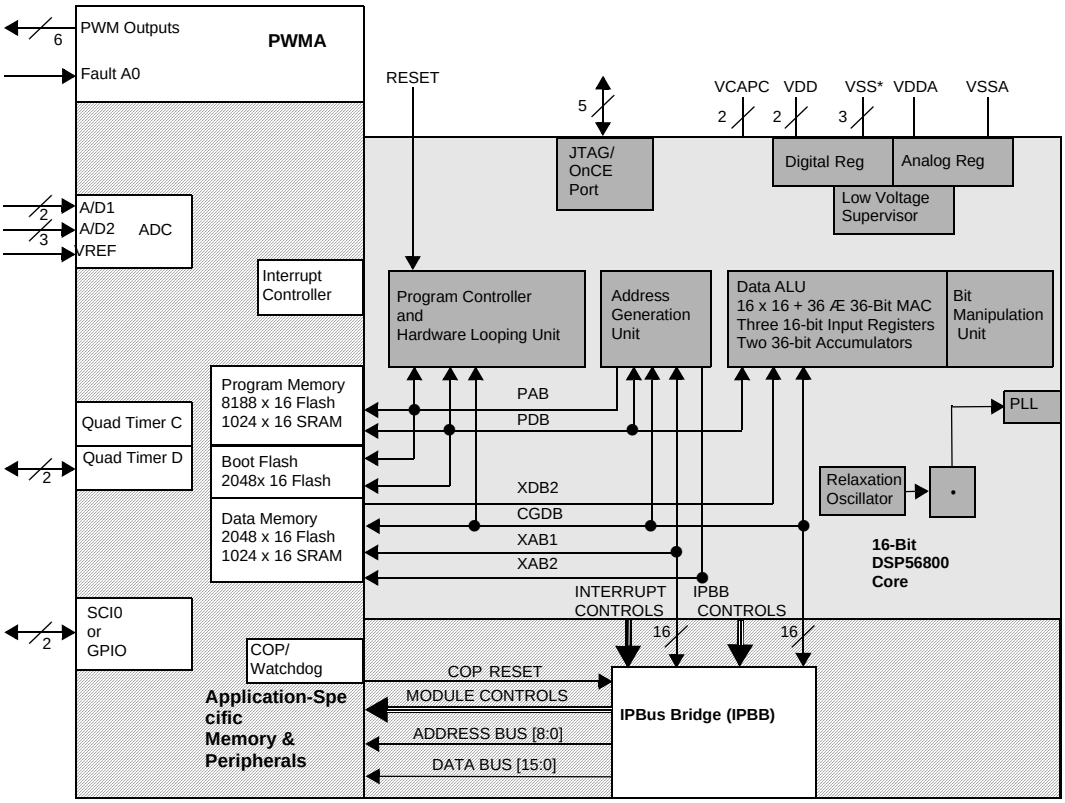


Figure 1-1. 56F801 Functional Block Diagram



*includes TCS pin which is reserved for factory use and is tied to VSS

Figure 1-2. 56F802 Functional Block Diagram

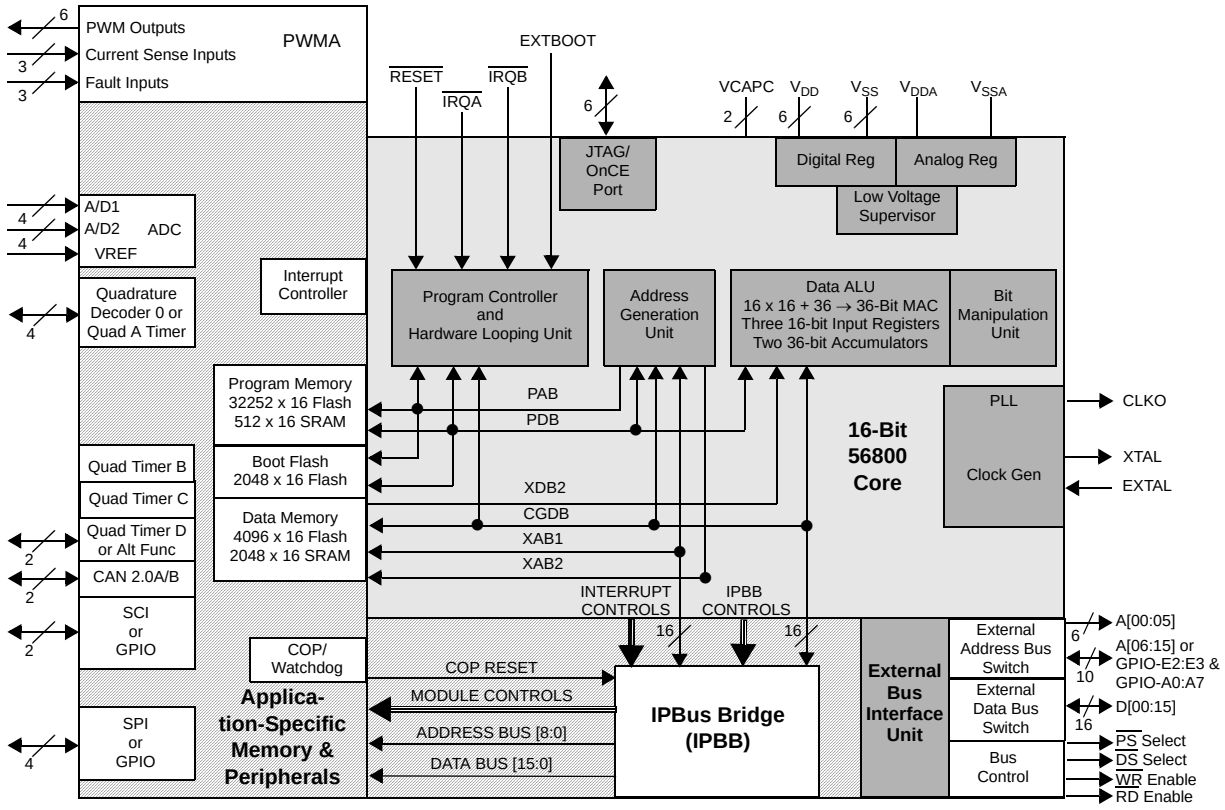


Figure 1-3. 56F803 Functional Block Diagram

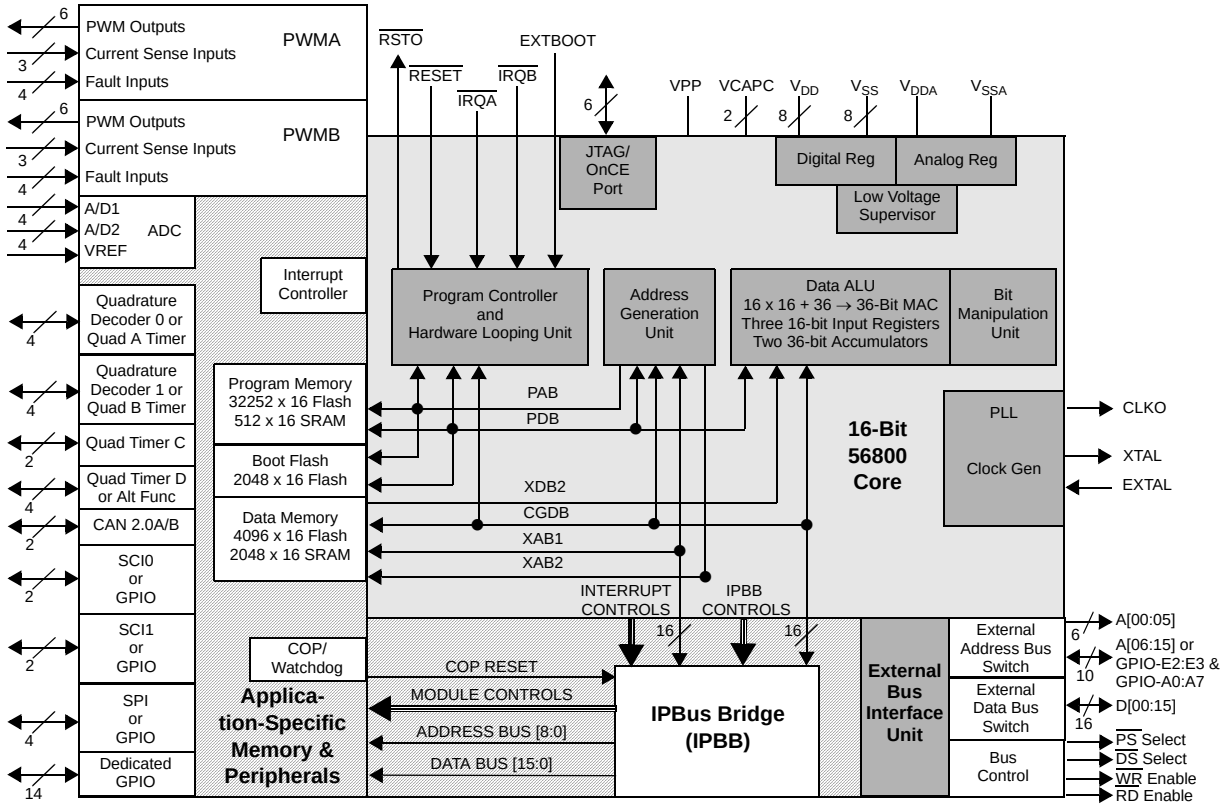


Figure 1-4. 56F805 Functional Block Diagram

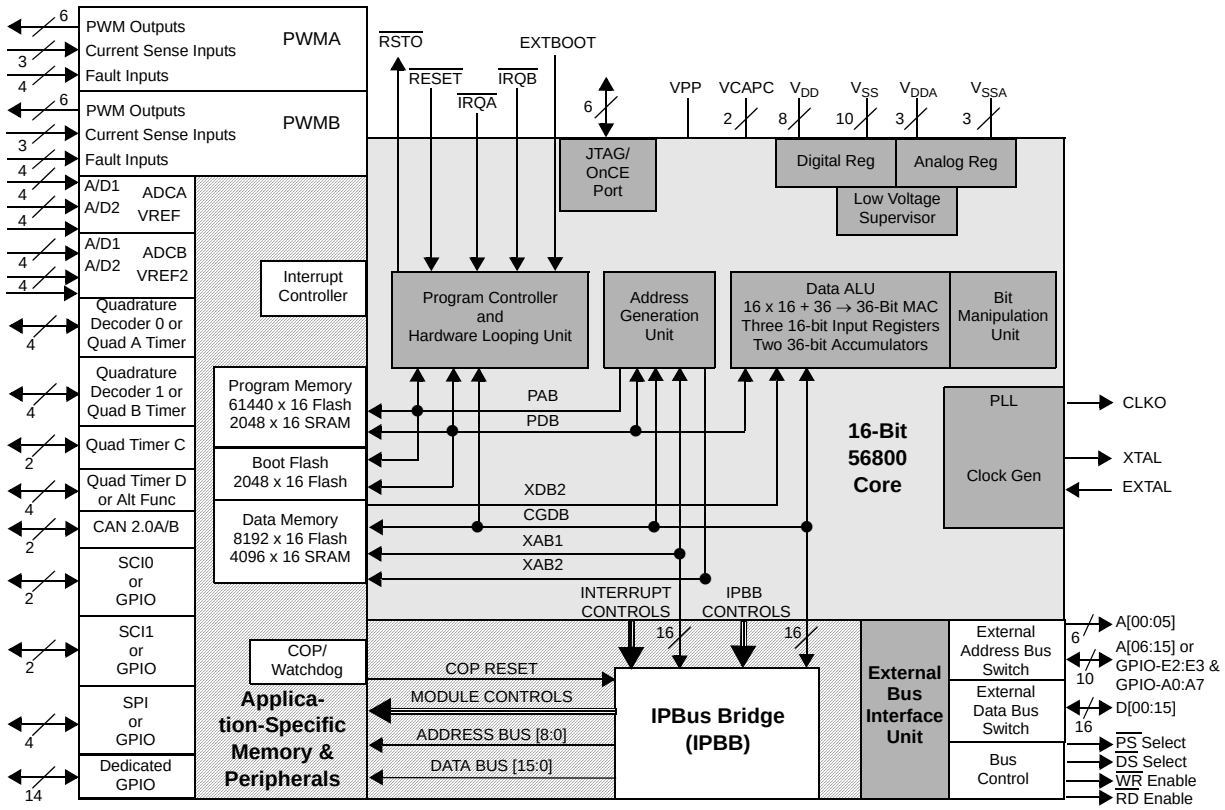


Figure 1-5. 56F807 Functional Block Diagram

Table 1-2. Feature Matrix

Feature	56F801	56F802	56F803	56F805	56F807
Speed (MIPS)	40	40	40	40	40
Program Flash	8188 X 16	8K X 16	32252 X 16	32252 X 16	61436 X 16
Data Flash	2K X 16	2K X 16	4K X 16	4K X 16	8K X 16
Program RAM	1K X 16	1K X 16	512 X 16	512 X 16	2K X 16
Data RAM	1K X 16	1K X 16	2K X 16	2K X 16	4K X 16
Boot Flash	2K X 16	2K X 16	2K X 16	2K X 16	2K X 16
Oscillator	Internal Relaxation or External Crystal	Internal Relaxation	External Crystal	External Crystal	External Crystal
PLL	1	1	1	1	1
SCI	1	1	1	2	2
SPI	1	0	1	1	1
Watchdog	1	1	1	1	1
General Purpose Timer (MAX)	2 Quad Timers	2 Quad Timers w/ 2 external pins	4 Quad Timers	4 Quad Timers	4 Quad Timers
Dedicated GPIO	0	0	0	14	14
Muxed GPIO	11	4	16	18	18
JTAG/OnCE	1	1	1	1	1
Interrupt Controller	1	1	1	1	1
PWM (6 Channel)	1	1	1	2	2
CAN	0	0	1	1	1
ADC	2, 4-input, 12-bit with analog mux	12-bit (1 x 2 channel and 1 x 3 channel)	2, 4-input, 12-bit with analog mux	2, 4-input, 12-bit with analog mux	4, 4-input, 12-bit with analog mux
Quadrature Decoder	0	—	1	2	2
Low-V Interrupt	1	1	1	1	1
External Bus	0	0	1	1	1
Package	48 LQFP	32 LQFP	100 LQFP	144 LQFP	160 LQFP & 160 MBGA

1.7 DSP56800 Core Description

The DSP56800 core consists of functional units operating in parallel to increase throughput of the machine. The DSP56800 core consists of three execution units operating in parallel. These units allow as many as six operations during each instruction cycle. The MPU-style programming model and optimized instruction set allow straightforward generation of efficient, compact controller and control code. The instruction set is also highly efficient for C-compilers. Major features of the DSP56800 core include the following:

- Efficient 16-bit DSP56800 family controller engine with dual Harvard architecture
- As many as 40 million instructions per second (MIPS) at 80MHz core frequency
- Single-cycle 16- × 16-bit parallel Multiplier-Accumulator (MAC)
- Two 36-bit accumulators, including extension bits
- 16-bit bidirectional barrel shifter
- Parallel instruction set with unique controller addressing modes
- Hardware DO and REP loops
- Three internal address buses and one external address bus available
- Four internal data buses and one external data bus available
- Instruction set supports both controller and controller functions
- Controller style addressing modes and instructions for compact code
- Efficient C-compiler and local variable support
- Software subroutine and interrupt stack with depth limited only by memory
- JTAG/OnCE debug programming interface

1.7.1 56800 Core Block Diagram

An overall block diagram of the DSP56800 core architecture is illustrated in **Figure 1-6.** The DSP56800 core is fed by an Internal program and Data memory, an External Memory Interface (EMI), and various peripherals suitable for embedded applications. The blocks of the DSP56800 core include the following:

- Data Arithmetic Logic Unit (Data ALU)
- Address Generation Unit (AGU)
- Program controller and hardware looping unit
- Bit Manipulation Unit
- OnCE port

- Interrupt controller
- External bus bridge
- Address buses
- Data buses

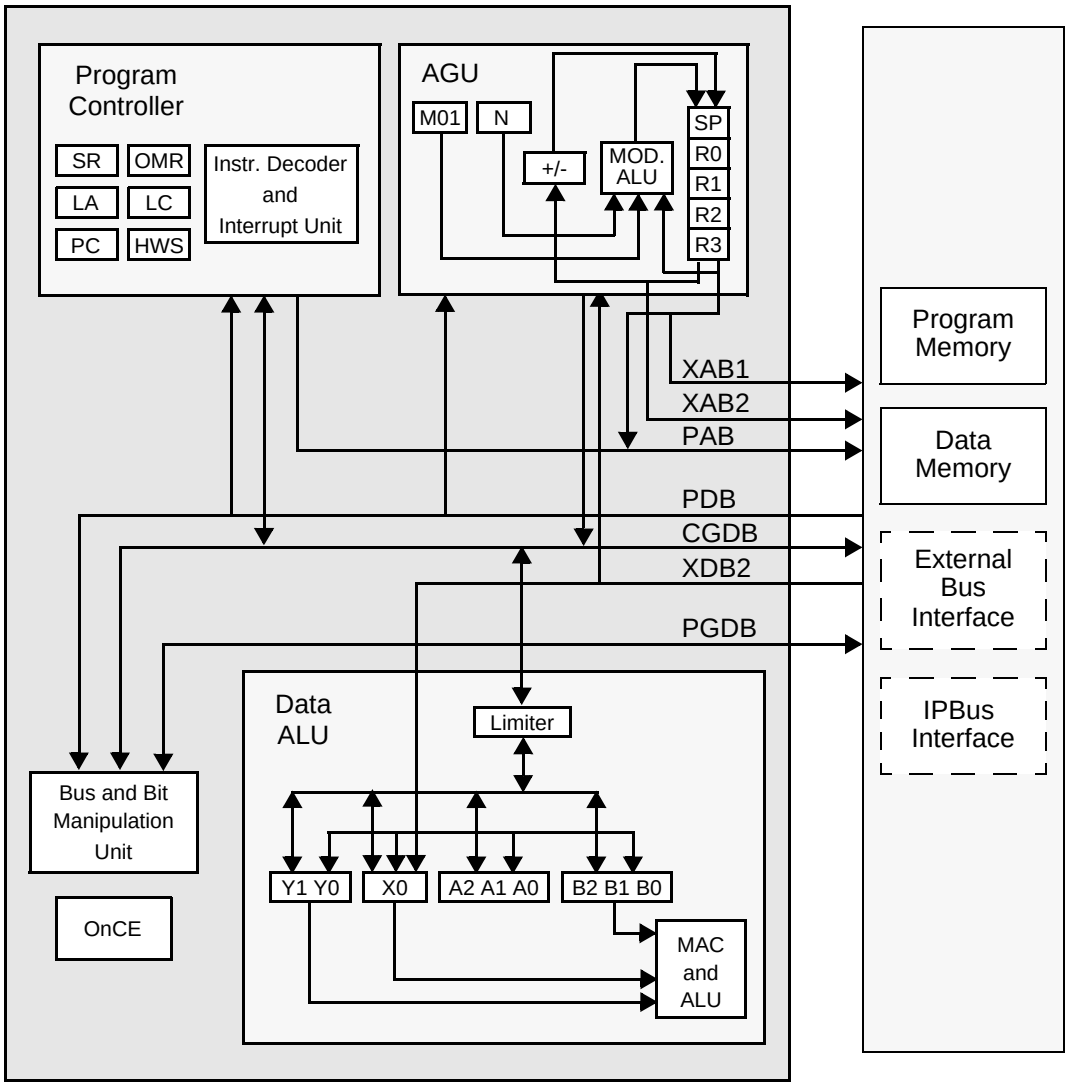


Figure 1-6. 56800 Core Block Diagram

The program controller, AGU and Data ALU each contain a discrete register set and control logic so each can operate independently and in parallel with the others. Likewise, each functional unit interfaces with other units with memory, and with memory-mapped peripherals over the core's internal address and data buses, illustrated in **Figure 1-7.**

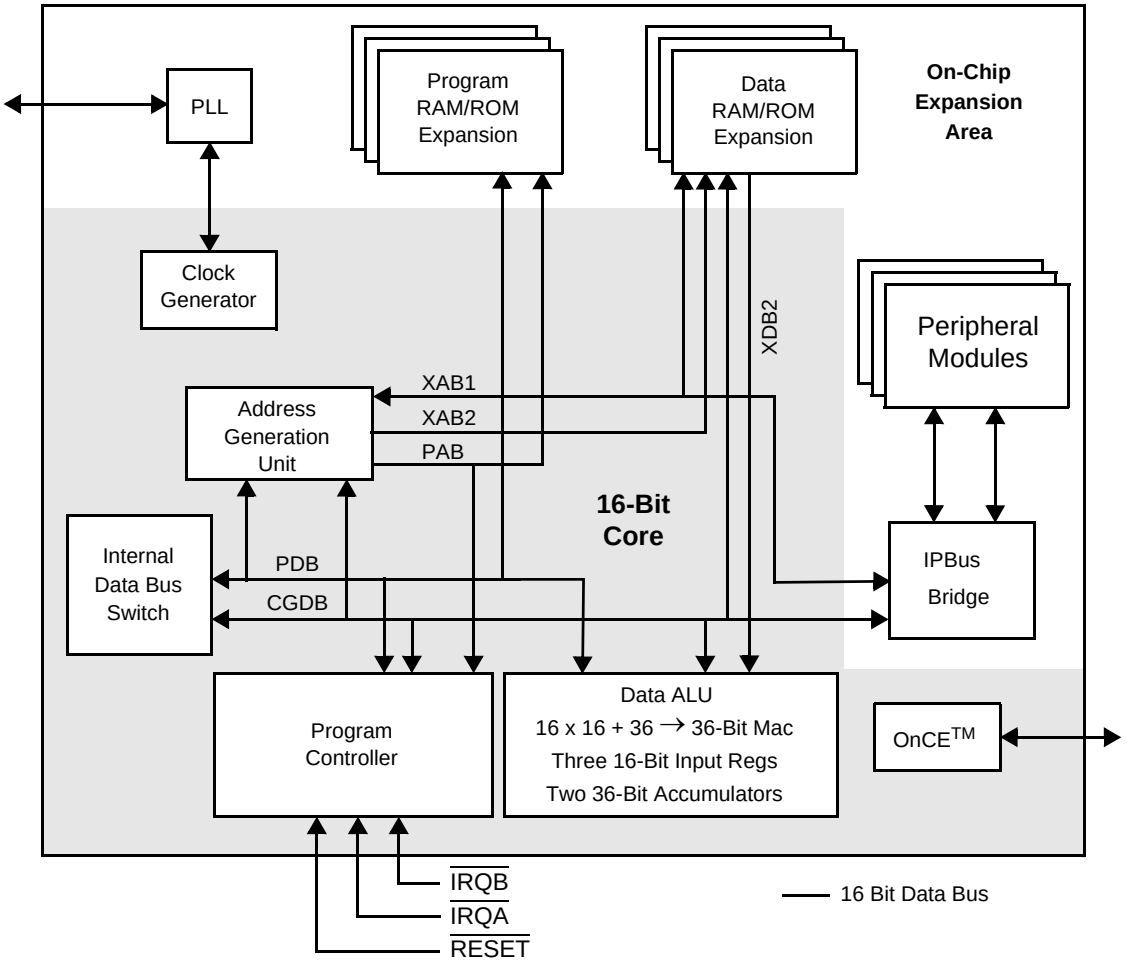


Figure 1-7. DSP56800 Bus Block Diagram

It is possible in a single instruction cycle for the program controller to be fetching a first instruction, the AGU to generate two addresses for a second instruction, and the Data ALU to perform a multiply in a third instruction. In a similar manner, the Bit Manipulation Unit can perform an operation of the third instruction described above instead of the multiplication in the Data ALU. The architecture is pipelined to take advantage of the parallel units and significantly decrease the execution time of each instruction.

1.7.2 Data Arithmetic Logic Unit (Data ALU)

The Data Arithmetic Logic Unit (Data ALU) performs all of the arithmetic and logical operations on data operands. It contains:

- Three 16-bit input registers
- Two 32-bit accumulator registers
- Two 4-bit accumulator extension registers
- One parallel, single cycle, non-pipelined MAC unit
- An accumulator shifter
- One data limiter
- One MAC output limiter
- One 16-bit barrel shifter

The Data ALU is capable of performing the following in one instruction cycle:

- Multiplication
- Multiply-accumulate with positive or negative accumulation
- Addition
- Subtraction
- Shifting
- Logical operations

Arithmetic operations are completed using two’s-complement fractional or integer arithmetic. Support is also provided for unsigned and multi-precision arithmetic.

Data ALU source operands can be 16, 32, or 36 bits and can originate from input registers and/or accumulators. ALU results are stored in one of the accumulators. In addition, some arithmetic instructions store their 16-bit results in any of the three Data ALU input registers or write directly to memory. Arithmetic operations and shifts have a 16-bit or 36-bit result and logical operations are performed on 16-bit operands yielding 16-bit results. Data ALU registers can be read or written by the Core Global Data Bus (CGDB) as 16-bit operands, and the X zero register can also be written by the X Data Bus two (XDB2) with a 16-bit operand.

1.7.3 Address Generation Unit (AGU)

The Address Generation Unit (AGU) performs all of the effective address calculations and address storage necessary to address data operands in memory. This unit operates in parallel with other chip resources to minimize address generation overhead. It contains two ALUs, allowing the generation of up to two 16-bit addresses every instruction cycle—one for either the XAB1 or

PAB bus and one for the XAB2 bus. The ALU can directly address 65,536 locations on the XAB1 or XAB2 bus and 65,536 locations on the Program Address Bus (PAB), for a total capability of 131,072 16-bit data words. Hooks are provided on the DSP56800 core to expand this address space. Its arithmetic unit can perform linear and modulo arithmetic.

1.7.4 Program Controller and Hardware Looping Unit

The Program Controller performs:

- Instruction Prefetch
- Instruction Decoding
- Hardware Loop Control
- Interrupt (Exception) Processing

Instruction execution is carried out in other core units such as the data ALU or AGU. The program controller consists of a Program Counter (PC) unit, Hardware Looping Control Logic, Interrupt Control Logic, and Status and Control registers.

Two interrupt control pins provide input to the program interrupt controller. The external Interrupt Request A ($\overline{\text{IRQA}}$) pin and the external Interrupt Request B ($\overline{\text{IRQB}}$) pin receive interrupt requests from external sources.

The $\overline{\text{RESET}}$ pin resets the 56F803/805/807. When it is asserted, it initializes the chip and places it in the Reset state. When the $\overline{\text{RESET}}$ pin is deasserted, the initial Chip Operating mode is latched into the Operating Mode Register (OMR) based upon the value present on the EXTBOOT pin. The 56F801/802 are internally pulled low. Refer to **Section 3.3.2** for additional details.

1.7.5 Bit Manipulation Unit

The Bit Manipulation Unit (BMU) performs bitfield manipulations on X-memory words, peripheral registers, and registers on the DSP56800 core. It is capable of testing, setting, clearing, or inverting any bits specified in a 16-bit mask. For branch-on-bitfield instructions, this unit tests bits on the upper or lower byte of a 16-bit word, meaning the mask tests a maximum of eight bits at a time.

Transfers between buses are accomplished in the bus unit. The bus unit is similar to a switch matrix, and can connect any two of the three data buses together without adding any pipeline delays. This is required for transferring a core register to a peripheral register, for example, because the core register is connected to the CGDB bus.

As a general rule, when reading any register less than 16 bits wide, unused bits are read as 0. Reserved and unused bits should always be written with a 0 ensuring future compatibility.

1.7.6 Address and Data Buses

Addresses are provided to the internal X-data memory on two unidirectional 16-bit buses—X Address Bus One (XAB1) and X Address Bus Two (XAB2). Program memory addresses are provided on the unidirectional 19-bit Program Address Bus (PAB). Note the XAB1 can provide addresses for accessing both internal and external memory, whereas the XAB2 can only provide addresses for accessing internal read-only memory. The External Address Bus (EAB) provides addresses for external memory.

Data movement on the 56F801/803/805/807 occurs over three bidirectional 16-bit buses—the CGDB, the Program Data Bus (PDB), the PGDB, and also over one unidirectional 16-bit bus: the X Data Bus Two (XDB2). When one memory access is performed, data transfer between the data ALU and the X data memory occurs over the CGDB. When two simultaneous memory reads are performed transfers occur over the CGDB and the XDB2. All other data transfers to core blocks occur over the CGDB, transferring all to and from peripherals over the PGDB. Instruction word fetches occur simultaneously over the PDB. The External Data Bus (EDB) provides bidirectional access to external data memory.

The bus structure supports:

1. General Register-to-Register
2. Register-to-Memory
3. Memory-to-Register Transfers

Each can transfer up to three 16-bit words in the same instruction cycle. Transfers between buses are accomplished in the Bit Manipulation Unit. **Table 1-3** lists the address and data buses for the DSP56800 core.

Table 1-3. 56F800 Address and Data Buses

Bus	Bus Name	Bus Width, Direction, and Use
XAB1	X Address Bus 1	16-bit, unidirectional, internal and external memory address
XAB2	X Address Bus 2	16-bit, unidirectional, internal memory address
PAB	Program Address Bus	16-bit, unidirectional, internal memory address
EAB	External Address Bus	16-bit, unidirectional, external memory address
CGDB	Core Global Data Bus	16-bit, bidirectional, internal data movement
PDB	Program Data Bus	16-bit, bidirectional, instruction word fetches
PGDB	Peripheral Global Data Bus	16-bit, bidirectional, internal data movement
XDB2	X Data Bus 2	16-bit, unidirectional, internal data movement
EDB	External Data Bus	16-bit, bidirectional, external data movement

Note: The 56F801/802 does not have external memory capabilities; therefore the external address and data buses are not relevant to these part.

1.7.7 On-Chip Emulation (OnCE) Module

The On-Chip Emulation (OnCE) module allows interaction in a debug environment with the 56F800 core and its peripherals. Its capabilities include examining registers, memory, or on-chip peripherals; setting breakpoints in memory; and stepping or tracing instructions. It provides simple, inexpensive, and speed independent access to the 56F800 core for sophisticated debugging and economical system development. The JTAG port allows access to the OnCE module and through the 56F801/803/805/807 to its target system, retaining debug control without sacrificing other user accessible on-chip resources. This technique eliminates the costly cabling and the access to processor pins required by traditional emulator systems. The OnCE interface is fully described in **Chapter 17, "OnCE Module"**.

1.7.8 On-Chip Clock Synthesis Block

The clock synthesis module generates the clocking for the 56F801/803/805/807. It generates clocks used by the DSP56800 core and 56F801/803/805/807 peripherals. It contains a PLL with the capacity to multiply up the frequency or can be bypassed. Additionally there is a prescaler/divider used to distribute clocks to peripherals and to lower power consumption on the 56F801/803/805/807. It also selects which clock, if any, is routed to the CLKO pin of the 56F803, 56F805, and 56F807. Neither the 56F801 nor 56F802 has a CLKO pin.

56F801 allows oscillation flexibility between using the external crystal oscillator or an on-chip relaxation oscillator, thereby lowering the system cost and provides two additional GPIO lines (because the XTAL and EXTAL pins are not needed for the external crystal).

1.7.9 Oscillators

The 56F803, 56F805 and 56F807 are clocked either from an external crystal or external clock generator input. The 56F801 and 56F802 can be clocked from an on-chip relaxation oscillator, eliminating the need for the external crystal and lowering system cost. The 56F801 can also run from an external crystal or clock generator in the same manner as the 56F803, 56F805, and 56F807.

- Crystal Oscillator uses an 8MHz crystal
- Optionally, it can use ceramic resonator in place of crystal
- Oscillator output can optionally be divided down by a programmable prescaler
- 56F801 and 56F802 has an on-chip relaxation oscillator

1.7.10 PLL

- The PLL will generate an interrupt to instruct the controller to gracefully shut down the system in the event reference clock is stopped.
- The PLL will continue running for at least 100 instruction cycles if oscillator source is removed.
- The PLL generates output frequencies up to 80MHz.
- The PLL can be bypassed to use oscillator or prescalar outputs directly, either on-chip or external crystal oscillator can be used for 56F801.
- Clock control

A clock gear shifter guarantees smooth transition from one clock source to the next during normal operation. Clock sources available for normal operation include:

- Crystal Oscillator output
- Relaxation Oscillator output, *only* 56F801/802
- PLL output
- Programmable prescalar output, a divided down version of the oscillator clock with legal divisors as one, two, four, or eight

1.7.11 Resets

- Integrated POR release occurs when V_{DD} exceeds 1.8V.
- Integrated low voltage detect interrupts the host processor when V_{DD} drops below 2.2V in the core or 2.7V in the I/O.

Note: Voltage levels are set to guarantee the host processor is still running at speed. There is nominally about 50mV of hysteresis present on each of the low voltage interrupt inputs.

1.7.12 Core Voltage Regulator

- Chip supply voltage = 3.3V
- Core voltage = 2.5V plus or minus 10 percent

1.7.13 IPBus Bridge

The IPBus Bridge converts data memory and interrupt interfaces to IPBus-compliant interface for peripherals.

This IPBus bridge allows communication between the core and peripherals utilizing the CGDB for data and XAB for addresses. The IPBus Bridge translates the four-phase clock bus protocol of the 56800 core to the single clock environment of the IPBus protocol used to communicate with the peripherals. All IPBus transfers are completed in one core clock cycle.

The 56800 only supports 16-bit word transfers on word boundaries.

The IPBus Bridge also provides upper level address decoding and peripheral module enable generation.

Note: All peripherals on the 56F801/803/805/807 except the COP/Watchdog Timer run off the IPBus clock frequency. It is the chip operating frequency divided by two. The maximum frequency of operation is 80MHz, correlating to a 40MHz IPBus clock frequency.

1.8 Memory Modules

- Harvard architecture permits as many as three simultaneous accesses to program and data memory.
- Off-Chip memory expansion capabilities (56F803, 56F805, and 56F807 only)
  - — As much as 64K × 16 Data Memory
  - — As much as 64K × 16 Program Memory
  - — External memory expansion port programmable for 0, 4, 8, or 12 wait states
- On-Chip memory, depending on specific chip selected
  - **— 56F801**
    - — 8K × 16-bit words of Program Flash
    - — 1K × 16-bit words of Program RAM

- 1K × 16-bit words of Data RAM
- 2K × 16-bit words of Data Flash
- 2K × 16-bit words of Boot Flash

— 56F802

- 8K × 16-bit words of Program Flash
- 1K × 16-bit words of Program RAM
- 1K × 16-bit words of Data RAM
- 2K × 16-bit words of Data Flash
- 2K × 16-bit words of Boot Flash

— 56F803

- 32252 × 16-bit words of Program Flash
- 512 × 16-bit words of Program RAM
- 2K × 16-bit words of Data RAM
- 4K × 16-bit words of Data Flash
- 2K × 16-bit words of Boot Flash

— 56F805

- 32252 × 16-bit words of Program Flash
- 512 × 16-bit words of Program RAM
- 2K × 16-bit words of Data RAM
- 4K × 16-bit words of Data Flash
- 2K × 16-bit words of Boot Flash

— 56F807

- 60K × 16-bit words of Program Flash
- 2K × 16-bit words of Program RAM
- 4K × 16-bit words of Data RAM
- 8K × 16-bit words of Data Flash
- 2K × 16-bit words of Boot Flash

1.8.1 Program Flash

- Single port memory compatible with the pipelined program bus structure
- Split-gate cell, NOR type structure

- Single cycle reads at 40MHz
- Intelligent word programming feature
- Memory is organized into a two row information block (= 128 bytes) and main memory block
- Pages are 512 bytes long
- Intelligent page erase and mass erase modes
- Can be programmed and erased under software control
- Optional interrupt on completion of intelligent program and erase functions

1.8.2 Program RAM

- Single port RAM is compatible with the pipelined program bus structure
- Single cycle reads at 40MHz

1.8.3 Data Flash

- Single port memory compatible with the pipelined data bus structure
- Muxing provided so this memory can be read from PAB, XAB1 or XAB2 databases
- Split-gate cell, NOR type structure
- Single cycle reads at 40MHz across the automotive temperature range
- Intelligent word programming feature
- Intelligent page erase and mass erase modes
- Can be programmed under software control in the user's system

1.8.4 Data RAM

- Single read, dual read or single write memory compatible with the pipelined data bus structure
- Single cycle reads/writes at 40MHz

1.9 56F801 Peripheral Blocks

The 56F801 provides the following peripheral blocks:

- Pulse Width Modulator (PWMA) module with six PWM outputs, one fault inputs, fault tolerant design with deadtime insertion, supports both center- and edge-aligned modes
- 12-bit, Analog-to-Digital Convertors (ADCs), are support by two simultaneous conversions with two four-pin multiplexed inputs, ADC and PWM modules are synchronized
- General-purpose Quad Timer D with three-pins, or three additional GPIO lines
- Serial Communication Interface (SCI0) with two pins, or two additional GPIO lines
- Serial Peripheral Interface (SPI) with configurable four pin port, or four additional GPIO lines
- Computer Operating Properly (COP) Watchdog Timer
- One dedicated external interrupt pins
- Eleven multiplexed GPIO pins
- External reset pin for hardware reset
- JTAG/OnCE
- Software-programmable, Phase Lock Loop (PLL) based frequency synthesizer for the core clock
- Oscillation flexibility between external crystal oscillator or on-chip relaxation oscillator for lower system cost and two additional GPIO lines

1.10 56F802 Peripheral Blocks

- Pulse Width Modulator (PWM) with six PWM outputs with deadtime insertion and fault protection; supports both center- and edge-aligned modes
- Two 12-bit, Analog-to-Digital Converters (ADCs), 1 x 2 channel and 1 x 3 channel, which support two simultaneous conversions; ADC and PWM modules can be synchronized
- Two general-purpose Quad Timers with two external pins (or two GPIO)
- Serial Communication Interface (SCI) with two pins (or two additional GPIO lines)
- Four multiplexed General-Purpose I/O (GPIO) pins
- Computer-Operating Properly (COP) Watchdog Timer
- External interrupts via GPIO
- Trimmable on-chip relaxation oscillator
- External reset pin for hardware reset
- JTAG/On-Chip Emulation (OnCE™) for unobtrusive, processor speed-independent debugging
- Software-programmable, Phase Lock Loop (PLL) based frequency synthesizer for the controller core clock

1.11 56F803 Peripheral Blocks

The 56F803 provides the following peripheral blocks:

- Pulse Width Modulator (PWMA) module with six PWM outputs, three current sense inputs, and three fault inputs, fault tolerant design with deadtime insertion, supports both center- and edge-aligned modes
- Twelve bit, Analog-to-Digital Convertors (ADCs), supporting two simultaneous conversions with two, four-pin multiplexed inputs, ADC and PWM modules are synchronized
- Quadrature Decoder (Quad Dec0) with four inputs, or additional Quad Timer A
- General-purpose Quad Timer D with two pins
- General-purpose Quad Timer B and C with no external pins
- CAN 2.0 A/B module with two-pin ports for transmit and receive
- Serial Communication Interface (SCI0) with two pins, or two additional GPIO lines
- Serial Peripheral Interface (SPI) with configurable four-pin port, or four additional GPIO lines
- Computer Operating Properly (COP) Watchdog Timer
- Two dedicated external interrupt pins
- Sixteen multiplexed GPIO pins
- External reset pin for hardware reset
- JTAG/OnCE
- Software-programmable, Phase Lock Loop (PLL) based frequency synthesizer for the core clock

1.12 56F805 Peripheral Blocks

The 56F805 provides the following peripheral blocks:

- Two Pulse Width Modulator (PWMA and PWMB) modules, each with six PWM outputs, three current sense inputs, and four fault inputs, fault tolerant design with deadtime insertion, supports both center- and edge-aligned modes
- Twelve bit, Analog-to-Digital Convertors (ADCs), supporting two simultaneous conversions with dual four-pin multiplexed inputs, ADC and PWM modules are synchronized
- Two Quadrature Decoders (Quad Dec0 and Quad Dec1), each with four inputs, or two additional Quad Timers A and B
- Two dedicated general-purpose Quad Timers totalling six pins: Timer C with two pins and Timer D with four pins
- CAN 2.0 A/B module with two-pin ports used to transmit and receive
- Two Serial Communication Interfaces (SCI0 and SCI1), each with two pins, or four additional GPIO lines

- Serial Peripheral Interface (SPI), with configurable four-pin port, or four additional GPIO lines
- Computer Operating Properly (COP) Watchdog Timer
- Two dedicated external interrupt pins
- Fourteen dedicated GPIO pins, 18 multiplexed GPIO pins
- External reset pin for hardware reset
- JTAG/OnCE
- Software-programmable, Phase Lock Loop (PLL) based frequency synthesizer for the core clock

1.13 56F807 Peripheral Blocks

The 56F807 provides the following peripheral blocks:

- Two Pulse Width Modulator modules (PWMA and PWMB), each with six PWM outputs, three current sense inputs endeavour fault inputs, fault tolerant design with deadtime insertion, supports both center- and edge-aligned modes
- Four 12-bit, Analog-to-Digital Convertors (ADCs), supporting four simultaneous conversions with quad four-pin multiplexed inputs, ADC and PWM modules are synchronized
- Two Quadrature Decoders (Quad Dec0 and Quad Dec1), each with four inputs, or two additional Quad Timers A and B
- Two dedicated general purpose Quad Timers totalling six pins: Timer C with two pins and Timer D with four pins
- CAN 2.0 A/B module with two-pin ports for transmit and receive
- Two Serial Communication Interfaces (SCI0 & SCI1) each with two pins, or four additional GPIO lines
- Serial Peripheral Interface (SPI) with configurable four-pin port, or four additional GPIO lines
- Computer Operating Properly (COP) Watchdog Timer
- Two dedicated external interrupt pins
- Fourteen dedicated GPIO pins, 18 multiplexed GPIO pins
- External reset pin for hardware reset
- JTAG/OnCE
- Software-programmable, Phase Lock Loop (PLL) based frequency synthesizer for the core clock

1.14 Peripheral Descriptions

The IPBus Bridge converts data memory and interrupt interfaces to the IPBus-compliant interface for peripherals. This IPBus bridge allows for communication between the core and peripherals utilizing the CGDB for data and XAB for addresses.

Peripherals run off the IPBus clock at 40MHz. The IPBus clock frequency is half of the oscillator frequency. Interrupt Priority Register (IPR) and BCR run at 80MHz while the COP/Watchdog Timer runs at 40MHz.

1.14.1 External Memory Interface (EMI)

The 56F803, 56F805, and 56F807 provide an External Memory Interface (EMI). This port provides a total of 36 pins; 16 pins for an external address bus, 16 pins for an external data bus, and four pins for bus control.

1.14.2 General Purpose Input/Output Port (GPIO)

This port is configured so it is possible to generate interrupts when a transition is detected on any of its lower eight pins.

- **56F801**
  - 11 shared GPIO, multiplexed with other peripherals
- **56F802**
  - 4 shared GPIO, multiplexed with other peripherals
- **56F803**
  - 16 shared GPIO, multiplexed with other peripherals
- **56F805**
  - 14 dedicated GPIO pins
  - 18 shared GPIO, multiplexed with other peripherals
- **56F807**
  - 14 dedicated GPIO pins
  - 18 shared GPIO, multiplexed with other peripherals
- Each bit may be individually configured as an input or output
- Selectable enable for pull-up resistors
- Each bit can be configured as an interrupt input

1.14.3 Serial Peripheral Interface (SPI)

The Serial Peripheral Interface (SPI) is an independent serial communications subsystem allowing the 56F80x family to communicate synchronously with peripheral devices such as LCD display drivers, A/D subsystems, and MCU microprocessors. The SPI is also capable of interprocessor communication in a multiple master system. The SPI system can be configured as either a master or a slave device with high data rates. The SPI works in a demand-driven mode. In Master mode, a transfer is initiated when data is written to the SPI data register. A transfer is initiated by the reception of a clock signal in the Slave mode.

- Each 56F80x device has one SPI, except for the 56F802 which has none
- Full-duplex synchronous operation via four-wire interface
- Configurable for either master or slave operation
- Multiple slaves may be enabled via GPIO pins
- Double-buffered operation with separate transmit and receive registers

1.14.4 COP/Watchdog Timer & Modes of Operation Module

The Computer Operating Properly (COP) module monitors processor activity and provides an automatic reset signal if a failure occurs. Please refer to **Chapter 16.**

- 12-bit counter to provide 4096 different time-out periods
- COP timebase is the CPU clock divided by 16384
- At 80MHz, minimum time-out period is 205μs, maximum is 839ms, with a resolution of 205μs

1.14.5 JTAG/OnCE Port

The JTAG/OnCE port allows insertion of the 56F801/803/805/807 into a target system while retaining debug control. The JTAG port provides board-level testing capability for scan-based emulation compatible with the IEEE 1149.1a-1993 IEEE Standard Test Access Port and Boundary Scan Architecture specification defined by the JTAG. Five dedicated pins interface to a TAP containing a 16-state controller.

The OnCE module allows the user to interact in a debug environment with the DSP56800 core and its peripherals nonintrusively. Its capabilities include:

- Examining registers, memory, or on-chip peripherals
- Setting breakpoints in memory
- Stepping or tracing instructions

It provides simple, inexpensive, and speed-independent access to the DSP56800 core for sophisticated debugging and economical system development. The JTAG/OnCE port provides access to the OnCE module. Through the 56F801/803/805/807 to its target system, it retains debug control without sacrificing other user accessible on-chip resources.

1.14.6 Quadrature Decoder

- **56F801**and **56F802**
  - — No Quadrature Decoder
- **56F803**
  - — One Quadrature Decoder, or Timer A with four pins
- **56F805** and **56F807**
  - — Two Quadrature Decoders, or Timer A and B each with four pins

Each Quadrature Decoder:

- Integrates the two-phase output of the encoder to extract position
- INDEX input to reset the integration and/or integrate revolutions
- Accounts for direction of rotation in position and revolution integrals
- Includes an instant access to the general purpose timer for capturing all four transitions on the two-phase input, offering higher resolution input for slow moving shafts
- Optionally can be used as a single phase pulse accumulator
- Integral Watchdog Timer alarms the core when no shaft motion is detected
- Filtered inputs ensuring only true transitions are recorded

1.14.7 Quad Timer Module

- **56F801**
  - — Timer D with three pins
- **56F802**
  - — Timer A with two pins
- **56F803**
  - — Timer C with two pins
  - — Timer D with four pins
  - — Optional timer A with four pins muxed to quadrature decoder zero

- **56F805** and **56F807**
  - Timer C with two pins
  - Timer D with four pins
  - Optional Timer A with four pins muxed to Quadrature Decoder0
  - Optional Timer B with four pins muxed to Quadrature Decoder1

Each Quad Timer features:

- Four channels, independently programmable as input capture or output compare
- Each channel has its own timebase
- Each of four channels can use any of four timer inputs
- Rising edge, falling edge, or any edge input capture trigger
- Set, clear, or toggle output capture action
- Pulse Width Modulator (PWM) signal generation
- Programmable clock sources and frequencies, including external clock
- External synchronization input

1.14.8 Pulse Width Modulator (PWM) Module

- **56F801** and **56F802**
  - One Pulse Width Modulator (PWMA)
    - Zero current sense pins
    - One fault pin
- **56F803**
  - One Pulse Width Modulator (PWMA)
    - Three current sense pins each
    - Three fault pins each
- **56F805** and **56F807**
  - Two Pulse Width Modulator (PWMA and PWMB)
    - Three current sense pins
    - Four fault pins

Pulse Width Modulator (PWM) module specifically designed for motor control:

- Six independent outputs, or three complementary pairs of outputs
- Center-aligned or edge-aligned pulses

- 15-bit counters with programmable resolutions down to 25ns
- Automatic deadtime insertion for complementary outputs, including data toggling based on motor phase current polarity
- Half-cycle or multi-cycle PWM updates
- Buffered registers with interlock bit, preventing erroneous PWM loads
- Programmable outputs with 16mA sink and 10mA source current at V_{DD} = 3.3V
- Four programmable fault inputs to individually disable PWM channels
- Input pins to control the reset state of the PWM outputs

1.14.9 Analog-to-Digital Conversion (ADC)

- **56F801, 56F803,** and **56F805**
  - — Dual Analog-to-Digital Converter (ADC) modules—four inputs on each
  - — Eight total analog inputs
- **56F802**
  - — Dual Analog-to-Digital Converter (ADC) modules
  - — Five total analog inputs
- **56F807**
  - — Two dual Analog-to-Digital Converter (ADC) modules—four inputs on each
  - — 16 total analog inputs
- 12-bit range
- Monotonic over entire range with no missing codes
- First channel on each 56F80x ADC can be swapped with the alternate ADC
- Can perform two simultaneous Analog-to-Digital Conversions
- Conversion time = 1.25µs
- Contains programmable zero offset register
- Generates interrupt on completion of conversion
- Optional conversion interrupt is asserted when the analog voltage level exceeds, or falls below, the value contained in the zero offset register
- Output is in twos complement or unsigned formats

1.14.10 ADC and PWM Synchronization Feature

- Implemented with an instantiation of the general-purpose timer

- Synchronizes the Analog-to-Digital Converter (ADC) modules to the PWM module
- Additional outputs to synchronize external events to the Pulse Width Modulator (PWM) module

1.14.11 Serial Communications Interface (SCI)

- **56F801, 56F802,** and **56F803**
  - — One Serial Communication Interface (SCI0)
- **56F805** and **56F807**
  - — Two Serial Communication Interfaces (SCI0 & SCI1)
- Asynchronous operation
- Baud rate generation
- IR interface support

1.14.12 Controller Area Network (CAN) Module

- **56F801** and **56F802**
  - — No CAN module available
- **56F803, 56F805,** and **56F807**
  - — One CAN module available
- Module board scalable CAN_12 implementation of Bosch CAN 2.0B protocol
- Double-buffered receiver and triple-buffered transmitter
- *Local priority* concept for transmit buffers
- Two programmable 4- × 8-bit ID filters with mask
- Programmable wake-up
- Low power Sleep mode
- Programmable bit rate up to 1Mbit per second

1.14.13 Peripheral Interrupts

The peripherals on the 56F80x use the interrupt channels found on the DSP56800 core. Each peripheral has its own interrupt vector, often more than one interrupt vector for each peripheral, with the capacity to selectively enable or disable via the IPR found on the DSP56800 core. **Chapter 4, "Interrupt Controller (ITCN)"** provides more details on interrupt vectors.

Table 1-4. Document Revision History for Chapter 1

Version History	Description of Change
Rev. 8	Formatting, layout, spelling, and grammar corrections. Added revision history table.

Chapter 2
Pin Descriptions

2.1 Introduction

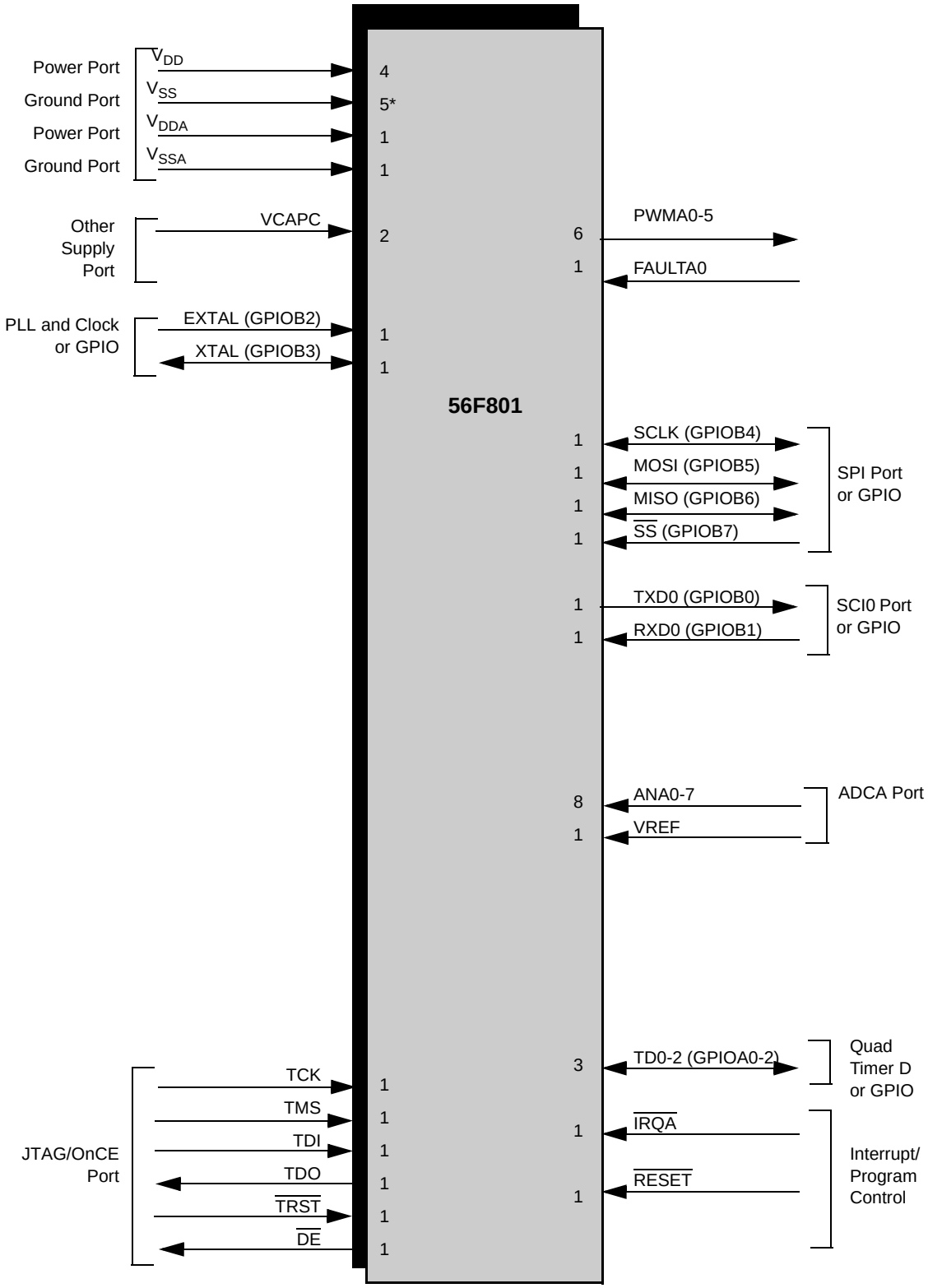
Chapter Two details the input, output signals and functions of the 56F800 core-based family of digital signal processors. Depending on the device, its pins allow clock signals, locking, address interrupt signals, analog-to-digital signals, and more. Each is detailed in this chapter.

The input and output signals of the 56F801/803/805/807 devices are organized into functional groups explained in **Table 2-1.** Those groups are referenced in the tables and figures throughout the chapter. **Table 2-2** through **Table 2-18** describe the signal or signals present on a pin. After reset, all pins are by default the primary function. Any alternate functionality must be programmed. Alternate functions are shown in parentheses.

Table 2-1. Functional Group Pin Allocations

Functional Group	# of Pins 801	# of Pins 802	# of Pins 803	# of Pins 805	# of Pins 807	Detailed Description
Power (V_{DD}, V_{DDA} or V_{DD} Core)	5	3	7	9	11	Table 2-2
Ground (V_{SS} or V_{SSA})	6	4	7	9	13	Table 2-3
Supply Capacitors and VPP	2	2	2	3	4	Table 2-4
PLL and Clock	2	0	3	3	3	Table 2-5
Address Bus[1]	0	0	16	16	16	Table 2-6
Data Bus	0	0	16	16	16	Table 2-7
Bus Control	0	0	4	4	4	Table 2-8
Interrupt and Program Control Signals	2	1	4	5	5	Table 2-9
Dedicated Multi Purpose Input/Output	0	0	0	14	14	Table 2-10
Pulse Width Modulator (PWM) Ports	7	7	12	26	26	Table 2-11
Serial Peripheral Interface (SPI) Port[1]	4	0	4	4	4	Table 2-12
Quadrature Decoder Ports[2]	0	0	4	8	8	Table 2-13
Serial Communications Interface (SCI) Ports[1]	2	2	2	4	4	Table 2-14
CAN Ports	0	0	2	2	2	Table 2-15
Analog-to-Digital Converter (ADC) Ports	9	6	9	9	18	Table 2-16
Timer Module Ports	3	2	2	6	6	Table 2-17
JTAG/On-Chip Emulation (OnCE)	6	5	6	6	6	Table 2-18

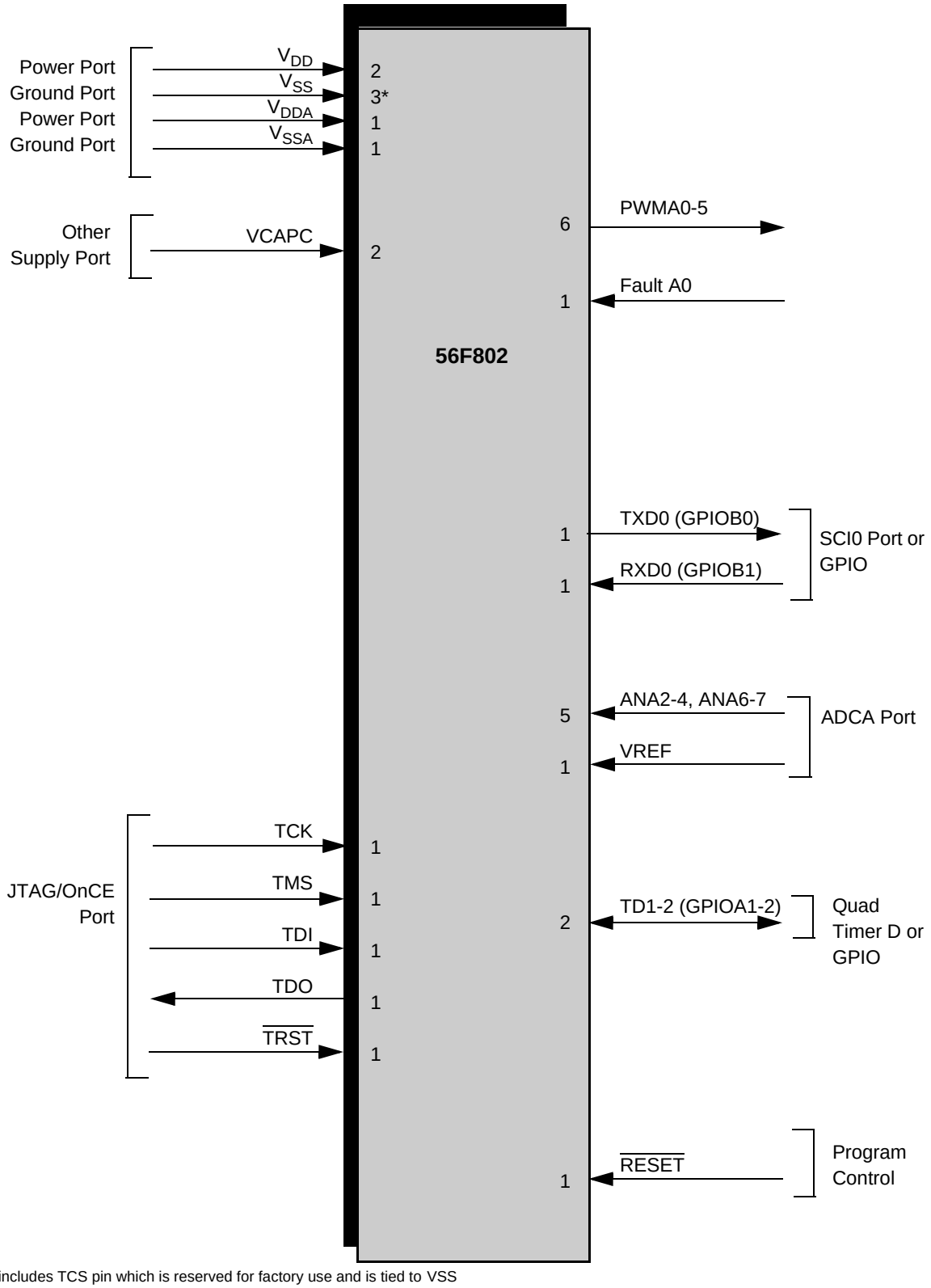
1. Alternately, GPIO pins
2. Alternately, quad timer pins



*includes TCS pin which is reserved for factory use and is tied to VSS

Figure 2-1. 56F801 Signals Identified by Functional Group[1]

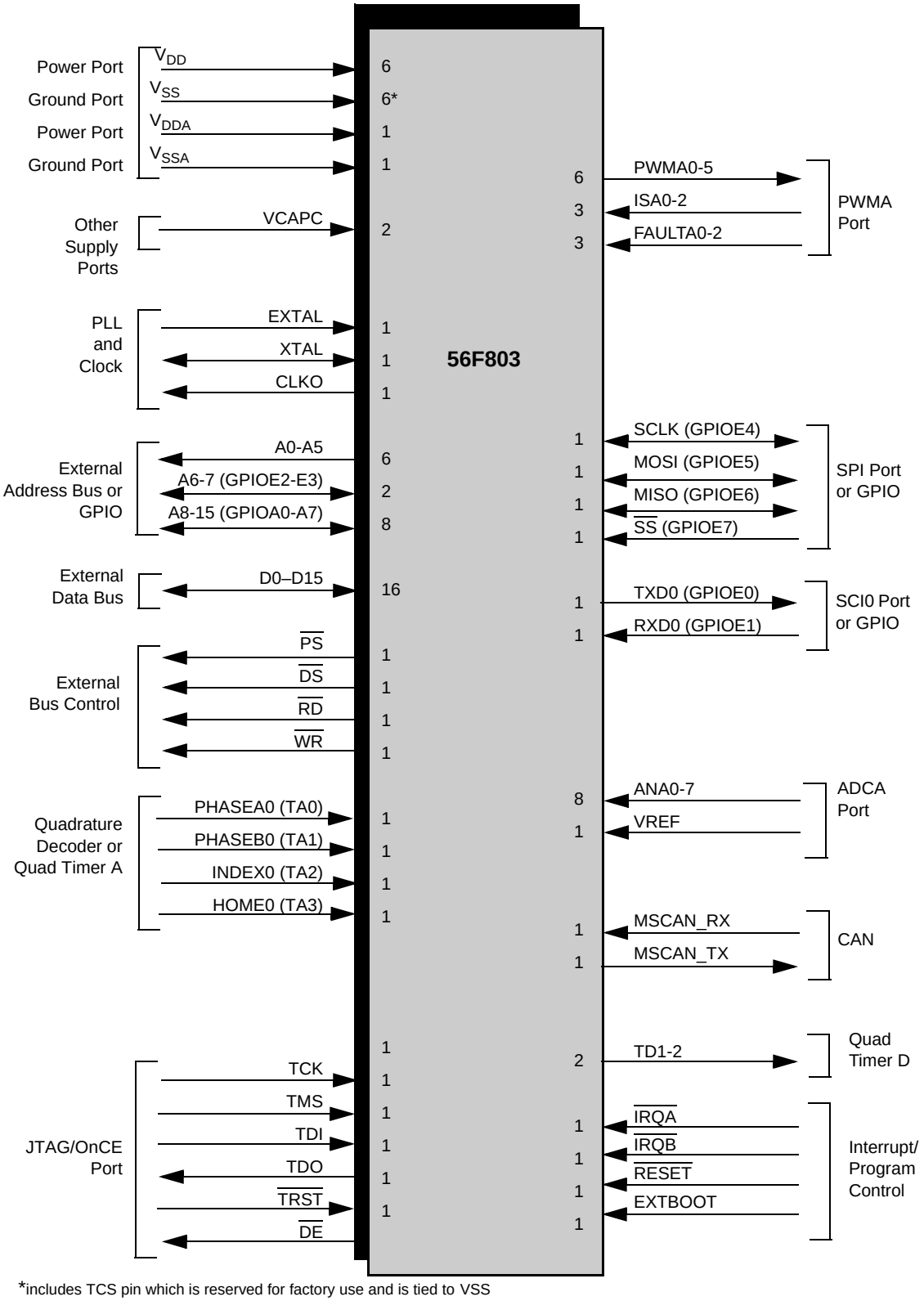
1. Alternate pin functionality is shown in parenthesis.



*includes TCS pin which is reserved for factory use and is tied to VSS

Figure 2-2. 56F802 Signals Identified by Functional Group[1]

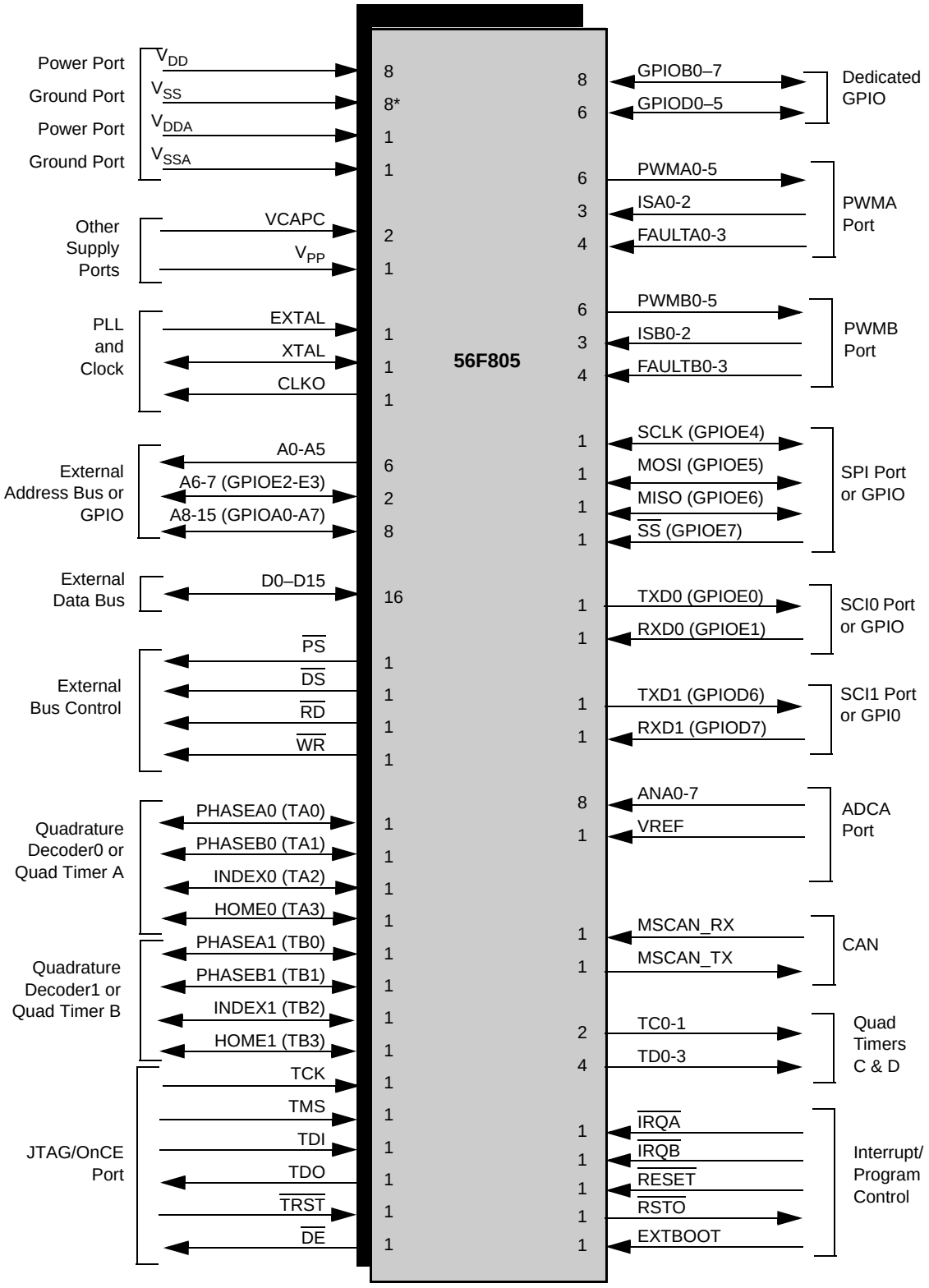
1. Alternate pin functionality is shown in parenthesis.



*includes TCS pin which is reserved for factory use and is tied to VSS

Figure 2-3. 56F803 Signals Identified by Functional Group[1]

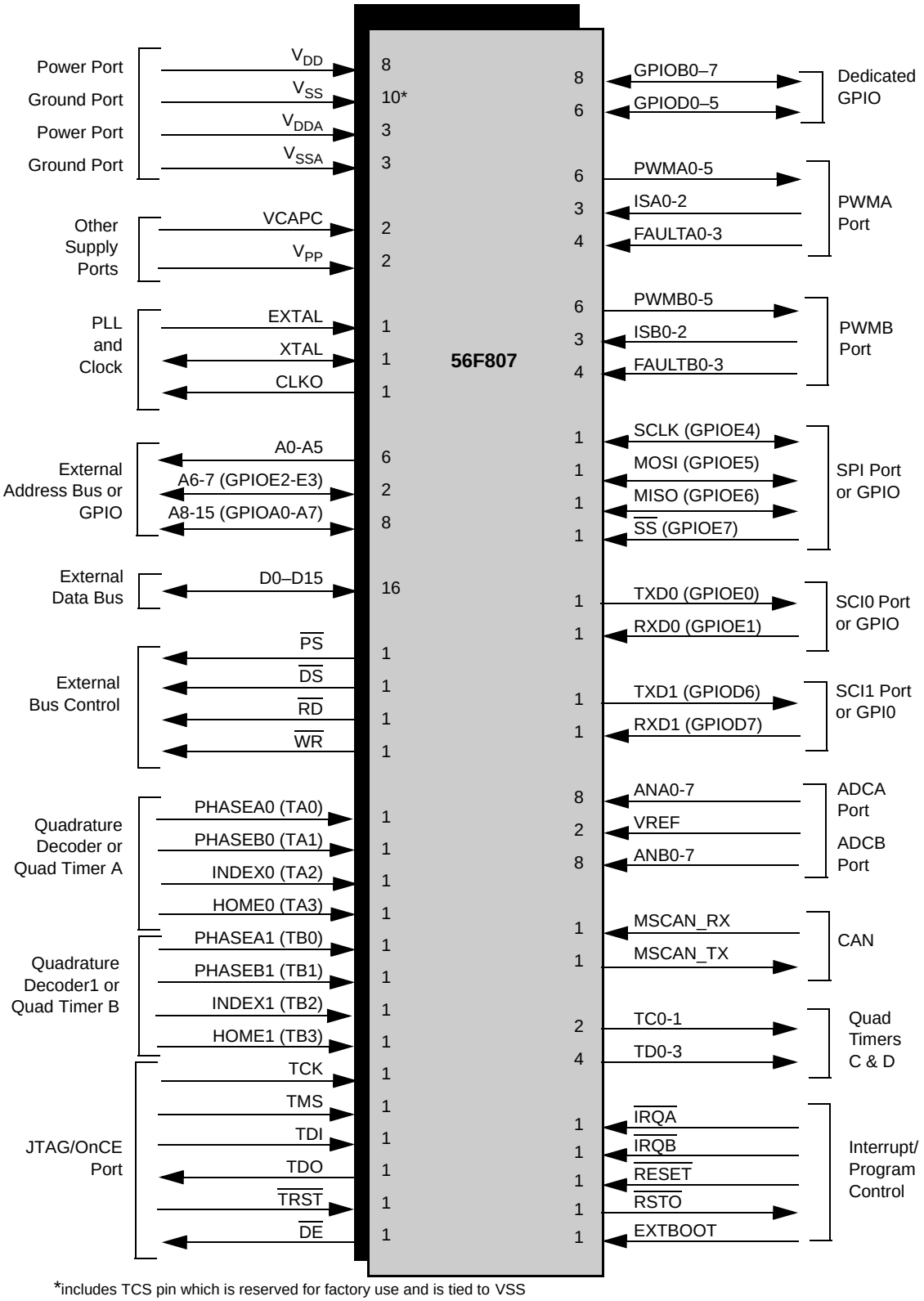
1. Alternate pin functionality is shown in parenthesis.



*includes TCS pin which is reserved for factory use and is tied to VSS

Figure 2-4. 56F805 Signals Identified by Functional Group[1]

1. Alternate pin functionality is shown in parenthesis.



*includes TCS pin which is reserved for factory use and is tied to VSS

Figure 2-5. 56F807 Signals Identified by Functional Group[1]

1. Alternate pin functionality is shown in parenthesis.

2.2 Power and Ground Signals

The following tables show power, ground, and bypass capacitor pinout data. Figure 2-6 through Figure 2-10 show the physical layout of various grounds and voltage input signals associated with the input/output ring.

Table 2-2. Power Inputs

801 Pins	802 Pins	803 Pins	805 Pins	807 Pins	Signal Name	Signal Description
4	2	6	8	8	V_{DD}	**Power**—These pins provide power to the internal structures of the chip, and should all be attached to V_{DD}.
1	1	1	1	3	V_{DDA}	**Analog Power**—These pins are dedicated power pins for the analog portion of the chip and should be connected to a low noise 3.3V supply.

Table 2-3. Grounds

801 Pins	802 Pins	803 Pins	805 Pins	807 Pins	Signal Name	Signal Description
4	2	5	7	9	V_{SS}	**GND**—These pins provide grounding for the internal structures of the chip and should all be attached to V_{SS}.
1	1	1	1	3	V_{SSA}	**Analog Ground**—This pin supplies an analog ground.
1	1	1	1	1	**TCS**	**TCS**—This Schmitt pin is reserved for factory use and must be tied to V_{SS} for normal use. In block diagrams, this pin is considered an additional V_{SS}.

Table 2-4. Supply Capacitors and VPP

801 Pins	802 Pins	803 Pins	805 Pins	807 Pins	Signal Name	Signal Type	State During Reset	Signal Description
2	2	2	2	2	**VCAPC**	Supply	Supply	**VCAPC**—Connect each pin to a 2.2µF or greater bypass capacitor in order to bypass the core logic voltage regulator, required for proper chip operation.
N/A	N/A	N/A	1	2	**VPP**	Input	Input	**VPP**—This pin should be left unconnected as an open circuit for normal functionality.

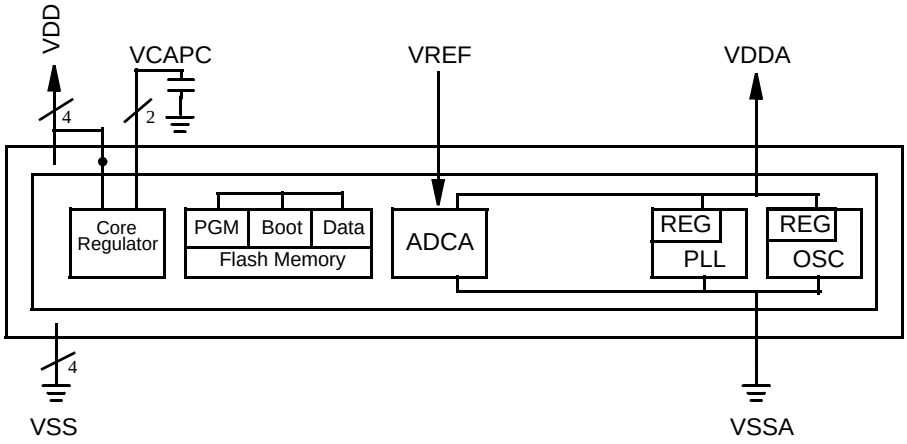


Figure 2-6. 56F801 Power and Ground Pins

Notes:

- VPP is not bonded out.
- Core regulator is internally shorted to one of the VDD pins.
- Flash, RAM and internal logic are powered from the core regulator output.
- Internally, core regulator and VDD share common ground. All must be connected.

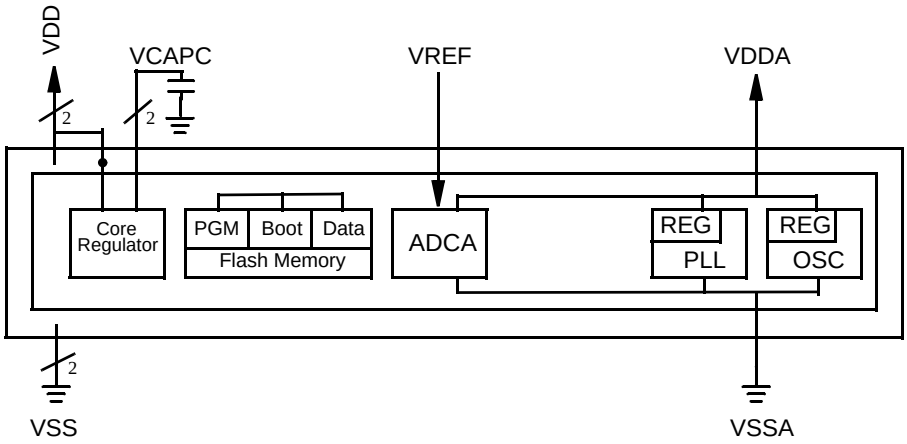


Figure 2-7. 56F802 Power and Ground Pins

Notes:

- VPP is not bonded out.
- Core regulator is internally shorted to one of the VDD pins.
- Flash, RAM and internal logic are powered from the core regulator output.
- Internally, core regulator and VDD share common ground. All must be connected.

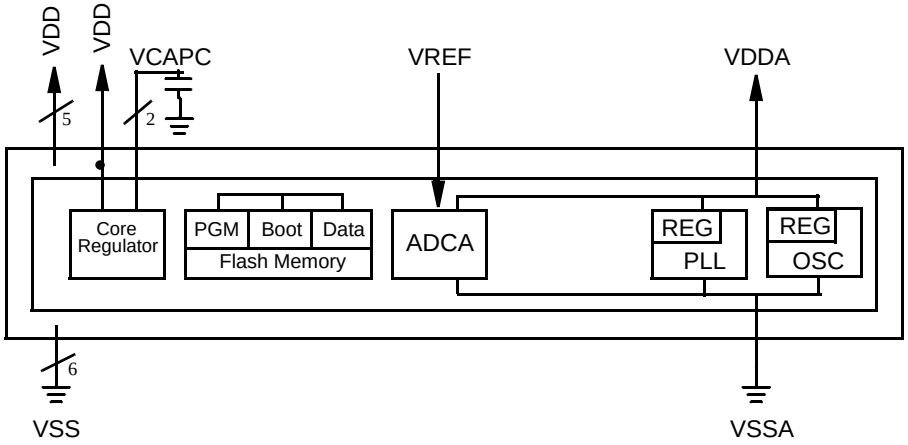


Figure 2-8. 56F803 Power and Ground Pins

Notes:

- VPP is not bonded out.
- Flash, RAM and internal logic are powered from the core regulator output.
- Internally, core regulator and VDD share common ground. All must be connected.

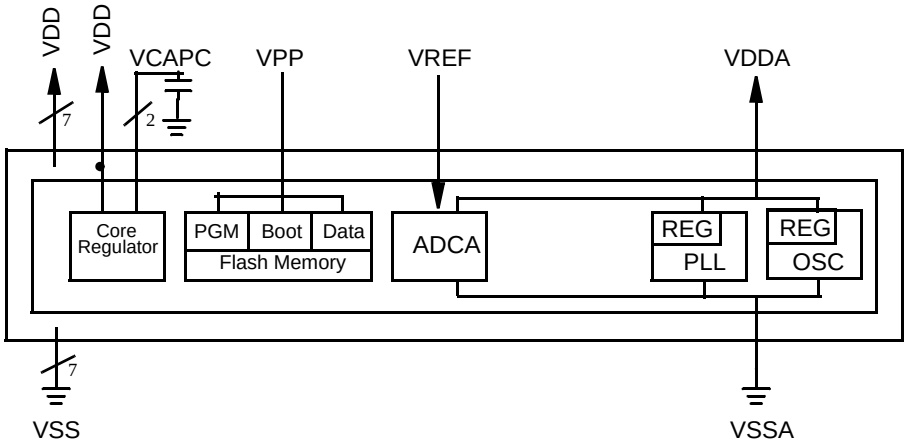


Figure 2-9. 56F805 Power and Ground Pins

Notes:

- VPP is not connected in the customer system.
- Flash, RAM and internal logic are powered from the core regulator output.
- Internally, core regulator and VDD share common ground. All must be connected.

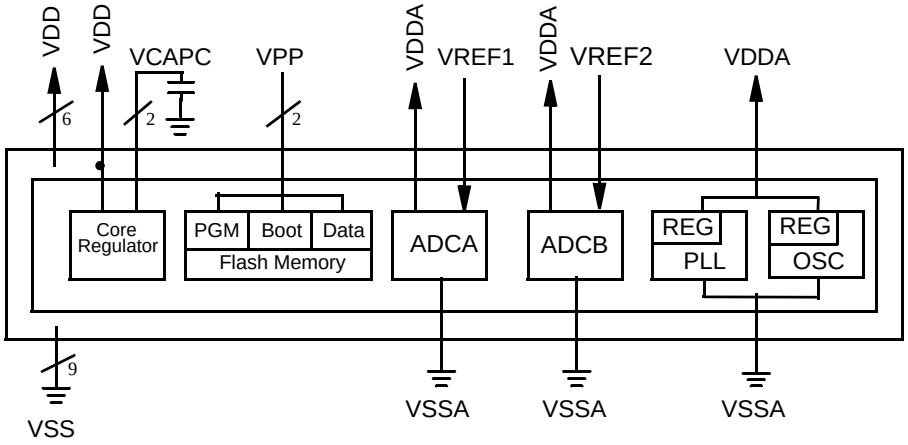


Figure 2-10. 56F807 Power and Ground Pins

Notes:

- Flash, RAM and internal logic are powered from the core regulator output.
- VPP is not connected in the customer system.
- Internally, core regulator and VDD share common ground. All must be connected.

2.3 Clock and Phase Lock Loop Signals

Table 2-5. PLL and Clock Signals

801 Pins	802 Pins	803 Pins	805 Pins	807 Pins	Signal Name	Signal Type	State During Reset	Signal Description
N/A	N/A	1	1	1	**EXTAL**	Input	Input	**External Crystal Oscillator Input**—This input should be connected to an 8MHz external crystal or ceramic resonator. If an 8MHz or less external clock source is used, EXTAL can be used as the input and XTAL must not be connected. For more information, please refer to **Section 15.3.2.2, "External Clock Source".** The input clock can be selected to provide the clock directly to the core. This input clock can also be selected as input clock for the on-chip PLL.
1	N/A	N/A	N/A	N/A	**EXTAL** **(GPIOB2)** Default state	Input Input/ Output	Input Input	**External Crystal Oscillator Input**—This input should be connected to an 8MHz external crystal or ceramic resonator. For more information, please refer to Section 15.3.2.2, "External Clock Source". **Port B GPIO**—This multiplexed pin is a General Purpose I/O (GPIO) pin that can be programmed as an input or output pin. This I/O can be utilized when using the on-chip relaxation oscillator so the EXTAL pin is not needed.
N/A	N/A	1	1	1	**XTAL**	Input/ Output	Chip-driven	**Crystal Oscillator Output**—This output connects the internal crystal oscillator output to an external crystal or ceramic resonator. If an external clock over 8MHz is used, XTAL must be used as the input and EXTAL connected to GND. For more information, please refer to **Section 15.3.2.1, "Crystal Oscillator".**
1	N/A	N/A	N/A	N/A	**XTAL** **(GPIOB3)** Default state	Output Input/ Output	Chip-driven Input	**Crystal Oscillator Output**—This output should be connected to an 8MHz external crystal or ceramic resonator. This pin can also be connected to an external clock source. For more information, please refer to **Section 15.3.2.1, "Crystal Oscillator".** **Port B GPIO**—This multiplexed pin is a General Purpose I/O (GPIO) pin can be programmed as an input or output pin. This I/O can be utilized when using the on-chip relaxation oscillator so the XTAL pin is not needed.

Table 2-5. PLL and Clock Signals (Continued)

801 Pins	802 Pins	803 Pins	805 Pins	807 Pins	Signal Name	Signal Type	State During Reset	Signal Description
N/A	N/A	1	1	1	**CLKO**	Output	Chip-driven	**Clock Output**—This pin outputs a buffered clock signal. By programming the CLKOSEL[4:0] bits in the CLKO Select Register (CLKOSR), the user can select between outputting a version of the signal applied to XTAL and a version of the device's master clock at the output of the PLL. The clock frequency on this pin can also be disabled by programming the CLKOSEL[4:0] bits in CLKOSR.

2.4 Address, Data, and Bus Control Signals

Table 2-6. Address Bus Signals

801 Pins	802 Pins	803 Pins	805 Pins	807 Pins	Signal Name	Signal Type	State During Reset	Signal Description
N/A	N/A	6	6	6	**A0–A5**	Output	Tri-stated	**Address Bus**—A0–A5 specify the address for external Program or Data memory accesses.
N/A	N/A	2	2	2	**A6–A7** **(GPIOE2–GPIOE3)**	Output Input/ Output	Tri-stated Input	**Address Bus**—A6–A7 specify the address for external Program or Data memory accesses. **Port E GPIO**—These two General Purpose I/O (GPIO) pins can be individually programmed as input or output pins. After reset, the default state is address bus.
N/A	N/A	8	8	8	**A8–A15** **(GPIOA0 – GPIOA7)**	Output Input/ Output	Tri-stated Input	**Address Bus**—A8–A15 specify the address for external Program or Data memory accesses. **Port A GPIO**—These eight General Purpose I/O (GPIO) pins can be individually programmed as input or output pins. After reset, the default state is Address Bus.

Table 2-7. Data Bus Signals

801 Pins	802 Pins	803 Pins	805 Pins	807 Pins	Signal Name	Signal Type	State During Reset	Signal Description
N/A	N/A	16	16	16	**D0–D15**	Input/ Output	Tri-stated	**Data Bus**—D0–D15 specify the data for external program or data memory accesses. D0–D15 are tri-stated when the external bus is inactive. Internal pull-ups may be active.

Table 2-8. Bus Control Signals

801 Pins	802 Pins	803 Pins	805 Pins	807 Pins	Signal Name	Signal Type	State During Reset	Signal Description
N/A	N/A	1	1	1	**$\overline{\text{PS}}$**	Output	Tri-stated	**Program Memory Select**—$\overline{\text{PS}}$ is asserted low for external Program memory access.
N/A	N/A	1	1	1	**$\overline{\text{DS}}$**	Output	Tri-stated	**Data Memory Select**—$\overline{\text{DS}}$ is asserted low for external Data memory access.
N/A	N/A	1	1	1	**$\overline{\text{WR}}$**	Output	Tri-stated	**Write Enable**—$\overline{\text{WR}}$ is asserted during external memory write cycles. When $\overline{\text{WR}}$ is asserted low, pins D0–D15 become outputs and the device puts data on the bus. When $\overline{\text{WR}}$ is deasserted high, the external data is latched inside the external device. When $\overline{\text{WR}}$ is asserted, it qualifies the A0–A15, $\overline{\text{PS}}$, and $\overline{\text{DS}}$ pins. $\overline{\text{WR}}$ can be connected directly to the $\overline{\text{WE}}$ pin of a Static RAM.
N/A	N/A	1	1	1	**$\overline{\text{RD}}$**	Output	Tri-stated	**Read Enable**—$\overline{\text{RD}}$ is asserted during external memory read cycles. When $\overline{\text{RD}}$ is asserted low, pins D0–D15 become inputs and an external device is enabled onto the data bus. When $\overline{\text{RD}}$ is deasserted high, the external data is latched inside the DSC. When $\overline{\text{RD}}$ is asserted, it qualifies the A0–A15, $\overline{\text{PS}}$, and $\overline{\text{DS}}$ pins. $\overline{\text{RD}}$ can be connected directly to the $\overline{\text{OE}}$ pin of a static RAM or ROM.

2.5 Interrupt and Program Control Signals

Table 2-9. Interrupt and Program Control Signals

801 Pins	802 Pins	803 Pins	805 Pins	807 Pins	Signal Name	Signal Type	State During Reset	Signal Description
1	N/A	1	1	1	**$\overline{\text{IRQA}}$**	Input (Schmitt)	Input	**External Interrupt Request A**—The $\overline{\text{IRQA}}$ input is a synchronized external interrupt request that indicates that an external device is requesting service. It can be programmed to be level-sensitive or negative-edge-triggered.

Table 2-9. Interrupt and Program Control Signals (Continued)

801 Pins	802 Pins	803 Pins	805 Pins	807 Pins	Signal Name	Signal Type	State During Reset	Signal Description
N/A	N/A	1	1	1	**$\overline{\text{IRQB}}$**	Input (Schmitt)	Input	**External Interrupt Request B**—The $\overline{\text{IRQB}}$ input is an external interrupt request that indicates that an external device is requesting service. It can be programmed to be level-sensitive or negative-edge-triggered.
1	1	1	1	1	**$\overline{\text{RESET}}$**	Input (Schmitt)	Input	**Reset**—This input is a direct hardware reset on the processor. When $\overline{\text{RESET}}$ is asserted low, the DSC is initialized and placed in the Reset state. A Schmitt trigger input is used for noise immunity. When the $\overline{\text{RESET}}$ pin is deasserted, the initial chip operating mode is latched from the EXTBOOT pin. The internal reset signal will be deasserted synchronously with the internal clocks after a fixed number of internal clocks. To ensure complete hardware reset, $\overline{\text{RESET}}$ and $\overline{\text{TRST}}$ should be asserted together. The only exception occurs in a debugging environment when a hardware device reset is required and it is necessary not to reset the JTAG/OnCE module. In this case, assert $\overline{\text{RESET}}$, but do not assert $\overline{\text{TRST}}$.
N/A	N/A	N/A	1	1	**$\overline{\text{RSTO}}$**	Output	Output	**Reset Output**—This output reflects the internal reset state of the chip.
N/A	N/A	1	1	1	**EXTBOOT**	Input (Schmitt)	Input	**External Boot**—This input is tied to V_{DD} to force the device to boot from off-chip memory. Otherwise, it is tied to V_{SS}.

2.6 GPIO Signals

Table 2-10. Dedicated General Purpose Input/Output (GPIO) Signals

801 Pins	802 Pins	803 Pins	805 Pins	807 Pins	Signal Name	Signal Type	State During Reset	Signal Description
N/A	N/A	N/A	8	8	**GPIOB0–GPIOB7**	Input/ Output	Input	**Port B GPIO**—These eight dedicated General Purpose I/O (GPIO) pins can be individually programmed as input or output pins. After reset, the default state is GPIO input.

Table 2-10. Dedicated General Purpose Input/Output (GPIO) Signals (Continued)

801 Pins	802 Pins	803 Pins	805 Pins	807 Pins	Signal Name	Signal Type	State During Reset	Signal Description
N/A	N/A	N/A	6	6	**GPIOD0–GPIOD5**	Input/ Output	Input	**Port D GPIO**—These six dedicated General Purpose I/O (GPIO) pins can be individually programmed as input or output pins. After reset, the default state is GPIO input.

2.7 Pulse Width Modulator (PWM) Signals

Table 2-11. Pulse Width Modulator (PWMA and PWMB) Signals

801 Pins	802 Pins	803 Pins	805 Pins	807 Pins	Signal Name	Signal Type	State During Reset	Signal Description
6	6	6	6	6	**PWMA0–5**	Output	Tri-stated	**PWMA0–5**—These are six PWMA output pins.
N/A	N/A	3	3	3	**ISA0–2**	Input (Schmitt)	Input	**ISA0–2**—These three input current status pins are used for top/bottom pulse width correction in complementary channel operation for PWMA.
1	1	1	1	1	**FAULTA0**	Input (Schmitt)	Input	**FAULTA0**—This fault input pin is used for disabling selected PWMA outputs in cases where fault conditions originate off-chip.
N/A	N/A	2	2	2	**FAULTA1–2**	Input (Schmitt)	Input	**FAULTA1-2**—These two fault input pins are used for disabling selected PWMA outputs in cases where fault conditions originate off-chip.
N/A	N/A	N/A	1	1	**FAULTA3**	Input (Schmitt)	Input	**FAULTA3**—This fault input pin is used for disabling selected PWM outputs in cases where fault conditions originate off-chip.
N/A	N/A	N/A	6	6	**PWMB0–5**	Output	Tri-stated	**PWMB0–5**—Six PWMB output pins.
N/A	N/A	N/A	3	3	**ISB0–2**	Input (Schmitt)	Input	**ISB0–2**—These three input current status pins are used for top/bottom pulse width correction in complementary channel operation for PWMB.
N/A	N/A	N/A	4	4	**FAULTB0–3**	Input (Schmitt)	Input	**FAULTB0–3**—These four fault input pins are used for disabling selected PWMB outputs in cases where fault conditions originate off-chip.

2.8 Serial Peripheral Interface (SPI) Signals

Table 2-12. Serial Peripheral Interface (SPI) Signals

801 Pins	802 Pins	803 Pins	805 Pins	807 Pins	Signal Name	Signal Type	State During Reset	Signal Description
1	N/A	1	1	1	**MISO**	Input/ Output	Input	**SPI Master In/Slave Out (MISO)**—This serial data pin is an input to a master device and an output from a slave device. The MISO line of a slave device is placed in the high-impedance state if the slave device is not selected. **Port E/B GPIO**—This General Purpose I/O (GPIO) pin can be individually programmed as an input or output pin. After reset, the default state is MISO.
					(GPIOE6) **GPIOB6 for 801**	Input/ Output	Input	
1	N/A	1	1	1	**MOSI**	Input/ Output	Input	**SPI Master Out/Slave In (MOSI)**—This serial data pin is an output from a master device and an input to a slave device. The master device places data on the MOSI line a half-cycle before the clock edge the slave device uses to latch the data. **Port E/B GPIO**—This General Purpose I/O (GPIO) pin can be individually programmed as an input or output pin. After reset, the default state is MOSI.
					(GPIOE5) **GPIOB5 for 801**	Input/ Output	Input	
1	N/A	1	1	1	**SCLK**	Input/ Output	Input	**SPI Serial Clock**—In master mode, this pin serves as an output, clocking slaved listeners. In slave mode, this pin serves as the data clock input. **Port E/B GPIO**—This General Purpose I/O (GPIO) pin can be individually programmed as an input or output pin. After reset, the default state is SCLK.
					(GPIOE4) **GPIOB4 for 801**	Input/ Output	Input	
1	N/A	1	1	1	**$\overline{\text{SS}}$**	Input	Input	**SPI Slave Select**—In master mode, this pin is used to arbitrate multiple masters. In slave mode, this pin is used to select the slave. **Port E/B GPIO**—This General Purpose I/O (GPIO) pin can be individually programmed as an input or output pin. After reset, the default state is $\overline{\text{SS}}$.
					(GPIOE7) **GPIOB7 for 801**	Input/ Output	Input	

2.9 Quadrature Decoder Signals

Table 2-13. Quadrature Decoder (Quad Dec0 and Quad Dec1) Signals

801 Pins	802 Pins	803 Pins	805 Pins	807 Pins	Signal Name	Signal Type	State During Reset	Signal Description
N/A	N/A	1	1	1	**PHASEA0**	Input	Input	**Phase A**—Quadrature Decoder #0 PHASEA input
					(TA0)	Input/ Output	Input	**TA0**—Timer A Channel 0
N/A	N/A	1	1	1	**PHASEB0**	Input	Input	**Phase B**— Quadrature Decoder #0 PHASEB input
					(TA1)	Input/ Output	Input	**TA1**—Timer A Channel 1
N/A	N/A	1	1	1	**INDEX0**	Input	Input	**Index**—Quadrature Decoder #0 INDEX input
					(TA2)	Input/ Output	Input	**TA2**—Timer A Channel 2
N/A	N/A	1	1	1	**HOME0**	Input	Input	**Home**—Quadrature Decoder #0 HOME input
					(TA3)	Input/ Output	Input	**TA3**—Timer A Channel 3
N/A	N/A	N/A	1	1	**PHASEA1**	Input	Input	**Phase A**—Quadrature Decoder #1 PHASEA input
					(TB0)	Input/ Output	Input	**TB0**—Timer B Channel 0
N/A	N/A	N/A	1	1	**PHASEB1**	Input	Input	**Phase B**—Quadrature Decoder #1 PHASEB input
					(TB1)	Input/ Output	Input	**TB1**—Timer B Channel 1
N/A	N/A	N/A	1	1	**INDEX1**	Input	Input	**Index**—Quadrature Decoder #1 INDEX input
					(TB2)	Input/ Output	Input	**TB2**—Timer B Channel 2
N/A	N/A	N/A	1	1	**HOME1**	Input	Input	**Home**—Quadrature Decoder #1 HOME input
					(TB3)	Input/ Output	Input	**TB3**—Timer B Channel 3

2.10 Serial Communications Interface (SCI) Signals

Table 2-14. Serial Communications Interface (SCI0 and SCI1) Signals

801 Pins	802 Pins	803 Pins	805 Pins	807 Pins	Signal Name	Signal Type	State During Reset	Signal Description
1	1	1	1	1	**TXD0** **(GPIOE0)** **GPIOB0 for 801 & 802**	Output Input/ Output	Input Input	**Transmit Data**—SCI0 transmit data output **Port E/B GPIO**—This General Purpose I/O (GPIO) pin can be individually programmed as an input or output pin. After reset, the default state is SCI output.
1	1	1	1	1	**RXD0** **(GPIOE1)** **GPIOB1 for 801 & 802**	Input Input/ Output	Input Input	**Receive Data**—SCI0 receive data input **Port E/B GPIO**—This General Purpose I/O (GPIO) pin can be individually programmed as an input or output pin. After reset, the default state is SCI input.
N/A	N/A	N/A	1	1	**TXD1** **(GPIOD6)**	Output Input/ Output	Input Input	**Transmit Data**—SCI1 transmit data output **Port D GPIO**—This General Purpose I/O (GPIO) pin can be individually programmed as an input or output pin. After reset, the default state is SCI output.
N/A	N/A	N/A	1	1	**RXD1** **(GPIOD7)**	Input Input/ Output	Input Input	**Receive Data**—SCI1 receive data input **Port D GPIO**—This General Purpose I/O (GPIO) pin can be individually programmed as an input or output pin. After reset, the default state is SCI input.

2.11 CAN Signals

Table 2-15. CAN Module Signals

801 Pins	802 Pins	803 Pins	805 Pins	807 Pins	Signal Name	Signal Type	State During Reset	Signal Description
N/A	N/A	1	1	1	**MSCAN_ RX**	Input (Schmitt)	Input	**CAN Receive Data**—This is the CAN input. This pin has an internal pull up resistor.
N/A	N/A	1	1	1	**MSCAN_ TX**	Output	Output	**CAN Transmit Data**—CAN output. CAN output is an open-drain output and a pull-up resistor is needed.

2.12 Analog-to-Digital Converter (ADC) Signals

Table 2-16. Analog-to-Digital Converter (ADCA and ADCB) Signals

801 Pins	802 Pins	803 Pins	805 Pins	807 Pins	Signal Name	Signal Type	State During Reset	Signal Description
4	N/A	4	4	4	**ANA0–3**	Input	Input	**ANA0–3**—Analog inputs to ADCA channel 1
4	N/A	4	4	4	**ANA4–7**	Input	Input	**ANA4–7**—Analog inputs to ADCA channel 2
1	1	1	1	2	**VREF**	Input	Input	**VREF**—Analog reference voltage for ADC. Must be set to V_{DDA}-0.3V for optimal performance.
N/A	N/A	N/A	N/A	4	**ANB0–3**	Input	Input	**ANB0–3**—Analog inputs to ADCB, channel 1
N/A	N/A	N/A	N/A	4	**ANB4–7**	Input	Input	**ANB4–7**—Analog inputs to ADCB channel 2
N/A	3	N/A	N/A	N/A	**ANA2–4**	Input	Input	**ANA2–4—**Analog inputs to ADCA channel 1
N/A	2	N/A	N/A	N/A	**ANA6–7**	Input	Input	**ANA6–7—**Analog inputs to ADCA channel 2

2.13 Quad Timer Module Signals

Table 2-17. Quad Timer Module Signals

801 Pins	802 Pins	803 Pins	805 Pins	807 Pins	Signal Name	Signal Type	State During Reset	Signal Description
N/A	N/A	N/A	2	2	**TC0-1**	Input/ Output	Input	**TC0-1**—Timer C Channels 0 and 1
1	N/A	N/A	1	1	**TD0**	Input/ Output	Input	**TD0**—Timer D Channel 0
					(GPIOA0) for 801 only	Input/ Output	Input	**Port A GPIO**—This General Purpose I/O (GPIO) pin can be individually programmed as an input or output pin. After reset, the default state is the quad timer input.

Table 2-17. Quad Timer Module Signals (Continued)

801 Pins	802 Pins	803 Pins	805 Pins	807 Pins	Signal Name	Signal Type	State During Reset	Signal Description
2	2	2	2	2	**TD1–2** **(GPIOA1-GPIOA2) for 801 and 802 only**	Input/ Output Input/ Output	Input Input	**TD1–2**—Timer D Channel 1–2 **Port A GPIO**—These General Purpose I/O (GPIO) pins can be individually programmed as input or output pins. After reset, the default state is the quad timer input.
N/A	N/A	N/A	1	1	**TD3**	Input/ Output	Input	**TD3**—Timer D Channel 3

2.14 JTAG/OnCE

Table 2-18. JTAG/On-Chip Emulation (OnCE) Signals

801 Pins	802 Pins	803 Pins	805 Pins	807 Pins	Signal Name	Signal Type	State During Reset	Signal Description
1	1	1	1	1	**TCK**	Input (Schmitt)	Input, pulled low internally	**Test Clock Input**—This input pin provides a gated clock to synchronize the test logic and shift serial data to the JTAG/OnCE port. The pin is connected internally to a pull-down resistor.
1	1	1	1	1	**TMS**	Input (Schmitt)	Input, pulled high internally	**Test Mode Select Input**—This input pin is used to sequence the JTAG TAP controller's state machine. It is sampled on the rising edge of TCK and has an on-chip pull-up resistor. **Note:** Always tie the TMS pin to V_{DD} through a 2.2K resistor.
1	1	1	1	1	**TDI**	Input (Schmitt)	Input, pulled high internally	**Test Data Input**—This input pin provides a serial input data stream to the JTAG/OnCE port. It is sampled on the rising edge of TCK and has an on-chip pull-up resistor.
1	1	1	1	1	**TDO**	Output	Tri-stated	**Test Data Output**—This tri-statable output pin provides a serial output data stream from the JTAG/OnCE port. It is driven in the shift-IR and shift-DR controller states, and changes on the falling edge of TCK.
1	1	1	1	1	**$\overline{\text{TRST}}$**	Input (Schmitt)	Input, pulled high internally	**Test Reset**—As an input, a low signal on this pin provides a reset signal to the JTAG TAP controller. To ensure complete hardware reset, $\overline{\text{TRST}}$ should be asserted at power-up and whenever $\overline{\text{RESET}}$ is asserted. The only exception occurs in a debugging environment when a hardware device reset is required and it is necessary not to reset the OnCE/JTAG module. In this case, assert $\overline{\text{RESET}}$, but do not assert $\overline{\text{TRST}}$. **Note:** For normal operation, connect $\overline{\text{TRST}}$ directly to V_{SS}. If the design is to be used in a debugging environment, $\overline{\text{TRST}}$ may be tied to V_{SS} through a 1K resistor.
1	N/A	1	1	1	**$\overline{\text{DE}}$**	Output	Output	**Debug Event**—$\overline{\text{DE}}$ provides a low pulse on recognized debug events.

Table 2-19. Document Revision History for Chapter 2

Version History	Description of Change
Rev. 8	Formatting, layout, spelling, and grammar corrections. Added revision history table. Added the following note to the description of the TMS signal in **Table 2-18**: "Always tie the TMS pin to V_{DD} through a 2.2K resistor." Added the following note to the description of the $\overline{\text{TRST}}$ signal in **Table 2-18**: "For normal operation, connect $\overline{\text{TRST}}$ directly to V_{SS}. If the design is to be used in a debugging environment, $\overline{\text{TRST}}$ may be tied to V_{SS} through a 1K resistor."

Chapter 3
Memory and Operating Modes

3.1 Memory Map

The 56F800 family is very adaptable. It is a *complete* controller on a single, high-performance chip with Flash technology and analog-to-digital converters embedded on a single mechanism. This section describes detailed, on-chip memories and the opening modes of the 56F800 core-based chip family. Additionally, interrupt vectors, Interrupt Priority Register (IPR) and peripheral memory maps are also located in this chapter.

3.2 Memory Map Description

The 56F80x chip family uses two independent memory spaces, data and program, using a Harvard architecture. RAM and Flash memory are used for the on-chip data memory and for the on-chip program memory.

Table 3-1. Chip Memory Configurations

On-Chip Memory	56F801	56F802	56F803	56F805	56F807
Program Flash (PFLASH)	8K x 16	8K x 16	32K x 16	32K x 16	64K x 16
Data Flash (XFLASH)	2K x 16	2K x 16	4K x 16	4K x 16	8K x 16
Program RAM (PRAM)	1K x 16	1K x 16	512 x 16	512 x 16	2K x 16
Data RAM (XRAM)	1K x 16	1K x 16	2K x 16	2K x 16	4K x 16
Program Boot Flash	2K x 16	2K x 16	2K x 16	2K x 16	2K x 16

On-chip memory sizes for each of the parts are summarized in **Table 3-1.** Both the program and data memories can be expanded off-chip. The program memory map is located in **Table 3-2.** The operating mode control bits (MA and MB) in the Operating Mode Register (OMR), coupled with the $\overline{\text{BOOTMAP}}$ bit in the SYS_CNTL register, see **Section 16.9.1, "System Control Register (SYS_CNTL),** control the program memory map and select the vector address.

The 56F803, 805, and 807 chips include external memory expansion capabilities up to 64K words. This is not available on the 56F801or 56F802 devices.

Table 3-2. Program Memory Map for 56F80x

Begin/ End Address	Mode 0A[1]				Mode 0B[1]				Mode 3[1]
	801/802	803	805	807	801/802	803	805	807	803, 805, 807
0000 0003	BFlash 4	BFlash 4	BFlash 4	BFlash 4	Not Supported	BFlash 4	B Flash 4	BFlash 4	PExternal 64K
0004 1FFF	PFlash 8K-4	PFlash 31.5K-4	PFlash 31.5K-4	PFlash 32K-4		PFlash 31.5K-4	P.Flash 31.5K-4	PFlash 28K-4	
2000 6FFF	Reserved								
7000 73FF								PRAM 2K	
7400 77FF									
7800 7BFF								BFlash 2K	
7C00 7DFF	PRAM 1K								
7E00 7FFF		PRAM 512	PRAM 512			PRAM 512	PRAM 512		
8000 87FF	BFlash 2K	BFlash 2K	BFlash 2K	PFlash 28K		PExternal 32K	PExternal 32K	PExternal 32K	
8800 EFFF	Reserved	Reserved	Reserved						
F000 F3FF				PRAM 2K					
F400 F7FF									
F800 FFFF				BFlash 2K					

1. See Table 16-1, page 16-12 for an explanation of Boot Modes 0A, 0B, and 3.

P represents Program

B represents Boot

In all modes, except 3, the first four logical addresses are a reflection of the first four physical addresses of Boot Flash. In Mode 0B, this is largely an academic point, as any Reset or COP Reset resets the memory map back to Mode 0A.

Note: Modes one and two are not supported in this group of devices.

Note: For more information about mode three, see **Section 3.7.3, "Mode 3–External".**

Table 3-3. Data Memory Map for 56F80x

Begin/ End Address	EX=0				EX=1 803, 805, 807
	801/802	803	805	807	
0000 03FF	XRAM 1K	XRAM 2K	XRAM 2K	XRAM 4K	XExternal 64K
0400 07FF	Reserved				
0800 0BFF		Reserved	Reserved		
0C00 0FFF	Peripherals	Peripherals	Peripherals		
1000 13FF	XFlash 2K	XFlash 4K	XFlash 4K	Peripherals	
1400 17FF					
1800 1FFF	Reserved			Reserved	
2000 2FFF		XExternal 56K-128	XExternal 56K-128	XFlash 8K	
3000 3FFF					
4000 FF7F				XExternal 48K-128	
FF80 FFFF	Core Regs 128	Core Regs 128	Core Regs 128	Core Regs 128	Core Regs 128

X represents Data

With EX = 1, all 64K address space is external, unless I/O Short Addressing is used to address core register.

If I/O Short Addressing is used, then the Core Registers become available.

Note: Table 3-3 summarizes the data memory map. The External X-memory (EX) control bit in the Operating Mode Register (OMR) controls the data memory map.

Note: Throughout this manual, *data memory locations* are noted as X:$xxxx and *program memory locations* are noted as P:$xxxx, where $ represents a value in hex format.

3.3 Data Memory

The 56F801/802 have 1K words of on-chip data RAM and 2K words of on-chip data Flash. The 56F803 and 56F805 parts have 2K words of on-chip data RAM and 4K words of on-chip data Flash. The 56F807 has 4K words of on-chip data RAM and 8K words of data Flash.

The 64 data memory addressed at the top of the memory map ($FF80 to $FFFF) are reserved for 56800 on-chip core configuration registers. Additionally, there is a 1K word, $0800 to $0BFF, a

non-accessible segment hole in the memory map in the 56F803 and 56F805. The 56F801/802 have a 2K hole while the 56F807 has no hole. No memory location will be selected on attempted accesses to the hole when EX = 0, internal data memory map enabled. When EX = 1, the hole does not exist and may be accessed like all other external data memory.

Note: For 56800 core instructions performing two reads from the data memory in a single instruction, the *second access* using the R3 pointer always occurs to *on-chip memory*, regardless of how the OMR's EX bit is programmed. For example, the second read of the following code sequence accesses on-chip X data memory X:$0:

```
MOVE #$0000,R3
NOP
MOVE X:(R1)+,Y0 X:(R3)+,X0
```

The data memory may be expanded off-chip for the 56F803/805/807 but not for the 56F801or 56F802. When the OMR's EX bit is programmed with a *one*, there are 65,536 addressable off-chip data memory locations.

When the EX bit is set, the on-chip core configuration registers may only be accessed using the I/O short addressing mode.

The external data memory bus access time is controlled by four bits of the Bus Control Register (BCR) located at X:$FFF9. This register is shown in **Figure 3-1.**

The External X bit (EX, bit 3) of the OMR in the 56800 core determines the mapping of the data memory, shown in **Table 3-3.** Setting the EX bit to 1 completely disables the on-chip data memory and enables a full 64K *external* data memory map.

Note: Two exceptions to this rule exist. The first exception is, if a MOVE, TSTW, or BIT FIELD instruction is used with the I/O short addressing mode, the EX bit is ignored. This allows the on-chip core control registers to be accessed when the EX bit is set. When the EX bit is set, the access time to any external data memory is controlled by the BCR. The second exception is for instructions performing two reads from the data memory in a single cycle, the EX bit is ignored during the second access using the R3 pointer.

A complete description of the Operating Mode Register (OMR) is provided in *DSP56800 Family Manual (DSP56800FAM).*

3.3.1 Bus Control Register (BCR)

BCR $FFF9	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	0	DRV	0	WAIT STATE FIELD for EXTERNAL X-MEMORY				WAIT STATE FIELD for EXTERNAL P-MEMORY			
Write																
Reset	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0

Figure 3-1. Bus Control Register (BCR)

See Programmer's Sheet, Appendix B, on page B-19

3.3.1.1 Reserved—Bits 15–10

These bits are reserved and are read as 0 during operations. The bits should be written with 0 ensuring future compatibility.

3.3.1.2 Drive (DRV)—Bit 9

The Drive (DRV) control bit is used to specify what occurs on the external memory port pins when no external access is performed—whether the pins remain driven or are tri-stated. The DRV bit is cleared on hardware reset.

Table 3-4. Port A Operation with DRV Bit = 0

Mode	Pins		
	A0–A15	$\overline{PS}$, $\overline{DS}$, $\overline{RD}$, $\overline{WR}$	D0–D15
Normal Mode, External Access	Driven	Driven	Driven
Normal Mode, Internal Access	Tri-Stated	Tri-Stated	Tri-Stated
Stop Mode	Tri-Stated	Tri-Stated	Tri-Stated
Wait Mode	Tri-Stated	Tri-Stated	Tri-Stated
Reset Mode	Tri-Stated	Pulled High Internally	Tri-Stated

Table 3-5. Port A Operation with DRV Bit = 1

Mode	Pins		
	A0–A15	$\overline{PS}$, $\overline{DS}$, $\overline{RD}$, $\overline{WR}$	D0–D15
Normal Mode, External Access	Driven	Driven	Driven
Normal Mode, Internal Access	Driven	Driven	Tri-Stated
Stop Mode	Driven	Driven	Tri-Stated
Wait Mode	Driven	Driven	Tri-Stated
Reset Mode	Tri-Stated	Pulled High Internally	Tri-Stated

3.3.1.3 Reserved—Bit 8

This bit is reserved or not implemented. It is read as 0 during operations. It should be written with 0 ensuring future compatibility.

3.3.1.4 Wait State Data Memory (WSX[3:0])—Bits 7–4

These bits allow programming of the wait states for external data memory. The bottom two bits, five and four, are hardcoded to zero. The WSX[3:0] bits are programmed as follows:

Table 3-6. Programming WSX[3:0] Bits for Wait States

Bit String	Hex Value	Number of Wait States
0000	$0	0
0100	$4	4
1000	$8	8
1100	$C	12
All Others		Illegal

3.3.1.5 Wait State P Memory (WSP[3:0])—Bits 3–0

These bits allow programming of the wait states for external program memory. The bottom two bits, one and zero, are hardcoded to zero. The WSP[3:0] bits are programmed as follows:

Table 3-7. Programming WSP[3:0] Bits for Wait States

Bit String	Hex Value	Number of Wait States
0000	$0	0
0100	$4	4
1000	$8	8
1100	$C	12
All Others		Illegal

3.3.2 Operating Mode Register (OMR)

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	NL	0	0	0	0	0	0	CC	0	SD	R	SA	EX	0	MB	MA
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3-2. Operating Mode Register (OMR)

See Programmer's Sheet, Appendix B, on page B-20

Note: MA and MB are latched from the EXTBOOT pin on reset.

3.3.2.1 Nested Looping (NL)—Bit 15

The Nested Looping (NL) bit displays the status of program DO looping and the hardware stack. When the NL bit is set, it indicates the program is currently in a nested DO loop, that is, two DO

loops are active. When this bit is cleared, it indicates the program is currently not in a nested DO loop. There may be a single active DO loop or no DO loop active. This bit is necessary for saving and restoring the contents of the hardware stack. REP looping does not affect this bit.

It is important to never put the processor in the reserved combination specified in **Table 3-8.** This can be avoided by ensuring the Looped Flag (LF) bit is never cleared when the NL bit is set. The NL bit is cleared on processor reset.

If both the NL and LF bits are set, meaning two DO loops are active and a DO instruction is executed, a hardware stack overflow interrupt occurs because there is no more space in the HWS to support a third DO loop.

The NL bit is also affected by any accesses to the HWS. Any MOVE instruction writing this register copies the old contents of the LF bit into the NL bit. It sets the NL bit into the LF bit before clearing the NL bit.

Table 3-8. Looping Status

NL (In OMR)	LF (in SR)	DO Loop Status
0	0	No DO Loops Active
0	1	Single DO Loop Active
1	0	Reserved
1	1	Two DO Loops Active

3.3.2.2 Reserved—Bits 14–9

This bit field is reserved or not implemented. It is read and can be written as 0.

3.3.2.3 Condition Codes (CC)—Bit 8

The Condition Code (CC) bit selects whether condition codes are generated using a 36-bit result from the Multiplier/Accumulator (MAC) array or a 32-bit result. When the CC bit is set, the C, N, V, and Z condition codes are generated based on bit 31of the data Arithmetic Logic Unit (data ALU) result. When cleared, the C, N, V, and Z condition codes are generated based on bit 35 of the data ALU result. The generation of the L, E, and U condition codes is not affected by the CC bit. The CC bit is cleared by processor reset.

Note: The unsigned condition tests for branching or jumping (HI, HS, LO, or LS) can be used only when the condition codes are generated with the CC bit set. Otherwise, the chip does not generate the unsigned conditions correctly.

3.3.2.4 Reserved—Bit 7

This bit field is reserved or not implemented. It is read and written as 0.

3.3.2.5 Stop Delay (SD)—Bit 6

The Stop Delay (SD) bit selects the delay the controller needs to exit the Stop mode. When the SD bit is set, the processor exits quickly from Stop mode. When the SD bit is cleared, the processor exits slowly from Stop mode. The SD bit is cleared by processor reset.

3.3.2.6 Rounding (R)—Bit 5

The Rounding (R) bit selects between two's-complement rounding and convergent rounding. When the R bit is set, two's-complement rounding (always round-up) is used. When the R bit is cleared, convergent rounding is used. The R bit is cleared by processor reset.

3.3.2.7 Saturation (SA)—Bit 4

The Saturation (SA) bit enables automatic saturation on 32-bit arithmetic results, providing a customer-enabled Saturation mode for chip algorithms not recognizing, or cannot take advantage of, the extension accumulator. When the SA bit is set, automatic saturation occurs at the output of the Multiply and Accumulate (MAC) unit for basic arithmetic operations such as multiplication, addition, and so. The saturation is performed by a special saturation circuit inside the MAC unit.

The saturation logic operates by checking three bits of the 36-bit result out of the MAC unit—exp[3], exp[0], and msp[15]. When the SA bit is set, these three bits determine if saturation is performed on the MAC unit's output, and whether to saturate to the maximum positive or negative value, as shown in **Table 3-9.** The SA bit is cleared by processor reset.

Table 3-9. MAC Unit Outputs with Saturation Mode Enabled (SA = 1)

exp[3]	exp[0]	msp[15]	Result Stored in Accumulator
0	0	0	(Unchanged)
0	0	1	$0 7FFF FFFF
0	1	0	$0 7FFF FFFF
0	1	1	$0 7FFF FFFF
1	0	0	$F 8000 0000
1	0	1	$F 8000 0000
1	1	0	$F 8000 0000
1	1	1	(Unchanged)

Note: Saturation mode is always disabled during the execution of the following instructions: ASLL, ASRR, LSLL, LSRR, ASRAC, LSRAC, IMPY16, MPYSU, MACSU, AND, OR, EOR, NOT, LSL, LSR, ROL, and ROR. For these instructions, no saturation is performed at the output of the MAC unit.

3.3.2.8 External X Memory (EX)—Bit 3

The External X Memory (EX) bit is necessary for providing a continuous memory map when using more than 64K of external data memory. When the EX bit is set, all accesses to X memory on the X Address Bus 1 (XAB1) and Core Global Data Bus (CGDB) or Peripheral Global Data Bus (PGDB) are forced to be external, except when a MOVE or bit-field instruction is executed using the I/O Short Addressing mode. In this case, the EX bit is ignored and the access is performed to the on-chip location. When the EX bit is cleared, internal X memory can be accessed with all addressing modes.

The EX bit is ignored by the second read of a dual-read instruction, using the X Address Bus 2 (XAB2) and X Data Bus 2 (XDB2) and always accesses on-chip X data memory. For instructions with two parallel reads, the second read is always performed to internal on-chip memory.

Note: When the EX bit is set, only the *upper* 64 peripheral memory-mapped locations are accessible (X:$FFC0-X:$FFFF) with the I/O Short Addressing mode. The *lower* 64 memory-mapped locations (X:$FF80–X:$FFBF) are not accessible when the EX bit is set. Access to these addresses results in an access to external memory.

The EX bit is cleared by processor reset.

3.3.2.9 Reserved—Bit 2

This bit field is reserved or not implemented. It is read and written as 0.

3.3.2.10 Operating Mode B (MB)—Bit 1

This bit is latched from the EXTBOOT pin on reset. See Section 3.7, “56800 Operating Modes” and **Section 16.5, “External Reset”.**

3.3.2.11 Operating Mode A (MA)—Bit 0

This bit is latched from the EXTBOOT pin on reset. See Section 3.7, “56800 Operating Modes” and **Section 16.5, “External Reset”.**

3.4 Core Configuration Memory Map

Core Configuration (CC) registers are part of the data memory map on the 56800 parts. These locations may be accessed with the same addressing modes used for ordinary data memory when the EX bit is cleared. However, when the EX bit is set, the addresses can only be accessed using the I/O Short Addressing mode. These registers are implemented as part of the 56800 core itself; therefore, they will be present on all family members based on the 56800 core. This is not necessarily true for the on-chip configuration registers discussed in the next section.

Table 3-10 shows the on-chip memory mapped core configuration registers.

Table 3-10. 56800 On-Chip Core Configuration Register Memory Map

X:$FFFF	OPGDBR—OnCE PGDB Bus Transfer Register
X:$FFFE	Reserved
X:$FFFD	Reserved
X:$FFFC	Reserved
X:$FFFB	IPR—Interrupt Priority Register
X:$FFFA	Reserved
X:$FFF9	BCR—Bus Control Register
X:$FF80–$FFF8	Reserved

3.5 On-Chip Peripheral Memory Map

On-Chip Peripheral registers are part of the data memory map on the 56F800 series. These locations may be accessed with the same addressing modes used for ordinary data memory. However, they may not be accessed by write and single read operations when the EX bit is set.

Table 3-11 illustrates the on-Chip Memory Mapped Peripheral registers. The base address represents the starting address used for each peripheral's registers. Register memory locations are assigned in each peripheral chapter based on this base address plus a given offset. Not all peripherals are used on each device. For example, only the 56F807 uses the ADCB peripheral and the PFIU2 section of the program Flash.

Also, the Data Memory Map for the 56F807 starts from a different location for the mapped peripheral registers. Therefore, the base addresses for the 56F807 are different from the ones for the rest of the family. Be sure to read the address range and base address corresponding to the chip you are using in **Table 3-11.**

Here is a list of peripheral abbreviations in the order they appear in **Table 3-11:**

- SIM: System Integration Module

- PFIU2: Program Flash Interface Unit Number Two (used for 56F807 only)
- TMRA: Quad Timer A
- TMRB: Quad Timer B
- TMRC: Quad Timer C
- TMRD: Quad Timer D
- CAN: Controller Area Network
- PWMA: Pulse Width Modulator Module A
- PWMB: Pulse Width Modulator Module B
- DEC0: Quadrature Decoder 0
- DEC1: Quadrature Decoder 1
- ITCN: Interrupt Controller
- ADCA: Analog-to-Digital Converter A
- ADCB: Analog-to-Digital Converter B (used for 56F807 only)
- SCI0: Serial Communications Interface 0
- SCI1: Serial Communications Interface 1
- SPI: Serial Peripheral Interface
- COP: Computer Operation Properly
- PFIU: Program Flash Interface Unit
- DFIU: Data Flash Interface Unit
- BFIU: Boot Flash Interface Unit
- CLKGEN: Clock Generation
- GPIOA: General Purpose I/O Port A
- GPIOB: General Purpose I/O Port B
- GPIOD: General Purpose I/O Port D
- GPIOE: General purpose I/O Port E

Table 3-11. Data Memory Peripheral Address Map

Addresses for 56F801/802/803/805			Address for 56F807		
Peripheral Name	**Address Range for 56F801, 56F802 56F803, & 56F805**	**Base Address for 56F801, 56F802, 56F803, & 56F805**	**Address Range for 56F807**	**Base Address for 56F807**	**Peripheral Register Address Tables**
SIM	$0C00–$0C0F	SYS_BASE=$0C00	$1000–$100F	SYS_BASE=$1000	Table 3-12
		Reserved		Reserved	
		Reserved		Reserved	
		Reserved		Reserved	
		Reserved	$1420-$143F	PFIU2_BASE=$1420	Table 3-13
TMRA	$0D00–$0D1F	TmrA_BASE=$0D00	$1100–$111F	TmrA_BASE=$1100	Table 3-14
TMRB	$0D20–$0D3F	TmrB_BASE=$0D20	$1120–$113F	TmrB_BASE=$1120	Table 3-15
TMRC	$0D40–$0D5F	TmrC_BASE=$0D40	$1140–$115F	TmrC_BASE=$1140	Table 3-16
TMRD	$0D60–$0D7F	TmrD_BASE=$0D60	$1160–$117F	TmrD_BASE=$1160	Table 3-17
CAN	$0D80–$0DFF	CAN_BASE=$0D80	$1180–$11FF	CAN_BASE=$1180	Table 3-18
PWMA	$0E00–$0E1F	PWMA_BASE=$0E00	$1200–$121F	PWMA_BASE=$1200	Table 3-19
PWMB	$0E20–$0E3F	PWMB_BASE=$0E20	$1220–$123F	PWMB_BASE=$1220	Table 3-20
DEC0	$0E40–$0E4F	DEC0_BASE=$0E40	$1240–$124F	DEC0_BASE=$1240	Table 3-21
DEC1	$0E50–$0E5F	DEC1_BASE=$0E50	$1250–$125F	DEC1_BASE=$1250	Table 3-22
ITCN	$0E60–$0E7F	ITCN_BASE=$0E60	$1260–$127F	ITCN_BASE=$1260	Table 3-23
ADCA	$0E80–$0EBF	ADCA_BASE=$0E80	$1280–$12BF	ADCA_BASE=$1280	Table 3-24
ADCB	$0EC0–$0EFF	ADCB_BASE=$0EC0	$12C0–$12FF	ADCB_BASE=$12C0	Table 3-25
SCI0	$0F00–$0F0F	SCI0_BASE=$0F00	$1300–$130F	SCI0_BASE=$1300	Table 3-26
SCI1	$0F10–$0F1F	SCI1_BASE=$0F10	$1310–$131F	SCI1_BASE=$1310	Table 3-27
SPI	$0F20–$0F2F	SPI_BASE=$0F20	$1320–$132F	SPI_BASE=$1320	Table 3-28
COP	$0F30–$0F3F	COP_BASE=$0F30	$1330–$133F	COP_BASE=$1330	Table 3-29
PFIU	$0F40-$0F5F	PFIU_Base=$0F40	$1340-$135F	PFIU_BASE=$1340	Table 3-30
DFIU	$0F60–$0F7F	DFIU_BASE=$0F60	$1360–$137F	DFIU_BASE=$1360	Table 3-31
BFIU	$0F80–$0F9F	BFIU_BASE=$0F80	$1380–$139F	BFIU_BASE=$1380	Table 3-32
CLKGEN	$0FA0–$0FAF	CLKGEN_BASE=$0FA0	$13A0–$13AF	CLKGEN_BASE=$13A0	Table 3-33
GPIOA	$0FB0–$0FBF	GPIOA_BASE=$0FB0	$13B0–$13BF	GPIOA_BASE=$13B0	Table 3-34
GPIOB	$0FC0–$0FCF	GPIOB_BASE=$0FC0	$13C0–$13CF	GPIOB_BASE=$13C0	Table 3-35
		Reserved		Reserved	
GPIOD	$0FE0–$0FEF	GPIOD_BASE=$0FE0	$13E0–$13EF	GPIOD_BASE=$13E0	Table 3-36
GPIOE	$0FF0–$0FFF	GPIOE_BASE=$0FF0	$13F0–$13FF	GPIOE_BASE=$13F0	Table 3-37

Table 3-12. System Control Registers Address Map

Register Abbreviation	Register Description	Address Offset	Base Address
SYS_CNTL	System Control Register	$0	SYS_BASE
SYS_STS	System Status Register	$1	SYS_BASE
MSH_ID	Most Significant Half of JTAG_ID	$6	SYS_BASE
LSH_ID	Least Significant Half of JTAG_ID	$7	SYS_BASE

Table 3-13. Program Flash Interface Unit #2 Registers Address Map

Register Abbreviation	Register Description	Address Offset	Base Address
PFIU2_CNTL	Program Flash #2 Control Register	$0	PFIU2_BASE
PFIU2_PE	Program Flash #2 Program Enable Register	$1	PFIU2_BASE
PFIU2_EE	Program Flash #2 Erase Enable Register	$2	PFIU2_BASE
PFIU2_ADDR	Program Flash #2 Address Register	$3	PFIU2_BASE
PFIU2_DATA	Program Flash #2 Data Register	$4	PFIU2_BASE
PFIU2_IE	Program Flash #2 Interrupt Enable Register	$5	PFIU2_BASE
PFIU2_IS	Program Flash #2 Interrupt Source Register	$6	PFIU2_BASE
PFIU2_IP	Program Flash #2 Interrupt Pending Register	$7	PFIU2_BASE
PFIU2_CKDIVISOR	Program Flash #2 Clock Divisor Register	$8	PFIU2_BASE
PFIU2_TERASEL	Program Flash #2 Terase Limit Register	$9	PFIU2_BASE
PFIU2_TMEL	Program Flash #2 Time Limit Register	$A	PFIU2_BASE
PFIU2_TNVSL	Program Flash #2 Tnvs Limit Register	$B	PFIU2_BASE
PFIU2_TPGSL	Program Flash #2 Tpgs Limit Register	$C	PFIU2_BASE
PFIU2_TPROGL	Program Flash #2 Tprog Limit Register	$D	PFIU2_BASE
PFIU2_TNVHL	Program Flash #2 TNVH Limit Register	$E	PFIU2_BASE
PFIU2_TNVH1L	Program Flash #2 TNVH1 Limit Register	$F	PFIU2_BASE
PFIU2_TRCVL	Program Flash #2 TRCV Limit Register	$10	PFIU2_BASE

Note: Program Flash Interface Unit 2 is used *only* on the 56F807 for the second half of the Program Flash.

Table 3-14. Quad Timer A Registers Address Map

Register Abbreviation	Register Description	Address Offset	Base Address
TMRA0_CMP1	Compare Register	$0	TMRA_BASE
TMRA0_CMP2	Compare Register	$1	TMRA_BASE
TMRA0_CAP	Capture Register	$2	TMRA_BASE
TMRA0_LOAD	Load Register	$3	TMRA_BASE
TMRA0_HOLD	Hold Register	$4	TMRA_BASE
TMRA0_CNTR	Counter	$5	TMRA_BASE

Table 3-14. Quad Timer A Registers Address Map (Continued)

Register Abbreviation	Register Description	Address Offset	Base Address
TMRA0_CTRL	Control Register	$6	TMRA_BASE
TMRA0_SCR	Status and Control Register	$7	TMRA_BASE
TMRA1_CMP1	Compare Register	$8	TMRA_BASE
TMRA1_CMP2	Compare Register	$9	TMRA_BASE
TMRA1_CAP	Capture Register	$A	TMRA_BASE
TMRA1_LOAD	Load Register	$B	TMRA_BASE
TMRA1_HOLD	Hold Register	$C	TMRA_BASE
TMRA1_CNTR	Counter	$D	TMRA_BASE
TMRA1_CTRL	Control Register	$E	TMRA_BASE
TMRA1_SCR	Status and Control Register	$F	TMRA_BASE
TMRA2_CMP1	Compare Register	$10	TMRA_BASE
TMRA2_CMP2	Compare Register	$11	TMRA_BASE
TMRA2_CAP	Capture Register	$12	TMRA_BASE
TMRA2_LOAD	Load Register	$13	TMRA_BASE
TMRA2_HOLD	Hold Register	$14	TMRA_BASE
TMRA2_CNTR	Counter	$15	TMRA_BASE
TMRA2_CTRL	Control Register	$16	TMRA_BASE
TMRA2_SCR	Status and Control Register	$17	TMRA_BASE
TMRA3_CMP1	Compare Register	$18	TMRA_BASE
TMRA3_CMP2	Compare Register	$19	TMRA_BASE
TMRA3_CAP	Capture Register	$1A	TMRA_BASE
TMRA3_LOAD	Load Register	$1B	TMRA_BASE
TMRA3_HOLD	Hold Register	$1C	TMRA_BASE
TMRA3_CNTR	Counter	$1D	TMRA_BASE
TMRA3_CTRL	Control Register	$1E	TMRA_BASE
TMRA3_SCR	Status and Control Register	$1F	TMRA_BASE

Table 3-15. Quad Timer B Registers Address Map

Register Abbreviation	Register Description	Address Offset	Base Address
TMRB0_CMP1	Compare Register	$0	TMRB_BASE
TMRB0_CMP2	Compare Register	$1	TMRB_BASE
TMRB0_CAP	Capture Register	$2	TMRB_BASE
TMRB0_LOAD	Load Register	$3	TMRB_BASE
TMRB0_HOLD	Hold Register	$4	TMRB_BASE
TMRB0_CNTR	Counter	$5	TMRB_BASE

Table 3-15. Quad Timer B Registers Address Map (Continued)

Register Abbreviation	Register Description	Address Offset	Base Address
TMRB0_CTRL	Control Register	$6	TMRB_BASE
TMRB0_SCR	Status and Control Register	$7	TMRB_BASE
TMRB1_CMP1	Compare Register	$8	TMRB_BASE
TMRB1_CMP2	Compare Register	$9	TMRB_BASE
TMRB1_CAP	Capture Register	$A	TMRB_BASE
TMRB1_LOAD	Load Register	$B	TMRB_BASE
TMRB1_HOLD	Hold Register	$C	TMRB_BASE
TMRB1_CNTR	Counter	$D	TMRB_BASE
TMRB1_CTRL	Control Register	$E	TMRB_BASE
TMRB1_SCR	Status and Control Register	$F	TMRB_BASE
TMRB2_CMP1	Compare Register	$10	TMRB_BASE
TMRB2_CMP2	Compare Register	$11	TMRB_BASE
TMRB2_CAP	Capture Register	$12	TMRB_BASE
TMRB2_LOAD	Load Register	$13	TMRB_BASE
TMRB2_HOLD	Hold Register	$14	TMRB_BASE
TMRB2_CNTR	Counter	$15	TMRB_BASE
TMRB2_CTRL	Control Register	$16	TMRB_BASE
TMRB2_SCR	Status and Control Register	$17	TMRB_BASE
TMRB3_CMP1	Compare Register	$18	TMRB_BASE
TMRB3_CMP2	Compare Register	$19	TMRB_BASE
TMRB3_CAP	Capture Register	$1A	TMRB_BASE
TMRB3_LOAD	Load Register	$1B	TMRB_BASE
TMRB3_HOLD	Hold Register	$1C	TMRB_BASE
TMRB3_CNTR	Counter	$1D	TMRB_BASE
TMRB3_CTRL	Control Register	$1E	TMRB_BASE
TMRB3_SCR	Status and Control Register	$1F	TMRB_BASE

Table 3-16. Quad Timer C Registers Address Map

Register Abbreviation	Register Description	Address Offset	Base Address
TMRC0_CMP1	Compare Register	$0	TMRC_BASE
TMRC0_CMP2	Compare Register	$1	TMRC_BASE
TMRC0_CAP	Capture Register	$2	TMRC_BASE

Table 3-16. Quad Timer C Registers Address Map (Continued)

Register Abbreviation	Register Description	Address Offset	Base Address
TMRC0_LOAD	Load Register	$3	TMRC_BASE
TMRC0_HOLD	Hold Register	$4	TMRC_BASE
TMRC0_CNTR	Counter	$5	TMRC_BASE
TMRC0_CTRL	Control Register	$6	TMRC_BASE
TMRC0_SCR	Status and Control Register	$7	TMRC_BASE
TMRC1_CMP1	Compare Register	$8	TMRC_BASE
TMRC1_CMP2	Compare Register	$9	TMRC_BASE
TMRC1_CAP	Capture Register	$A	TMRC_BASE
TMRC1_LOAD	Load Register	$B	TMRC_BASE
TMRC1_HOLD	Hold Register	$C	TMRC_BASE
TMRC1_CNTR	Counter	$D	TMRC_BASE
TMRC1_CTRL	Control Register	$E	TMRC_BASE
TMRC1_SCR	Status and Control Register	$F	TMRC_BASE
TMRC2_CMP1	Compare Register	$10	TMRC_BASE
TMRC2_CMP2	Compare Register	$11	TMRC_BASE
TMRC2_CAP	Capture Register	$12	TMRC_BASE
TMRC2_LOAD	Load Register	$13	TMRC_BASE
TMRC2_HOLD	Hold Register	$14	TMRC_BASE
TMRC2_CNTR	Counter	$15	TMRC_BASE
TMRC2_CTRL	Control Register	$16	TMRC_BASE
TMRC2_SCR	Status and Control Register	$17	TMRC_BASE
TMRC3_CMP1	Compare Register	$18	TMRC_BASE
TMRC3_CMP2	Compare Register	$19	TMRC_BASE
TMRC3_CAP	Capture Register	$1A	TMRC_BASE
TMRC3_LOAD	Load Register	$1B	TMRC_BASE
TMRC3_HOLD	Hold Register	$1C	TMRC_BASE
TMRC3_CNTR	Counter	$1D	TMRC_BASE
TMRC3_CTRL	Control Register	$1E	TMRC_BASE
TMRC3_SCR	Status and Control Register	$1F	TMRC_BASE

Table 3-17. Quad Timer D Registers Address Map

Register Abbreviation	Register Description	Address Offset	Base Address
TMRD0_CMP1	Compare Register	$0	TMRD_BASE
TMRD0_CMP2	Compare Register	$1	TMRD_BASE
TMRD0_CAP	Capture Register	$2	TMRD_BASE
TMRD0_LOAD	Load Register	$3	TMRD_BASE
TMRD0_HOLD	Hold Register	$4	TMRD_BASE

Table 3-17. Quad Timer D Registers Address Map (Continued)

Register Abbreviation	Register Description	Address Offset	Base Address
TMRD0_CNTR	Counter	$5	TMRD_BASE
TMRD0_CTRL	Control Register	$6	TMRD_BASE
TMRD0_SCR	Status and Control Register	$7	TMRD_BASE
TMRD1_CMP1	Compare Register	$8	TMRD_BASE
TMRD1_CMP2	Compare Register	$9	TMRD_BASE
TMRD1_CAP	Capture Register	$A	TMRD_BASE
TMRD1_LOAD	Load Register	$B	TMRD_BASE
TMRD1_HOLD	Hold Register	$C	TMRD_BASE
TMRD1_CNTR	Counter	$D	TMRD_BASE
TMRD1_CTRL	Control Register	$E	TMRD_BASE
TMRD1_SCR	Status and Control Register	$F	TMRD_BASE
TMRD2_CMP1	Compare Register	$10	TMRD_BASE
TMRD2_CMP2	Compare Register	$11	TMRD_BASE
TMRD2_CAP	Capture Register	$12	TMRD_BASE
TMRD2_LOAD	Load Register	$13	TMRD_BASE
TMRD2_HOLD	Hold Register	$14	TMRD_BASE
TMRD2_CNTR	Counter	$15	TMRD_BASE
TMRD2_CTRL	Control Register	$16	TMRD_BASE
TMRD2_SCR	Status and Control Register	$17	TMRD_BASE
TMRD3_CMP1	Compare Register	$18	TMRD_BASE
TMRD3_CMP2	Compare Register	$19	TMRD_BASE
TMRD3_CAP	Capture Register	$1A	TMRD_BASE
TMRD3_LOAD	Load Register	$1B	TMRD_BASE
TMRD3_HOLD	Hold Register	$1C	TMRD_BASE
TMRD3_CNTR	Counter	$1D	TMRD_BASE
TMRD3_CTRL	Control Register	$1E	TMRD_BASE
TMRD3_SCR	Status and Control Register	$1F	TMRD_BASE

Table 3-18. CAN Registers Address Map

Register Abbreviation	Register Description	Address Offset	Base Address
CANCTL0	CAN Control Register 0	$0	CAN_BASE
CANCTL1	CAN Control Register 1	$1	CAN_BASE
CANBTR0	CAN Bus Timing Register 0	$2	CAN_BASE
CANBTR1	CAN Bus Timing Register 1	$3	CAN_BASE
CANRFLG	CAN Receiver Flag Register	$4	CAN_BASE
CANRIER	CAN Receiver Interrupt Enable Register	$5	CAN_BASE

Table 3-18. CAN Registers Address Map (Continued)

Register Abbreviation	Register Description	Address Offset	Base Address
CANTFLG	CAN Transmitter Flag Register	$6	CAN_BASE
CANTCR	CAN Transmitter Control Register	$7	CAN_BASE
CANIDAC	Identifier Acceptance Control Register	$8	CAN_BASE
CANRXERR	CAN Receiver Error Counter Register	$E	CAN_BASE
CANTXERR	CAN Transmit Error Counter Register	$F	CAN_BASE
CANIDAR0–7	CAN Identifier Acceptance Registers 0–7	$10, $11, $12, $13, $18, $19, $1A, $1B	CAN_BASE
CANIDMR0–7	CAN Identifier Mask Registers 0–7	$14, $15, $16, $17, $1C, $1D, $1E, $1F	CAN_BASE
CAN_RB_IDR0–3	Receive Buffer Identifier Registers 0–3	$40, $41, $42, $43	CAN_BASE
CAN_RB_DSR0–7	Receive Buffer Data Segment Registers 0–7	$44, $45, $46, $47, $48, $49, $4A, $4B	CAN_BASE
CAN_RB_DLR	Receive Buffer Data Length Register	$4C	CAN_BASE
CAN_RB_TBPR	Receive Buffer Transmit Buffer Priority Register	$4D	CAN_BASE
CAN_TB0_IDR0–3	Transmit Buffer 0 Identifier Registers 0–3	$50, $51, $52, $53	CAN_BASE
CAN_TB0_DSR0–7	Transmit Buffer 0 Data Segment Registers 0–7	$54, $55, $56, $57, $58, $59, $5A, $5B	CAN_BASE
CAN_TB0_DLR	Transmit Buffer 0 Data Length Register	$5C	CAN_BASE
CAN_TB0_TBPR	Transmit Buffer 0 Transmit Buffer Priority Register	$5D	CAN_BASE
CAN_TB1_IDR0–3	Transmit Buffer 1 Identifier Registers 0–3	$60, $61, $62, $63	CAN_BASE
CAN_TB1_DSR0–7	Transmit Buffer 1 Data Segment Registers 0–7	$64, $65, $66, $67, $68, $69, $6A, $6B	CAN_BASE
CAN_TB1_DLR	Transmit Buffer 1 Data Length Register	$6C	CAN_BASE
CAN_TB1_TBPR	Transmit Buffer 1 Transmit Buffer Priority Register	$6D	CAN_BASE
CAN_TB2_IDR0–3	Transmit Buffer 2 Identifier Registers 0–3	$70, $71, $72, $73	CAN_BASE
CAN_TB2_DSR0–7	Transmit Buffer 2 Data Segment Registers 0–7	$74, $75, $76, $77, $78, $79, $7A, $7B	CAN_BASE
CAN_TB2_DLR	Transmit Buffer 2 Data Length Register	$7C	CAN_BASE
CAN_TB2_TBPR	Transmit Buffer 2 Transmit Buffer Priority Register	$7D	CAN_BASE

Table 3-19. PWMA Registers Address Map

Register Abbreviation	Register Description	Address Offset	Base Address
PWMA_PMCTL	PWM Control Register	$0	PWMA_BASE
PWMA_PMFCTL	PWM Fault Control Register	$1	PWMA_BASE
PWMA_PMFSA	PWM Fault Status Acknowledge	$2	PWMA_BASE
PWMA_PMOUT	PWM Output Control Register	$3	PWMA_BASE
PWMA_PMCNT	PWM Counter Register	$4	PWMA_BASE
PWMA_PWMCM	PWM Counter Modulo Register	$5	PWMA_BASE

Table 3-19. PWMA Registers Address Map (Continued)

Register Abbreviation	Register Description	Address Offset	Base Address
PWMA_PWMVAL0	PWM Value Register 0	$6	PWMA_BASE
PWMA_PWMVAL1	PWM Value Register 1	$7	PWMA_BASE
PWMA_PWMVAL2	PWM Value Register 2	$8	PWMA_BASE
PWMA_PWMVAL3	PWM Value Register 3	$9	PWMA_BASE
PWMA_PWMVAL4	PWM Value Register 4	$A	PWMA_BASE
PWMA_PWMVAL5	PWM Value Register 5	$B	PWMA_BASE
PWMA_PMDEADTM	PWM Deadtime Register	$C	PWMA_BASE
PWMA_PMDISMAP1	PWM Disable Mapping Register 1	$D	PWMA_BASE
PWMA_PMDISMAP2	PWM Disable Mapping Register 2	$E	PWMA_BASE
PWMA_PMCFG	PWM Configure Register	$F	PWMA_BASE
PWMA_PMCCR	PWM Channel Control Register	$10	PWMA_BASE
PWMA_PMPORT	PWM Port Register	$11	PWMA_BASE

Table 3-20. PWMB Registers Address Map

Register Abbreviation	Register Description	Address Offset	Base Address
PWMB_PMCTL	PWM Control Register	$0	PWMB_BASE
PWMB_PMFCTL	PWM Fault Control Register	$1	PWMB_BASE
PWMB_PMFSA	PWM Fault Status Acknowledge	$2	PWMB_BASE
PWMB_PMOUT	PWM Output Control Register	$3	PWMB_BASE
PWMB_PMCNT	PWM Counter Register	$4	PWMB_BASE
PWMB_PWMCM	PWM Counter Modulo Register	$5	PWMB_BASE
PWMB_PWMVAL0	PWM Value Register 0	$6	PWMB_BASE
PWMB_PWMVAL1	PWM Value Register 1	$7	PWMB_BASE
PWMB_PWMVAL2	PWM Value Register 2	$8	PWMB_BASE
PWMB_PWMVAL3	PWM Value Register 3	$9	PWMB_BASE
PWMB_PWMVAL4	PWM Value Register 4	$A	PWMB_BASE
PWMB_PWMVAL5	PWM Value Register 5	$B	PWMB_BASE
PWMB_PMDEADTM	PWM Deadtime Register	$C	PWMB_BASE
PWMB_PMDISMAP1	PWM Disable Mapping Register 1	$D	PWMB_BASE
PWMB_PMDISMAP2	PWM Disable Mapping Register 2	$E	PWMB_BASE
PWMB_PMCFG	PWM Configure Register	$F	PWMB_BASE
PWMB_PMCCR	PWM Channel Control Register	$10	PWMB_BASE
PWMB_PMPORT	PWM Port Register	$11	PWMB_BASE

Table 3-21. Quadrature Decoder #0 Registers Address Map

Register Abbreviation	Register Description	Address Offset	Base Address
QD0_DECCR	Decoder Control Register	$0	DEC0_BASE
QD0_FIR	Filter Interval Register	$1	DEC0_BASE
QD0_WTR	Watchdog Timeout Register	$2	DEC0_BASE
QD0_POSD	Position Difference Counter Register	$3	DEC0_BASE
QD0_POSDH	Position Difference Counter Hold Register	$4	DEC0_BASE
QD0_REV	Revolution Counter Register	$5	DEC0_BASE
QD0_REVH	Revolution Hold Register	$6	DEC0_BASE
QD0_UPOS	Upper Position Counter Register	$7	DEC0_BASE
QD0_LPOS	Lower Position Counter Register	$8	DEC0_BASE
QD0_UPOSH	Upper Position Hold Register	$9	DEC0_BASE
QD0_LPOSH	Lower Position Hold Register	$A	DEC0_BASE
QD0_UIR	Upper Initialization Register	$B	DEC0_BASE
QD0_LIR	Lower Initialization Register	$C	DEC0_BASE
QD0_IMR	Input Monitor Register	$D	DEC0_BASE

Table 3-22. Quadrature Decoder #1 Registers Address Map

Register Abbreviation	Register Description	Address Offset	Base Address
QD1_DECCR	Decoder Control Register	$0	DEC1_BASE
QD1_FIR	Filter Interval Register	$1	DEC1_BASE
QD1_WTR	Watchdog Timeout Register	$2	DEC1_BASE
QD1_POSD	Position Difference Counter Register	$3	DEC1_BASE
QD1_POSDH	Position Difference Counter Hold Register	$4	DEC1_BASE
QD1_REV	Revolution Counter Register	$5	DEC1_BASE
QD1_REVH	Revolution Hold Register	$6	DEC1_BASE
QD1_UPOS	Upper Position Counter Register	$7	DEC1_BASE
QD1_LPOS	Lower Position Counter Register	$8	DEC1_BASE
QD1_UPOSH	Upper Position Hold Register	$9	DEC1_BASE
QD1_LPOSH	Lower Position Hold Register	$A	DEC1_BASE
QD1_UIR	Upper Initialization Register	$B	DEC1_BASE
QD1_LIR	Lower Initialization Register	$C	DEC1_BASE
QD1_IMR	Input Monitor Register	$D	DEC1_BASE

Table 3-23. Interrupt Controller Registers Address Map

Register Abbreviation	Register Description	Address Offset	Base Address
ITCN_GPR0	Group Priority Register 0	$0	ITCN_BASE
ITCN_GPR1	Group Priority Register 1	$1	ITCN_BASE
ITCN_GPR2	Group Priority Register 2	$2	ITCN_BASE
ITCN_GPR3	Group Priority Register 3	$3	ITCN_BASE
ITCN_GPR4	Group Priority Register 4	$4	ITCN_BASE
ITCN_GPR5	Group Priority Register 5	$5	ITCN_BASE
ITCN_GPR6	Group Priority Register 6	$6	ITCN_BASE
ITCN_GPR7	Group Priority Register 7	$7	ITCN_BASE
ITCN_GPR8	Group Priority Register 8	$8	ITCN_BASE
ITCN_GPR9	Group Priority Register 9	$9	ITCN_BASE
ITCN_GPR10	Group Priority Register 10	$A	ITCN_BASE
ITCN_GPR11	Group Priority Register 11	$B	ITCN_BASE
ITCN_GPR12	Group Priority Register 12	$C	ITCN_BASE
ITCN_GPR13	Group Priority Register 13	$D	ITCN_BASE
ITCN_GPR14	Group Priority Register 14	$E	ITCN_BASE
ITCN_GPR15	Group Priority Register 15	$F	ITCN_BASE

Table 3-24. ADCA Registers Address Map

Register Abbreviation	Register Description	Address Offset	Base Address
ADCA_ADCR1	ADC Control Register 1	$0	ADCA_BASE
ADCA_ADCR2	ADC Control Register 2	$1	ADCA_BASE
ADCA_ADZCC	ADC Zero Crossing Control Register	$2	ADCA_BASE
ADCA_ADLST1	ADC Channel List Registers 1	$3	ADCA_BASE
ADCA_ADLST2	ADC Channel List Register 2	$4	ADCA_BASE
ADCA_ADSDIS	ADC Sample Disable Register	$5	ADCA_BASE
ADCA_ADSTAT	ADC Status Register	$6	ADCA_BASE
ADCA_ADLSTAT	ADC Limit Status Register	$7	ADCA_BASE
ADCA_ADZCSTAT	ADC Zero Crossing Status Register	$8	ADCA_BASE
ADCA_ADRSLT0–7	ADC Result Registers 0–7	$9, $A, $B, $C, $D, $E, $F, $10	ADCA_BASE
ADCA_ADLLMT0–7	ADC Low Limit Registers 0–7	$11, $12, $13, $14, $15, $16, $17, $18	ADCA_BASE
ADCA_ADHLMT0–7	ADC High Limit Registers 0–7	$19, $1A, $1B, $1C, $1D, $1E, $1F, $20	ADCA_BASE
ADCA_ADOFS0–7	ADC Offset Registers 0–7	$21, $22, $23, $24, $25, $26, $27, $28	ADCA_BASE

Table 3-25. ADCB Registers Address Map

Register Abbreviation	Register Description	Address Offset	Base Address
ADCB_ADCR1	ADC Control Register 1	$0	ADCB_BASE
ADCB_ADCR2	ADC Control Register 2	$1	ADCB_BASE
ADCB_ADZCC	ADC Zero Crossing Control Register	$2	ADCB_BASE
ADCB_ADLST1	ADC Channel List Registers 1	$3	ADCB_BASE
ADCB_ADLST2	ADC Channel List Register 2	$4	ADCB_BASE
ADCB_ADSDIS	ADC Sample Disable Register	$5	ADCB_BASE
ADCB_ADSTAT	ADC Status Register	$6	ADCB_BASE
ADCB_ADLSTAT	ADC Limit Status Register	$7	ADCB_BASE
ADCB_ADZCSTAT	ADC Zero Crossing Status Register	$8	ADCB_BASE
ADCB_ADRSLT0–7	ADC Result Registers 0–7	$9, $A, $B, $C, $D, $E, $F, $10	ADCB_BASE
ADCB_ADLLMT0–7	ADC Low Limit Registers 0–7	$11, $12, $13, $14, $15, $16, $17, $18	ADCB_BASE
ADCB_ADHLMT0–7	ADC High Limit Registers 0–7	$19, $1A, $1B, $1C, $1D, $1E, $1F, $20	ADCB_BASE
ADCB_ADOFS0–7	ADC Offset Registers 0–7	$21, $22, $23, $24, $25, $26, $27, $28	ADCB_BASE

Note: Analog to Digital Converter B (ADCB) is used *only* on the 56F807.

Table 3-26. SCI0 Registers Address Map

Register Abbreviation	Register Description	Address Offset	Base Address
SCI0_SCIBR	SCI Baud Rate Register	$0	SCI0_BASE
SCI0_SCICR	SCI Control Register	$1	SC0_BASE
SCI0_SCISR	SCI Status Register	$2	SCI0_BASE
SCI0_SCIDR	SCI Data Register	$3	SCI0_BASE

Table 3-27. SCI1 Registers Address Map

Register Abbreviation	Register Description	Address Offset	Base Address
SCI1_SCIBR	SCI Baud Rate Register	$0	SCI1_BASE
SCI1_SCICR	SCI Control Register	$1	SCI1_BASE
SCI1_SCISR	SCI Status Register	$2	SCI1_BASE
SCI1_SCIDR	SCI Data Register	$3	SCI1_BASE

Table 3-28. SPI Registers Address Map

Register Abbreviation	Register Description	Address Offset	Base Address
SPSCR	SPI Status and Control Register	$0	SPI_BASE
SPDSR	SPI Data Size Register	$1	SPI_BASE
SPDRR	SPI Data Receive Register	$2	SPI_BASE

Table 3-28. SPI Registers Address Map (Continued)

Register Abbreviation	Register Description	Address Offset	Base Address
SPDTR	SPI Data Transmit Register	$3	SPI_BASE

Table 3-29. COP Registers Address Map

Register Abbreviation	Register Description	Address Offset	Base Address
COPCTL	COP Control Register	$0	COP_BASE
COPTO	COP Time Out Register	$1	COP_BASE
COPSRV	COP Service Register	$2	COP_BASE

Table 3-30. Program Flash Interface Unit Registers Address Map

Register Abbreviation	Register Description	Address Offset	Base Address
PFIU_CNTL	Program Flash Control Register	$0	PFIU_BASE
PFIU_PE	Program Flash Program Enable Register	$1	PFIU_BASE
PFIU_EE	Program Flash Erase Enable Register	$2	PFIU_BASE
PFIU_ADDR	Program Flash Address Register	$3	PFIU_BASE
PFIU_DATA	Program Flash Data Register	$4	PFIU_BASE
PFIU_IE	Program Flash Interrupt Enable Register	$5	PFIU_BASE
PFIU_IS	Program Flash Interrupt Source Register	$6	PFIU_BASE
PFIU_IP	Program Flash Interrupt Pending Register	$7	PFIU_BASE
PFIU_CKDIVISOR	Program Flash Clock Divisor Register	$8	PFIU_BASE
PFIU_TERASEL	Program Flash Terase Limit Register	$9	PFIU_BASE
PFIU_TMEL	Program Flash Time Limit Register	$A	PFIU_BASE
PFIU_TNVSL	Program Flash Tnvs Limit Register	$B	PFIU_BASE
PFIU_TPGSL	Program Flash Tpgs Limit Register	$C	PFIU_BASE
PFIU_TPROGL	Program Flash Tprog Limit Register	$D	PFIU_BASE
PFIU_TNVHL	Program Flash TNVH Limit Register	$E	PFIU_BASE
PFIU_TNVH1L	Program Flash TNVH1 Limit Register	$F	PFIU_BASE
PFIU_TRCVL	Program Flash TRCV Limit Register	$10	PFIU_BASE

Table 3-31. Data Flash Interface Unit Registers Address Map

Register Abbreviation	Register Description	Address Offset	Base Address
DFIU_CNTL	Data Flash Control Register	$0	DFIU_BASE
DFIU_PE	Data Flash Program Enable Register	$1	DFIU_BASE
DFIU_EE	Data Flash Erase Enable Register	$2	DFIU_BASE
DFIU_ADDR	Data Flash Address Register	$3	DFIU_BASE
DFIU_DATA	Data Flash Data Register	$4	DFIU_BASE
DFIU_IE	Data Flash Interrupt Enable Register	$5	DFIU_BASE
DFIU_IS	Data Flash Interrupt Source Register	$6	DFIU_BASE
DFIU_IP	Data Flash Interrupt Pending Register	$7	DFIU_BASE
DFIU_CKDIVISOR	Data Flash Clock Divisor Register	$8	DFIU_BASE
DFIU_TERASEL	Data Flash Terase Limit Register	$9	DFIU_BASE
DFIU_TMEL	Data Flash TME Limit Register	$A	DFIU_BASE
DFIU_TNVSL	Data Flash TNVS Limit Register	$B	DFIU_BASE
DFIU_TPGSL	Data Flash TPGS Limit Register	$C	DFIU_BASE
DFIU_TPROGL	Data Flash TPROG Limit Register	$D	DFIU_BASE
DFIU_TNVHL	Data Flash TNVH Limit Register	$E	DFIU_BASE
DFIU_TNVHL1	Data Flash TNVH1 Limit Register	$F	DFIU_BASE
DFIU_TRCVL	Data Flash TRCV Limit Register	$10	DFIU_BASE

Table 3-32. Boot Flash Interface Unit Registers Address Map

Register Abbreviation	Register Description	Address Offset	Base Address
BFIU_CNTL	Boot Flash Control Register	$0	BFIU_BASE
BFIU_PE	Boot Flash Program Enable Register	$1	BFIU_BASE
BFIU_EE	Boot Flash Erase Enable Register	$2	BFIU_BASE
BFIU_ADDR	Boot Flash Address Register	$3	BFIU_BASE
BFIU_DATA	Boot Flash Data Register	$4	BFIU_BASE
BFIU_IE	Boot Flash Interrupt Enable Register	$5	BFIU_BASE
BFIU_IS	Boot Flash Interrupt Source Register	$6	BFIU_BASE
BFIU_IP	Boot Flash Interrupt Pending Register	$7	BFIU_BASE
BFIU_CKDIVISOR	Boot Flash Clock Divisor Register	$8	BFIU_BASE
BFIU_TERASEL	Boot Flash Terase Limit Register	$9	BFIU_BASE
BFIU_TMEL	Boot Flash TME Limit Register	$A	BFIU_BASE
BFIU_TNVSL	Boot Flash TNVS Limit Register	$B	BFIU_BASE
BFIU_TPGSL	Boot Flash TPGS Limit Register	$C	BFIU_BASE
BFIU_TPROGL	Boot Flash TPROG Limit Register	$D	BFIU_BASE

Table 3-32. Boot Flash Interface Unit Registers Address Map (Continued)

Register Abbreviation	Register Description	Address Offset	Base Address
BFIU_TNVHL	Boot Flash TNVH Limit Register	$E	BFIU_BASE
BFIU_TNVHL1	Boot Flash TNVH1 Limit Register	$F	BFIU_BASE
BFIU_TRCVL	Boot Flash TRCV Limit Register	$10	BFIU_BASE

Table 3-33. Clock Generation Registers Address Map

Register Abbreviation	Register Description	Address Offset	Base Address
PLLCR	Control Register	$0	CLKGEN_BASE
PLLDB	Divide-By Register	$1	CLKGEN_BASE
PLLSR	Status Register	$2	CLKGEN_BASE
	Reserved		
CLKOSR	Select Register	$4	CLKGEN_BASE
ISOCTL	Oscillator Control Register	$5	CLKGEN_BASE

Table 3-34. GPIO Port A Registers Address Map

Register Abbreviation	Register Description	Address Offset	Base Address
GPIO_A_PUR	Pull-Up Enable Register	$0	GPIOA_BASE
GPIO_A_DR	Data Register	$1	GPIOA_BASE
GPIO_A_DDR	Data Direction Register	$2	GPIOA_BASE
GPIO_A_PER	Peripheral Enable Register	$3	GPIOA_BASE
GPIO_A_IAR	Interrupt Assert Register	$4	GPIOA_BASE
GPIO_A_IENR	Interrupt Enable Register	$5	GPIOA_BASE
GPIO_A_IPOLR	Interrupt Polarity Register	$6	GPIOA_BASE
GPIO_A_IPR	Interrupt Pending Register	$7	GPIOA_BASE
GPIO_A_IESR	Interrupt Edge-Sensitive Register	$8	GPIOA_BASE

Table 3-35. GPIO Port B Registers Address Map

Register Abbreviation	Register Description	Address Offset	Base Address
GPIO_B_PUR	Pull-Up Enable Register	$0	GPIOB_BASE
GPIO_B_DR	Data Register	$1	GPIOB_BASE
GPIO_B_DDR	Data Direction Register	$2	GPIOB_BASE

Table 3-35. GPIO Port B Registers Address Map (Continued)

Register Abbreviation	Register Description	Address Offset	Base Address
GPIO_B_PER	Peripheral Enable Register	$3	GPIOB_BASE
GPIO_B_IAR	Interrupt Assert Register	$4	GPIOB_BASE
GPIO_B_IENR	Interrupt Enable Register	$5	GPIOB_BASE
GPIO_B_IPOLR	Interrupt Polarity Register	$6	GPIOB_BASE
GPIO_B_IPR	Interrupt Pending Register	$7	GPIOB_BASE
GPIO_B_IESR	Interrupt Edge-Sensitive Register	$8	GPIOB_BASE

Table 3-36. GPIO Port D Registers Address Map

Register Abbreviation	Register Description	Address Offset	Base Address
GPIO_D_PUR	Pull-Up Enable Register	$0	GPIOD_BASE
GPIO_D_DR	Data Register	$1	GPIOD_BASE
GPIO_D_DDR	Data Direction Register	$2	GPIOD_BASE
GPIO_D_PER	Peripheral Enable Register	$3	GPIOD_BASE
GPIO_D_IAR	Interrupt Assert Register	$4	GPIOD_BASE
GPIO_D_IENR	Interrupt Enable Register	$5	GPIOD_BASE
GPIO_D_IPOLR	Interrupt Polarity Register	$6	GPIOD_BASE
GPIO_D_IPR	Interrupt Pending Register	$7	GPIOD_BASE
GPIO_D_IESR	Interrupt Edge-Sensitive Register	$8	GPIOD_BASE

Table 3-37. GPIO Port E Registers Address Map

Register Abbreviation	Register Description	Address Offset	Base Address
GPIO_E_PUR	Pull-Up Enable Register	$0	GPIOE_BASE
GPIO_E_DR	Data Register	$1	GPIOE_BASE
GPIO_E_DDR	Data Direction Register	$2	GPIOE_BASE
GPIO_E_PER	Peripheral Enable Register	$3	GPIOE_BASE
GPIO_E_IAR	Interrupt Assert Register	$4	GPIOE_BASE
GPIO_E_IENR	Interrupt Enable Register	$5	GPIOE_BASE
GPIO_E_IPOLR	Interrupt Polarity Register	$6	GPIOE_BASE
GPIO_E_IPR	Interrupt Pending Register	$7	GPIOE_BASE
GPIO_E_IESR	Interrupt Edge-Sensitive Register	$8	GPIOE_BASE

3.6 Program Memory

Table 3-1 shows the 56F801/802 has 8188 words of on-chip program Flash and 1K words of on-chip program RAM. However, the 56F803 and 805 chips have 32.252 words of on-chip Program Flash Memory and 512 words of on-chip program RAM. The 56F807 has 61,436 words of on-chip Program Flash Memory and 2K words of on-chip program RAM.

For the 56F803 to 56F807 chips, the program memory may be expanded off-chip up to 64K. The external program bus access time is controlled by two of four bits of the Bus Control Register (BCR) located at X:$FFF9[1]. This register is shown in **Figure 3-1.**

The on-chip program Flash and RAM may hold a combination of interrupt vectors and program code, which may be modified by the application itself.

When mode three is selected, the complete 64K words of program memory are external.

The 56F801/802 do not permit external memory expansion.

3.7 56800 Operating Modes

The 56F80x chips have two valid operating modes determining the memory maps for program memory. Operating modes can be selected either by applying the appropriate signal to the EXTBOOT pin during reset, or by writing to the Operating Mode Register (OMR) and changing the MA and MB bits.

Table 3-38. Program Memory Chip Operating Modes

State of EXTBOOT Upon Reset	MB	MA	Chip Operating Mode
0	0	0	Mode 0 NORMAL Operation
N/A	0	1	NOT SUPPORTED
N/A	1	0	
1	1	1	Mode 3 EXTERNAL RAM

The EXTBOOT pin is sampled as the chip leaves the reset state, and the initial operating mode of the chip is set accordingly.

Chip operating modes can also be changed by writing to the operating mode bits MB and MA in the OMR. Changing operating modes does not reset the chip. Interrupts should be disabled immediately before changing the OMR. This will prevent an interrupt from going to the wrong memory location. Also, one No-Operation (NOP) instruction should be included after changing the OMR to allow for re-mapping to occur.

1. All 56800 chips have two low order wait state bits in BCR hardcoded to zero.

Note: Upon a Computer Operating Properly (COP) reset, the MA and MB bits will revert to the values originally latched from the EXTBOOT pin in contradiction of $\overline{\text{RESET}}$, hardware reset. These *original* mode values determine the COP reset vector.

3.7.1 Mode 0–Single Chip Mode: Start-Up

Mode zero is the single-chip mode. Internal Program RAM (PRAM) and PFLASH are enabled for reads and fetches. The 56F80x have two sub-modes for:

1. Mode A boot mode where all memory is internal
2. Mode B non-boot mode where the first 32K of memory is internal and the second 32K is external

If EXTBOOT is deasserted low during reset, then Mode A boot is automatically entered when exiting reset mode.

For the 56F801/802, Mode B are not supported because there is no external memory interface.

Note: Locations 0 through 3 in the program memory space are actually mapped to the first four locations in the Boot Flash.

Mode 0 is useful to enter when exiting reset for applications while executing primarily from internal program memory. The reset vector location in Modes 0 and 3 is located in the program memory space at location P:$0000, P:$0002 for COP timer reset. For Mode 0, this is in internal program memory. In Mode 3, it is in off-chip program memory.

3.7.2 Modes 1 and 2

Modes 1 and 2 are NOT SUPPORTED for these parts. They are used for ROM-based members of the 56800 family.

3.7.3 Mode 3–External

Mode 3 is a development mode in which the entire 64K program memory space is external. No internal program memory may be accessed, except as a secondary read of data RAM. The reset vector location in Mode 3 is located in the external program memory space at location P:$0000, P:$0002 for COP timer reset.

3.8 Boot Flash Operation

A Boot Flash is provided to handle device initialization in the event the program Flash becomes corrupted for any reason. The hardware and COP reset vectors, $0000 and $0002, are mapped into the Boot Flash at locations $8000 and $8002. The entire Boot Flash is available at locations $8000 through $87FF in the 801/802/803/805. The Boot Flash would normally be programmed once just before or after the device is mounted in the end application. Code in the Boot Flash is

typically responsible for checking the contents of PFLASH are correct, reloading the PFLASH, from SPI, SCI, CAN, and so on, if necessary. The 56F807 Boot Flash is available at locations $F800 through $FFFF. Please see **Table 3-39.**

Table 3-39. Example Contents of Data Stream to be Loaded from Serial EEPROM

Word Number	Description
1	N = Number of program words to be loaded (must be at least one word LESS than the size of the area to be loaded).
2	Starting address for load
3 -≥ N + 2	Code to be loaded

The code in the Boot Flash could contain an algorithm for checking the current contents of the PFLASH against a previously computed signature stored in the last entry of the PFLASH. If these agree, the PFLASH contents are assumed good, and the memory map can be switched, redirecting the program control to the PFLASH. If the entries do not agree, then the PFLASH is reloaded through one of the external ports. Control is then redirected to the code just loaded into memory.

The mechanism above applies only to the case where the device is being booted in Mode 0. It does not apply when the device is booted in Mode 3, where all program memory is external.

The 56F80x chips contain 2K of Boot Flash. An example algorithm for the Boot Flash follows.

Example 3-1. Example Boot Flash Algorithm

```
assume modulus1 = a prime number
assume max = maximum address of program Flash

/* Calculate a signature for the current contents of the program Flash */
/* (assume addresses are normalized to the pflash root */
for (i = 0; i<max; i++) hash_no = (64*hash_no + pflash[i]) % modulus1;

If (hash_no == pflash[max]) begin
reset memory map by writing to the system control register
JUMP to jumpAddr
end else begin /* need to reload contents of PFLASH */

LOAD CODE VIA SPI PORT
COMPUTE NEW SIGNATURE AS DATA IS LOADED VIA SPI
ERASE FLASH IF NECESSARY
PROGRAM FLASH WITH DATA FROM SPI
SET LAST LOCATION TO BE LOADED = HASHCODE
SET jumpAddr = Starting Address From SPI

end
reset memory map by writing to the system control register
jump to jumpAddr
```

Note: The algorithm above ignores users are dealing with a limited amount of on-chip RAM. Size limited to available RAM will download data in buckets. Further, the algorithm also needs to be extended to support multiple EEPROM types and CAN.

3.9 Executing Programs from XRAM

56F80x devices do *not* support execution of program from data RAM (XRAM).

3.10 56800 Reset and Interrupt Vectors

The reset and interrupt vector map specifies the address the processor jumps to when it recognizes an interrupt or encounters a reset condition. The instruction located at this address must be a JSR instruction for an interrupt vector, or a JMP instruction for a reset vector. The interrupt vector map for a given chip is specified by all possible interrupt sources on the 56800 core, as well as from the peripherals. No Interrupt Priority Level (IPL) is specified for hardware reset or for COP reset because these conditions reset the chip. Resetting takes precedence over all other interrupts.

Table 3-40 provides the reset and interrupt priority structure for the 56F80x chips, including on-chip peripherals. **Table 3-41** lists the reset and interrupt vectors for this family. A full description of interrupts is provided in the *DSP56800 Family Manual (DSP56800FM).*

Note: In Operating Modes 0 and 3, the hardware reset vector is at $0000 and the COP reset vector is at $0002.

Table 3-40. Reset and Interrupt Priority Structure

Priority	Exception	IPR Bits[1]
Level 1 (Non-maskable)		
Highest	Hardware $\overline{\text{RESET}}$	—
	COP Timer Reset	—
	Illegal Instruction Trap	—
	Hardware Stack Overflow	—
	OnCE Trap	—
Lower	SWI	—
Level 0 (Maskable)		
Higher	$\overline{\text{IRQA}}$ (External interrupt)	2, 1, 0
	$\overline{\text{IRQB}}$ (External interrupt)	5, 4, 3
	Channel 6 Peripheral Interrupt	9
	Channel 5 Peripheral Interrupt	10
	Channel 4 Peripheral Interrupt	11

Table 3-40. Reset and Interrupt Priority Structure (Continued)

Priority	Exception	IPR Bits[1]
Level 1 (Non-maskable)		
Highest	Hardware $\overline{\text{RESET}}$	—
	COP Timer Reset	—
	Illegal Instruction Trap	—
	Hardware Stack Overflow	—
	OnCE Trap	—
Lower	SWI	—
Level 0 (Maskable)		
Higher	$\overline{\text{IRQA}}$ (External interrupt)	2, 1, 0
	Channel 3 Peripheral Interrupt	12
	Channel 2 Peripheral Interrupt	13
	Channel 1 Peripheral Interrupt	14
Lowest	Channel 0 Peripheral Interrupt	15

1. Please refer to Section 4.7, “Interrupt Priority Register (IPR)”

Table 3-41. Reset and Interrupt Vector Map

Reset and Interrupt Starting Addresses[1]	Interrupt Priority Level	Interrupt Source
$0000[2]	—	Hardware $\overline{\text{RESET}}$
$0002[2]	—	COP Timer Reset
	—	Reserved
$0006	1	Illegal Instruction Trap
$0008		Software Interrupt (SWI)
$000A		Hardware Stack Overflow
$000C		OnCE Trap
	—	Reserved

Table 3-41. Reset and Interrupt Vector Map (Continued)

Reset and Interrupt Starting Addresses[1]	Interrupt Priority Level	Interrupt Source
$0010	0	$\overline{\text{IRQA}}$
$0012		$\overline{\text{IRQB}}$
$0014		Peripheral Interrupt Vectors[3]
$0016		
$0018		
•		
•		
•		
•		
•		
$0042		
$007C		
$007E		

1. These are 56F80x addresses, not IPBus addresses.
2. Reset vectors are aliased to the first two vectors located in Boot Flash.

   56F801/802/803/805 : $0000 = $8000
   $0002 = $8002
   56F807 : $0000 = $F800
   $0002 = $F802
3. Please see **Section 4.1, "Introduction"** for specific peripheral interrupt memory map locations.

3.11 Memory Architecture

The multiple bus Harvard architecture of the core within this device allows for certain dual, parallel moves. This greatly increases throughput on algorithms requiring a high bandwidth feed of data values. The following simplified diagram of the chips' on-board address and data buses helps illustrate the allowed and disallowed parallel data reads. As with any Harvard architecture, the *program* or op-code buses always work in parallel with the data buses.

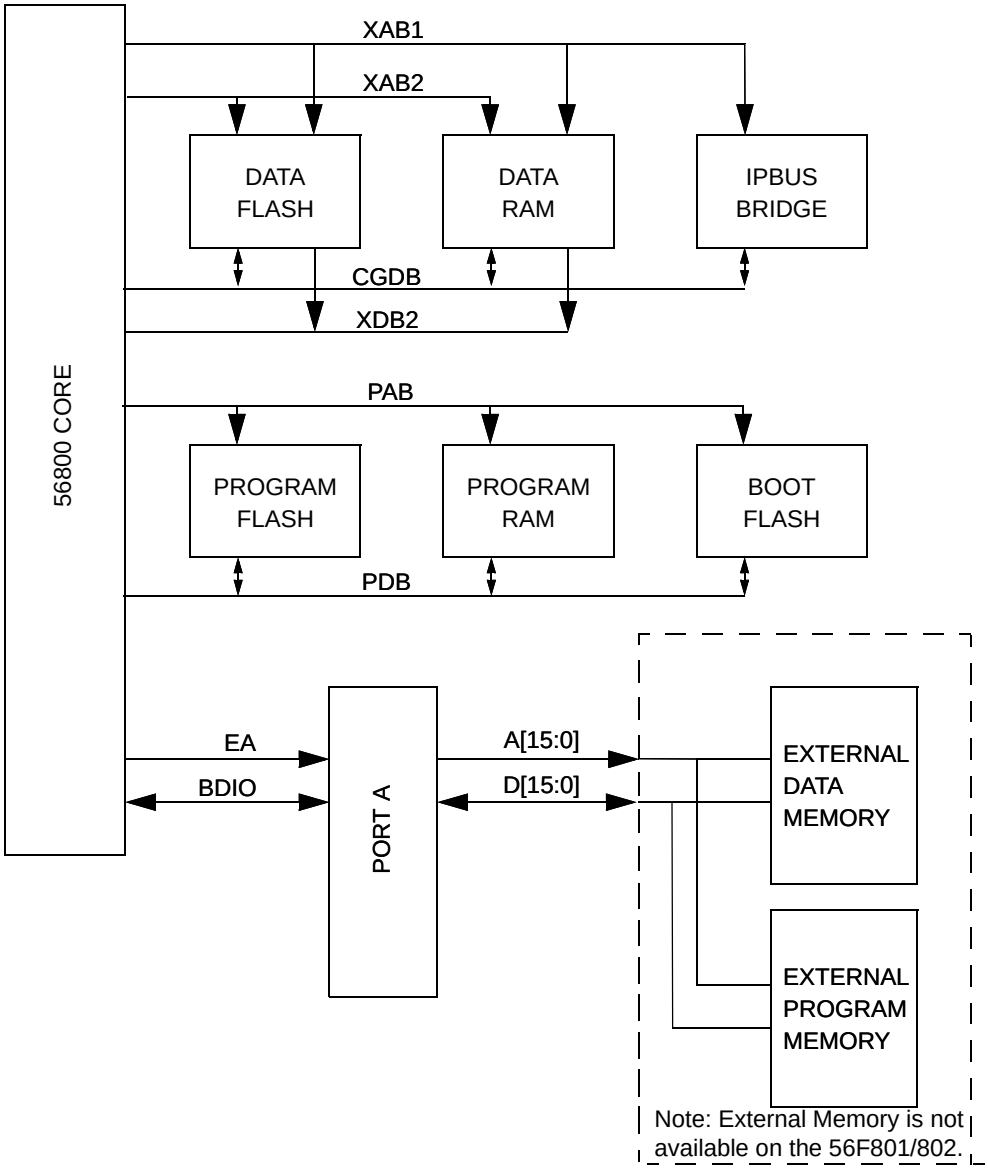


Figure 3-3. 56F80x On-Board Address and Data Buses

Note: Data memory is often noted as X-memory. For example, data RAM is called XRAM.

The following data space reads are possible:

- Dual read XRAM using CGDB and XRAM using XDB2
- Dual read RAM using CGDB and XFLASH using XDB2
- Dual read XRAM using XDB2 and XFLASH using CGDB
- Single read XRAM using CGDB
- Single read XFLASH using CGDB

Note: Dual read XFLASH using XDB2 and XFLASH using CGDB *is NOT allowed.* Dual reads solely from XFLASH *are NOT allowed*. CGDB reads and writes are a result of XAB1 addressing. XDB2 reads are a result of XAB2 addressing. *Writes are NOT allowed*. XAB2 addressing is always the second field of a dual *move* operation, and can only be sourced by the R3 register.

The XRAM is a true dual-port RAM. The XFLASH is not available as a dual-ported memory.

Note: Due to the IPBus methodology used for busing from core to peripherals, the 56F80x does not connect the PGDB from the controller core to the peripherals. Certain addressing modes of the MOVE command may very likely cause a transfer to happen on the core's Peripheral Global Data Bus (PGDB). The 56F80x does not connect the PGDB from the controller core to the peripherals, so instructions such as MOVEP, utilizing this data bus, are not useful. The MOVEP instruction only functions for registers internal to the core: OnCE PGDB Bus Transfer Register (OPGDBR), Interrupt Priority Register (IPR) and Bus Control Register (BCR).

Table 3-42. Document Revision History for Chapter 3

Version History	Description of Change
Rev. 8	Formatting, layout, spelling, and grammar corrections. Added revision history table.

Chapter 4
Interrupt Controller (ITCN)

4.1 Introduction

The IPBus Interrupt Controller (ITCN) accepts Interrupt Request (IRQ) signals for Interrupt Priority (IP) bus-based peripherals, assigns user-defined levels to each IRQ, and then selects the highest pending IRQ for presentation to the 56800 core. The 56800 core includes circuitry supports with seven levels of interrupts. The 56800 provides priority to the interrupts associated with the highest level.

The ITCN augments the 56800 interrupt controller by expanding the maximum number of peripheral interrupts to 64 and adding the ability to assign each of the 64 interrupt request sources to one of seven levels. Within any one of the seven levels, any number of interrupt sources may be assigned. For additional information, refer to **Figure 7-2** in the *DSP56800 Family Manual*.

4.2 Interrupt Source

Each Interrupt Source has a fixed vector number regardless of the level it has been assigned. For any given level, its vector number determines an interrupt priority. The interrupt source with the highest vector has the highest priority within a given level.

4.3 Interrupt Control

The Interrupt Controller module does not mask, generate, or clear IRQs. The Peripheral module issuing the IRQ defines the enabling and clearing methods of a particular interrupt. The Interrupt Controller provides only the priority assignments and the interrupt vector generation.

4.4 Priority Level Register (PLR)

There are 64 three-bit Priority Level Registers (PLRs) specifying the level assignment, one to seven, for each interrupt request signal. Four PLRs are assembled together in each Group Priority Register (GPR). Each 16-bit GPR is capable of being read and written. Each PLR value has a default value on reset suiting the application’s requirements. Each value may be changed. If a PLR value is set to zero, the effect is to disable that interrupt.

4.5 Interrupt Exceptions

The I1 and I0 bits in the 56800 core Status Register (SR) determine the class of the permitted exceptions. To permit mask exceptions, I bits should be set to 01. To disable the mask exceptions, I bits should be set to 11. The Interrupt Priority Register (IPR), also part of the 56800 core, is located at its X:$FFFB address. The IPR has an Interrupt Priority Level (IPL) bit assigned to each of the seven Interrupt Priority Levels generated by the Interrupt Controller. Each IPL bit enables interrupts associated with its Interrupt Priority Level.

4.6 Interrupt Enable

To enable a specific interrupt, the bit in the peripheral must be set, the PLR in the Interrupt Controller must be initialized and the corresponding IPL bits must be set in the processor status register.

The ITCN is designed for 64 interrupts; however, the 56F80x reserves the first 10 interrupt vectors for internally generated exceptions and the $\overline{\text{IRQA}}$ and $\overline{\text{IRQB}}$ external interrupts.

4.7 Interrupt Priority Register (IPR)

The core's Interrupt Priority Register (IPR) is a read/write memory-mapped register located at X:$FFFB. The IPR specifies the IPL for each of the interrupting devices, including the $\overline{\text{IRQA}}$ and $\overline{\text{IRQB}}$ pins, as well as each on-chip peripheral capable of generating interrupts. The interrupt arbiter on the 56800 core contains seven interrupt channels for peripherals' use in addition to the $\overline{\text{IRQ}}$ interrupts and the interrupts provided by the core. The IPL for all on-chip peripheral interrupt sources are interrupt channels zero–six. They are assigned by the customer. Using the IPR, each of these channels may be individually enabled or disabled under software control. Additionally, the IPR specifies the Trigger mode of each external interrupt source and can enable or disable the individual external interrupts. The IPR is cleared on Hardware Reset.

Peripheral Interrupts are enabled or masked by writing to the IPR after enabling them in the Status Register (SR). **Figure 4-1** illustrates the interrupt priority order and how IPR bits are programmed. **Figure 4-3** discloses interrupt programming. Unused bits are read as zero and should be written with 0, ensuring future compatibility.

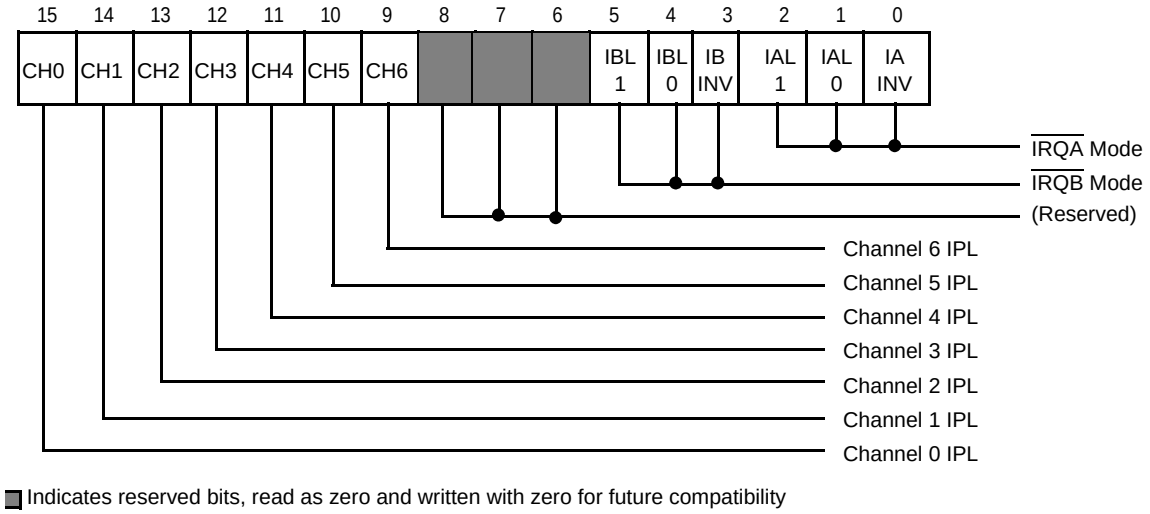


Figure 4-1. IPR Programming Model

Table 4-1. Interrupt Programming

IBL1 IAL1	IBINV IAINV	Trigger Mode
0	0	Low-Level Sensitive
0	1	High-Level Sensitive
1	0	Falling-Edge Sensitive
1	1	Rising-Edge Sensitive

IBL0 IAL0	Enabled?	IPL
0	No	—
1	Yes	0

CH0 CH1	Enabled?	IPL
0	No	—
1	Yes	0

Note: To avoid spurious interrupts, it may be necessary to disable $\overline{\text{IRQx}}$ interrupts, by clearing the IxL0 bit, before modifying IxL1 or IxINV.

4.8 ITCN Register Summary

The interrupt controller has the following registers:

- IPR core at X:$FFFB. For additional information, please refer to **Figure 7-2** in the *DSP56800 Family Manual*
- SR core program controller unit
- GPR 2–15 contain 3-bit PLRs for interrupt sources 10–63
- Test Control and Status Register (TCSR)
- Test Interrupt Request 0 (TIRQ0) register
- Test Interrupt Request 1 (TIRQ1) register

- Test Interrupt Request 2 (TIRQ2) register
- Test Interrupt Request 3 (TIRQ3) register
- Test Interrupt Source Register 0 (TISR0)
- Test Interrupt Source Register 1 (TISR1)
- Test Interrupt Source Register 2 (TISR2)
- Test Interrupt Source Register 3 (TISR3)

For Interrupt Controller registers of the address map, please refer to **Table 3-23.**

4.9 Priority Level and Vector Assignments

Table 4-2 indicates the vector for each interrupt source. For a selected level, the highest vector has the highest priority.

Table 4-2. Interrupt Vectors and Addresses

801	802	803/ 805	807	Vector	IRQ Table Address	Interrupt Source Description
x	x	x	x	63	$007E	Low Voltage Detector
x	x	x	x	62	$007C	PLL Loss of Lock/Clock
x	x	x	x	61	$007A	PWM A Fault
—	—	x	x	60	$0078	PWM B Fault
x	x	x	x	59	$0076	Reload PWM A
—	—	x	x	58	$0074	Reload PWM B
x	x	x	x	57	$0072	ADC A Zero Crossing or Limit Error
—	—	—	x	56	$0070	ADC B Zero Crossing or Limit Error
x	x	x	x	55	$006E	ADC A Conversion Complete
—	—	—	x	54	$006C	ADC B Conversion Complete
x	x	x	x	53	$006A	SCI #0 Receiver Full
x	x	x	x	52	$0068	SCI #0 Receiver Error
x	x	x	x	51	$0066	SCI #0 Transmitter Ready
x	x	x	x	50	$0064	SCI #0 Transmit Complete
—	—	x	x	49	$0062	SCI #1 Receiver Full
—	—	x	x	48	$0060	SCI #1 Receiver Error
—	—	x	x	47	$005E	SCI #1 Transmitter Ready

Table 4-2. Interrupt Vectors and Addresses (Continued)

801	802	803/ 805	807	Vector	IRQ Table Address	Interrupt Source Description
—	—	x	x	46	$005C	SCI #1 Transmitter Complete
—	—	x	x	45	$005A	Timer A Channel 3
—	—	x	x	44	$0058	Timer A Channel 2
—	—	x	x	43	$0056	Timer A Channel 1
—	—	x	x	42	$0054	Timer A Channel 0
—	—	x	x	41	$0052	Timer B Channel 3
—	—	x	x	40	$0050	Timer B Channel 2
—	—	x	x	39	$004E	Timer B Channel 1
—	—	x	x	38	$004C	Timer B Channel 0
x	x	x	x	37	$004A	Timer C Channel 3
x	x	x	x	36	$0048	Timer C Channel 2
x	x	x	x	35	$0046	Timer C Channel 1
x	x	x	x	34	$0044	Timer C Channel 0
x	x	x	x	33	$0042	Timer D Channel 3
x	x	x	x	32	$0040	Timer D Channel 2
x	x	x	x	31	$003E	Timer D Channel 1
x	x	x	x	30	$003C	Timer D Channel 0
—	—	x	x	29	$003A	Quadrature Decoder #0 INDEX Pulse
—	—	x	x	28	$0038	Quadrature Decoder #0 Home Switch or Watchdog
—	—	x	x	27	$0036	Quadrature Decoder #1 INDEX Pulse
—	—	x	x	26	$0034	Quadrature Decoder #1 Home Switch or Watchdog
x	—	x	x	25	$0032	SPI Receiver Full and/or Error
x	—	x	x	24	$0030	SPI Transmitter Empty
x	x	x	x	23	$002E	GPIO A
x	x	x	x	22	$002C	GPIO B
—	—	x	x	21	$002A	Reserved
—	—	x	x	20	$0028	GPIO D

Table 4-2. Interrupt Vectors and Addresses (Continued)

801	802	803/ 805	807	Vector	IRQ Table Address	Interrupt Source Description
—	—	x	x	19	$0026	GPIO E
—	—	—	x	18	$0024	Program Flash Interface 2
—	—	x	x	17	$0022	CAN Wakeup
—	—	x	x	16	$0020	CAN Error
—	—	x	x	15	$001E	CAN Receiver Full
—	—	x	x	14	$001C	CAN Transmitter Ready
x	x	x	x	13	$001A	Data Flash Interface
x	x	x	x	12	$0018	Program Flash Interface
x	x	x	x	11	$0016	Boot Flash Interface
—	—	—	—	10	$0014	Reserved
All vectors listed above this point in the table are controlled by the ITCN module						
All vectors listed below this point in the table are controlled directly by the 56800 Core						
—	—	x	x	9	$0012	$\overline{\text{IRQB}}$
x	—	x	x	8	$0010	$\overline{\text{IRQA}}$
—	—	—	—	7	$000E	Reserved
x	x	x	x	6	$000C	OnCE Trap
x	x	x	x	5	$000A	HW Stack Overflow
x	x	x	x	4	$0008	SWI
x	x	x	x	3	$0006	Illegal Instruction
—	—	—	—	2	$0004	Reserved
x	x	x	x	1	$0002	COP/Watchdog Reset
x	x	x	x	0	$0000	Hardware Reset

4.10 Register Definitions

Table 4-3. ITCN Memory Map

Device	Peripheral	Address
801/802/ 803/805	ITCN_BASE	$0E60
807	ITCN_BASE	$1260

The address of a register is the sum of a base address and an address offset. The base address is defined as ITCN_BASE and the address offset is defined for each register below. See **Table 3-23** for the ITCN_BASE definition.

Table 4-4. ITCN Register Summary

Address Offset	Register Acronym	Register Name	Access Type	Register Location
Base + $2	ITCN_GPR2	Group Priority Register 2	Read/Write	Section 4.10.1 "Group Priority Registers 2–15 (GPR2–GPR15)"
Base + $3	ITCN_GPR3	Group Priority Register 3	Read/Write	
Base + $4	ITCN_GPR4	Group Priority Register 4	Read/Write	
Base + $5	ITCN_GPR5	Group Priority Register 5	Read/Write	
Base + $6	ITCN_GPR6	Group Priority Register 6	Read/Write	
Base + $7	ITCN_GPR7	Group Priority Register 7	Read/Write	
Base + $8	ITCN_GPR8	Group Priority Register 8	Read/Write	
Base + $9	ITCN_GPR9	Group Priority Register 9	Read/Write	
Base + $A	ITCN_GPR10	Group Priority Register 10	Read/Write	
Base + $B	ITCN_GPR11	Group Priority Register 11	Read/Write	
Base + $C	ITCN_GPR12	Group Priority Register 12	Read/Write	
Base + $D	ITCN_GPR13	Group Priority Register 13	Read/Write	
Base + $E	ITCN_GPR14	Group Priority Register 14	Read/Write	
Base + $F	ITCN_GPR15	Group Priority Register 15	Read/Write	

Bit fields of the 14 registers are summarized in **Figure 4-2.** Details of each follow.

Add. Offset	Register Name		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$2	GPR2	R		PLR11				PLR10										
		W																
$3	GPR3	R		PLR15				PLR14				PLR13				PLR12		
		W																
$4	GPR4	R		PLR19				PLR18				PLR17				PLR16		
		W																
$5	GPR5	R		PLR23				PLR22				PLR21				PLR20		
		W																
$6	GPR6	R		PLR27				PLR26				PLR25				PLR24		
		W																
$7	GPR7	R		PLR31				PLR30				PLR29				PLR28		
		W																
$8	GPR8	R		PLR35				PLR34				PLR33				PLR32		
		W																
$9	GPR9	R		PLR39				PLR38				PLR37				PLR36		
		W																
$A	GPR10	R		PLR43				PLR42				PLR41				PLR40		
$B	GPR11	R		PLR47				PLR46				PLR45				PLR44		
		W																
$C	GPR12	R		PLR51				PLR50				PLR49				PLR48		
		W																
$D	GPR13	R		PLR55				PLR54				PLR53				PLR52		
		W																
$E	GPR14	R		PLR59				PLR58				PLR57				PLR56		
		W																
$F	GPR15	R		PLR63				PLR62				PLR61				PLR60		
		W																

Figure 4-2. ITCN Register Map Summary

4.10.1 Group Priority Registers 2–15 (GPR2–GPR15)

Each interrupt source has an associated 3-bit Priority Level Register (PLR) defining its priority level. The PLRs are read/write, and are packed together in groups of four per 16-bit register, except the GPR2.

GPR2 16-bit registers are called Group Priority Registers (GPR). Level seven is the highest priority, while level one is the lowest priority. Within each priority level, the highest vector has the highest priority. For more details, refer to **Table 4-2.**

Note: In a Group Priority Register (GPR), a PLR with a value of one corresponds to channel zero in the interrupt priority register. A PLR with a value of seven corresponds to Interrupt Priority register channel six. If the PLR value is set to zero, the effect will disable the interrupt. PLR0–PLR10 are reserved.

Note: The first two factory test registers are not illustrated here.

ITCN_BASE+$2	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read		PLR11				PLR10										
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 4-3. Group Priority Register 2 (GPR2)

See Table A-8, List of Programmer's Sheets

ITCN_BASE+$3	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read		PLR15				PLR14				PLR13				PLR12		
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 4-4. Group Priority Register 3 (GPR3)

See Table A-8, List of Programmer's Sheets

ITCN_BASE+$4	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read		PLR19				PLR18				PLR17				PLR16		
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 4-5. Group Priority Register 4 (GPR4)

See Table A-8, List of Programmer's Sheets

ITCN_BASE+$5	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read		PLR23				PLR22				PLR21				PLR20		
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 4-6. Group Priority Register 5 (GPR5)

See Table A-8, List of Programmer's Sheets

ITCN_BASE+$6	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read		PLR27				PLR26				PLR25				PLR24		
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 4-7. Group Priority Register 6 (GPR6)

See Table A-8, List of Programmer's Sheets

ITCN_BASE+$7	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read		PLR31				PLR30				PLR29				PLR28		
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 4-8. Group Priority Register 7 (GPR7)

See Table A-8, List of Programmer's Sheets

ITCN_BASE+$8	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read		PLR35				PLR34				PLR33				PLR32		
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 4-9. Group Priority Register 8 (GRP8)

See Table A-8, List of Programmer's Sheets

ITCN_BASE+$9	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read		PLR39				PLR38				PLR37				PLR36		
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 4-10. Group Priority Register 9 (GRP9)

See Table A-8, List of Programmer's Sheets

ITCN_BASE+$A	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read		PLR43				PLR42				PLR41				PLR40		
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 4-11. Group Priority Register 10 (GPR10)

See Table A-8, List of Programmer's Sheets

ITCN_BASE+$B	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read		PLR47				PLR46				PLR45				PLR44		
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 4-12. Group Priority Register 11 (GPR11)

See Table A-8, List of Programmer's Sheets

ITCN_BASE+$C	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read		PLR51				PLR50				PLR49				PLR48		
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 4-13. Group Priority Register 12 (GPR12)

See Table A-8, List of Programmer's Sheets

ITCN_BASE+$D	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read		PLR55				PLR54				PLR53				PLR52		
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 4-14. Group Priority Register 13 (GPR13)

See Table A-8, List of Programmer's Sheets

ITCN_BASE+$E	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read		PLR59				PLR58				PLR57				PLR56		
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 4-15. Group Priority Register 14 (GPR14)

See Table A-8, List of Programmer's Sheets

ITCN_BASE+$F	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read		PLR63				PLR62				PLR61				PLR60		
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 4-16. Group Priority Register 15 (GPR15)

See Table A-8, List of Programmer's Sheets

Table 4-5. Document Revision History for Chapter 4

Version History	Description of Change
Rev. 8	Formatting, layout, spelling, and grammar corrections. Added revision history table. Corrected register acronyms in the ITCN register map summary (were GRPn, are GPRn).

Chapter 5
Flash Memory Interface

5.1 Introduction

Flash Memory provides reliable, non-volatile memory storage, eliminating required external storage devices. Its software is easily programmable and can be booted directly from Flash.

5.2 Features

- Endurance: 10,000 cycles (typical)
- Memory organization for 56F801 and 56F802

	Main Block	Information Block
— Program Flash =	8K × 16	64 × 16
— Data Flash =	2K × 16	64 × 16
— Boot Flash =	2K × 16	64 × 16

- Memory organization for 56F803 and 56F805

	Main Block	Information Block
— Program Flash =	32K × 16	64 × 16
— Data Flash =	4K × 16	64 × 16
— Boot Flash =	2K × 16	64 × 16

- Memory organization for 56F807

	Main Block	Information Block
— Program Flash =	64K × 16	64 × 16
— Data Flash =	8K × 16	64 × 16
— Boot Flash =	2K × 16	64 × 16

- Page Erase capability: 512 bytes per page
- Block (Mass) Erase capability
- Access Time: 20ns (max)
- Word (16-bit) Program Time: 20μs (min)

- Page Erase Time: 40ms (min)
- Mass Erase Time: 100ms (min)

Note: Because the 56F807 has 64K Program Flash, two Program Flash Interface Units are required to reach all 64K memory locations. As a result, the 56F807 has a second set of PFIU registers beginning with memory location PFIU2_BASE, defined in **Section 3.11, "Memory Architecture".** The second Program Flash Interface Unit (PFIU2) functions the same as the original PFIU. For the 56F807, PFIU controls the first half of Program Flash and PFIU2 controls the last half of Program Flash.

5.3 Flash Description

The FLASH is a CMOS, page erasable 16-bit word programmable embedded memory. It is partitioned into two memory blocks:

1. Main Memory Block
2. Information Block

The Main Memory Block is used for storage of application code and data. The Information Block can be used for the storage of device information. The Active block is selected by the IFREN bit in the FIU_CNTL register. Program and erase operations are performed on the Active block through the associated Flash Interface Unit (FIU).

The Flash Erase and programming voltages are regulated from the extreme 3.3V supply within the device. No special erase and programming voltages are required. The Page Erase operation erases all bytes within a page. The Mass Erase operation erases all bytes within the Flash block. The program operation may be performed on a single word or multiple words up to one row.

Table 5-1. Truth Table

Mode	XE	YE	SE	OE	PROG	ERASE	MAS1	NVSTR
Standby	L	L	L	L	L	L	L	L
Read	H	H	H	H	L	L	L	L
Word Program	H	H	L	L	H	L	L	H
Page Erase	H	L	L	L	L	H	L	H
Mass Erase	H	L	L	L	L	H	H	H

Table 5-2. Information Row Enable Truth Table

Mode	IFREN = 1	IFREN = 0
Read	Read Information Block	Read Main Memory Block
Word Program	Program Information Block	Program Main Memory Block
Page Erase	Erase Information Block	Erase Main Memory Block
Mass Erase	Erase Both Block	Erase Main Memory Block

Table 5-3. Flash Timing Variables

Parameter	Symbol
X Address Access Time	t_{XA}
Y Address Access Time	t_{YA}
OE Access Time	t_{OA}
PROG/ERASE to NVSTR Setup Time	t_{NVS}
NVSTR Hold Time	t_{NVH}
NVSTR Hold Time (Mass Erase)	t_{NVHL}
NVSTR to Program Setup Time	t_{PGS}
Program Hold Time	t_{PGH}
Program Time	t_{PROG}
Address/Data Setup Time	t_{ADS}
Address/Data Hold Time	t_{ADH}
Recovery Time	t_{RCV}
Cumulative Program HV Period	t_{HV}
Erase Time	t_{ERASE}
Mass Erase Time	t_{ME}

5.4 Program Flash (PFLASH)

Devices described in this section use Flash blocks as non-volatile memory. The Flash block is instantiated within the Program Address Space (PFLASH). PFLASH is connected to core buses via a Program Flash Interface.

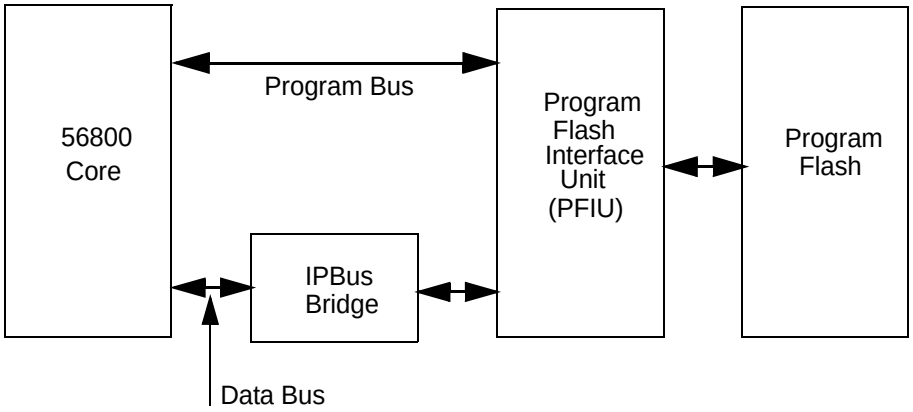


Figure 5-1. Program Flash Block Integration

PFLASH configurations for the devices described here are presented in **Table 5-4.**

Table 5-4. Program Flash Main Block Organization

Chip	Total Size	Row Size	# of Rows	Page Size	# of Pages
56F801	8K x 16 bits	512 bits (32 words)	256	256 words (8 rows)	32
56F802	8K x 16 bits	512 bits (32 words)	256	256 words (8 rows)	32
56F803	32K x 16 bits[1]	512 bits (32 words)	1024	256 words (8 rows)	128
56F805	32K x 16 bits[1]	512 bits (32 words)	1024	256 words (8 rows)	128
56F807	64K x 16 bits[2]	512 bits (32 words)	2048	256 words (8 rows)	256

1. Use up to 32252 words, the balance are reserved.
2. Use up to 60K words, the balance are reserved.

In addition to the Main Block sizes shown in the above table, each PFLASH contains a two-row Information Block.

Table 5-5. Program Flash Information Block Organization

Chip	Total Size	Row Size	# of Rows	Page Size	# of Pages
56F801	64 x 16 bits	512 bits (32 words)	2	64 words	1
56F802	64 x 16 bits	512 bits (32 words)	2	64 words	1
56F803	64 x 16 bits	512 bits (32 words)	2	64 words	1
56F805	64 x 16 bits	512 bits (32 words)	2	64 words	1
56F807	64 x 16 bits	512 bits (32 words)	2	64 words	1

5.5 Data Flash (DFLASH)

The devices described in this section use Flash blocks as non-volatile memory. This section contains a description of the Flash block instantiated within the Data Address Space (DFLASH). The DFLASH is connected to the buses via a Data Flash Interface.

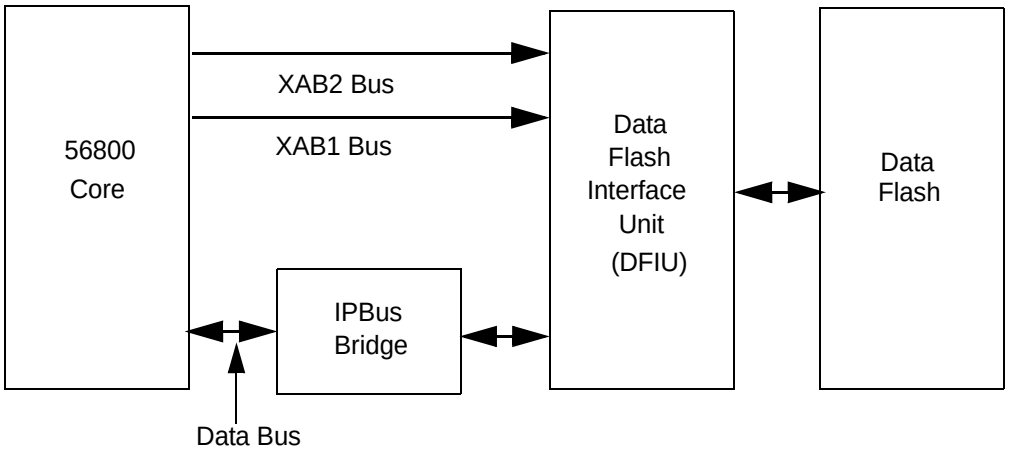


Figure 5-2. Data Flash Block Integration

DFLASH configurations for the devices are described in **Table 5-6.**

Table 5-6. Data Flash Main Block Organization

Chip	Total Size	Row Size	# of Rows	Page Size	# of Pages
56F801	2K x 16 bits	512 bits (32 words)	64	256 words (8 rows)	8
56F802	2K x 16 bits	512 bits (32 words)	256	256 words (8 rows)	32
56F803	4K x 16 bits	512 bits (32 words)	128	256 words (8 rows)	16
56F805	4K x 16 bits	512 bits (32 words)	128	256 words (8 rows)	16
56F807	8K x 16 bits	512 bits (32 words)	256	256 words (8 rows)	32

In addition to the Main Block sizes shown in the table above, each DFLASH contains a two-row Information Block.

Table 5-7. Program Flash Information Block Organization

Chip	Total Size	Row Size	# of Rows	Page Size	# of Pages
56F801	64 x 16 bits	512 bits (32 words)	2	64 words	1
56F802	64 x 16 bits	512 bits (32 words)	2	64 words	1
56F803	64 x 16 bits	512 bits (32 words)	2	64 words	1
56F805	64 x 16 bits	512 bits (32 words)	2	64 words	1
56F807	64 x 16 bits	512 bits (32 words)	2	64 words	1

5.6 Boot Flash (BFLASH)

Devices described within this section use Flash blocks as non-volatile memory. The Flash Block is instantiated within the Boot Address Space (BFLASH). BFLASH is connected to the core buses via a Boot Flash Interface.

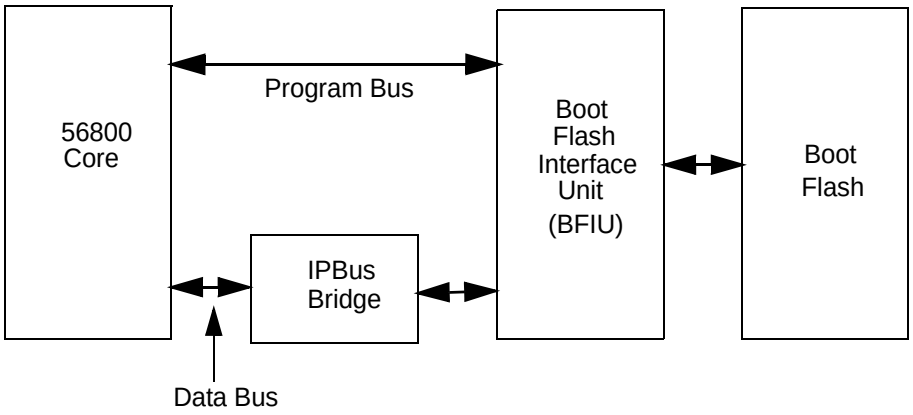


Figure 5-3. Boot Flash Block Integration

BFLASH configurations for the devices are described in **Table 5-8.**

Table 5-8. Boot Flash Main Block Organization

Chip	Total Size	Row Size	# of Rows	Page Size	# of Pages
56F801	2K x 16 bits	512 bits (32 words)	64	256 words (8 rows)	8
56F802	2K x 16 bits	512 bits (32 words)	256	256 words (8 rows)	32
56F803	4K x 16 bits	512 bits (32 words)	128	256 words (8 rows)	16
56F805	4K x 16 bits	512 bits (32 words)	128	256 words (8 rows)	16
56F807	8K x 16 bits	512 bits (32 words)	256	256 words (8 rows)	32

In addition to the Main Block sizes shown in the above table, each BFLASH contains a two-row Information Block.

Table 5-9. Boot Flash Information Block Organization

Chip	Total Size	Row Size	# of Rows	Page Size	# of Pages
56F801	64 x 16 bits	512 bits (32 words)	2	64 words	1
56F802	64 x 16 bits	512 bits (32 words)	2	64 words	1
56F803	64 x 16 bits	512 bits (32 words)	2	64 words	1
56F805	64 x 16 bits	512 bits (32 words)	2	64 words	1
56F807	64 x 16 bits	512 bits (32 words)	2	64 words	1

5.7 Program/Data/Boot Flash Interface Unit Features

This section details Program, Data, and Boot Flash Interface Unit features:

- Single port memory compatible with core pipelined program bus structure
- Single cycle reads at 40MHz across the automotive temperature range
- Word (16-bit) readable and programmable
- Supports memory organization defined in **Table 3-1**
- Page erase and Mass Erase modes
- Can be programmed under software control in the developer's system

5.8 Program/Data/Boot Flash Modes

By changing any of the following register, the Information Block may be swapped for the Main Memory Block.

- IFREN bit of PFIU_CNTL
- IFREN bit of DFIU_CNTL
- IFREN bit of BFIU_CNTL

Modes of operation include:

- Standby
- Read word
- Simple single word program operation
- Multiword programming (up to one row) possible
- Page Erase
- Mass Erase
- The PFLASH, DFLASH or BFLASH is automatically put in Standby mode when not being read, programmed, or erased
- Reads are disabled during program/erase operations
- Interrupts are individually maskable
- An interrupt will be generated for:
  - Reads out of range (trying to access a row past the first two in the Information Block)
  - Read attempted during program
  - Read attempted during erase

— Any of seven internal timer timeouts used during program/erase operations

- Intelligent erase and word programming features

5.9 Functional Description of the PFIU, DFIU, and BFIU

- The PFIU, DFIU, and BFIU have internal timers for:
  — t_{NVS} representing PROG/ERASE to NVSTR Setup Time
  — t_{NVH} and t_{NVHL} representing NVSTR Hold Times
  — t_{PGS} representing NVSTR to Program Setup Time
  — t_{PROG} representing Program time
  — t_{REC} representing Recovery time
  — t_{ERASE} representing Erase time
  — t_{ME} representing Mass Erase time
- These are initialized upon reset to default values expected to work correctly for word programming and page erase operations. They may be overwritten by application code after reset if it is necessary to change operation based upon Flash characterization data.

Note: Please see the *56F800 Data Sheet(s) (DSP56F801-DSP56F807/D)* for Characterization Data Values.

- All read/program/erase functions are terminated by a system reset.
- The Flash will automatically be placed in Standby mode when not being accessed.
- During an Erase Cycle, all bits within the affected area are changed to one. Programming a bit results in it being changed to zero. Bits can only be changed back to one by erasing that section of memory again. *A bit cannot be programmed to one*.

5.10 Flash Programming and Erase Models

Please refer to figures in Section 5.3 for Flash timing relationships required in the following Intelligent Word, Dumb Word Programming, and Intelligent Erase Operation.

The recommended Flash Programming mode is the Intelligent Word Programming, described in **Section 5.10.1.** *Although the Dumb Word Programming is described in* **Section 5.10.2,** *it is not the preferred nor recommended method of programming the Flash.*

Flash *cannot be accessed directly* when programming or erasing is underway. The program code for changing the Program Flash contents must be executing from some other memory.

5.10.1 Intelligent Word Programming

Assuming the word in question has already been erased, a single 16-bit word may be programmed through the following method:

1. Start with the Flash in either Standby or Read modes. Read FIU_CNTL register to insure this is true. All bits should be clear with the exception of IFREN when programming the Information Block.
2. Enable programming by setting the Intelligent Program Enable (IPE) bit and row number in the FIU_PE register. To calculate the row number, use the following algorithm:
   - Target_Address AND $7FFF divided by $20 = ROW
   - Put differently, set the MSB of the target address to zero and right shift the result five places
3. Set the IE[8] bit in the FIU_IE register to enable to t_{RCV} interrupt if desired to flag completion of programming.
4. Write the desired value to the proper word in the Flash Memory Map. Note a single location in the Flash may map to different locations in the Memory Map based upon the mode selected on startup. The FIU will adjust accordingly. This write to the Flash Memory Map, while the IPE bit is set, will start the internal state machine to run the Flash through its programming paces.
5. Do not attempt to access the Flash again until the BUSY signal clears in the FIU_CNTL register, corresponding to the t_{RCV} interrupt when enabled.
6. When programming is completed, be certain to clear the IPE bit in the FIU_PE register.

The Intelligent Programming feature can program *only* one word at a time. Multiple words may be programmed by repeating the process, or by switching to the Dumb Word Programming method, subsequently outlined.

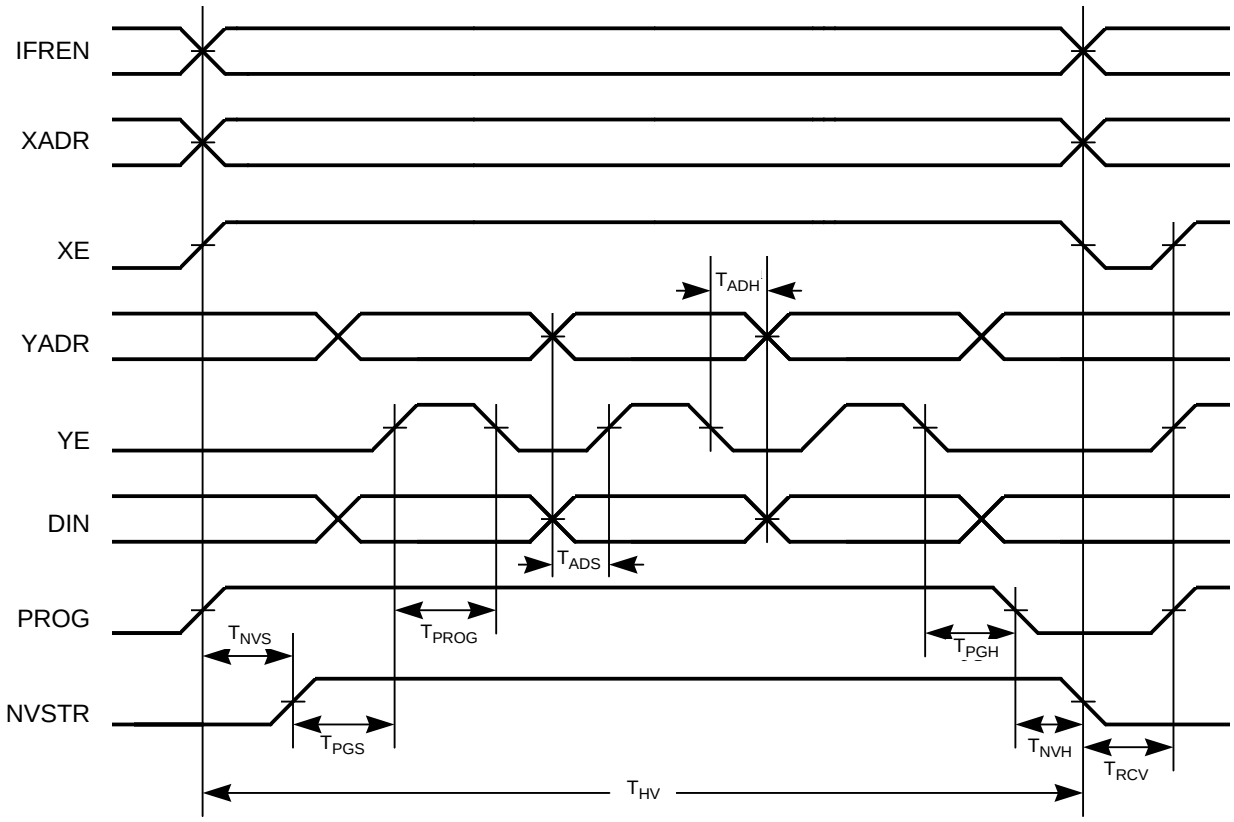


Figure 5-4. Flash Program Cycle

5.10.2 Dumb Word Programming

Flash allows programming of one or more words in a row. Assuming the words in question have already been erased, 16-bit words may be programmed using the following method:

1. Start with Flash in either Standby or Read modes. Read the BUSY bit in the FIU_CNTL register ensuring it is clear.
2. Enable programming by setting the DPE bit and row number in the FIU_PE register.
3. Write the desired value to the applicable word in the Flash Memory space assuring IS[0] is clear before proceeding.
4. Set the XE and PROG bits in the FIU_CNTL register.
5. Delay for t_{NVC}
6. Set the NVSTR bit in the FIU_CNTL register.
7. Delay for t_{PGHS}
8. Set the YE bit in the FIU_CNTL register.

9. Delay for t_{PROG}
10. Clear the YE bit in the FIU_CNTL register.
11. By writing to another location in the Flash Memory space, update the values in FIU_ADDR and FIU_DATA if additional words are required, loop back to step eight above. Address changes must be limited to within a single row making certain IS[0] is clear before proceeding. Attempts to change the FIU_CNTL register with IS[0] set will have no effect.
12. Clear the PROG bit in the FIU_CNTL register, creating a separate step from clearing YE.
13. Delay for t_{NVH}
14. Clear the NVSTR and XE bits in the FIU_CNTL register.
15. Delay for t_{RCV}

The FIU timer registers *cannot* be used to generate interrupts for each of the delays above. If interrupts are desired to generate the timing breakpoints for the dumb programming process, they must be generated from some other timing or interrupt source.

5.10.3 Intelligent Erase Operation

1. Start with the Flash in either Standby or Read modes. Read FIU_CNTL register ensuring this is true. All bits should be clear with the exception of IFREN when erasing the Information Block.
2. To flag completion of Erase, set the IE[8] bit in the FIU_IE register, enabling to t_{RCV} interrupt.
3. Enable Erase by setting the IEE bit and page number in the FIU_EE register. To calculate a page number, use the following algorithm:
   - Target_Address AND $7FFF divided by $100 = PAGE.
   - Put differently, set the MSB of the target address to zero and right shift the result eight places.
4. For a Mass Erase, set the page to $0. (Exception: when mass erasing boot Flash of 56F807, set the page to $78.) Also, for a mass erase, the MAS1 bit of the FIU_CNTL register must be set.
5. While the IEE bit is set, write any value to an address in the page to be erased. This write to the Flash Memory Map will start the FIU internal state machine, to run the Flash through its erase paces. Flash logic will erase the entire row, consequently it is not necessary to YADR[4.0] sequence within FIU.
6. Do not attempt to access Flash again until the BUSY signal clears in the FIU_CNTL register, corresponding to the t_{RCV} interrupt, when it is enabled.

7. Ensure FIU_CNTL and FIU_EE registers are cleared when finished.

Note: For information concerning YE, SE, OE appearing in the previous figures, refer to **Table 5-1.**

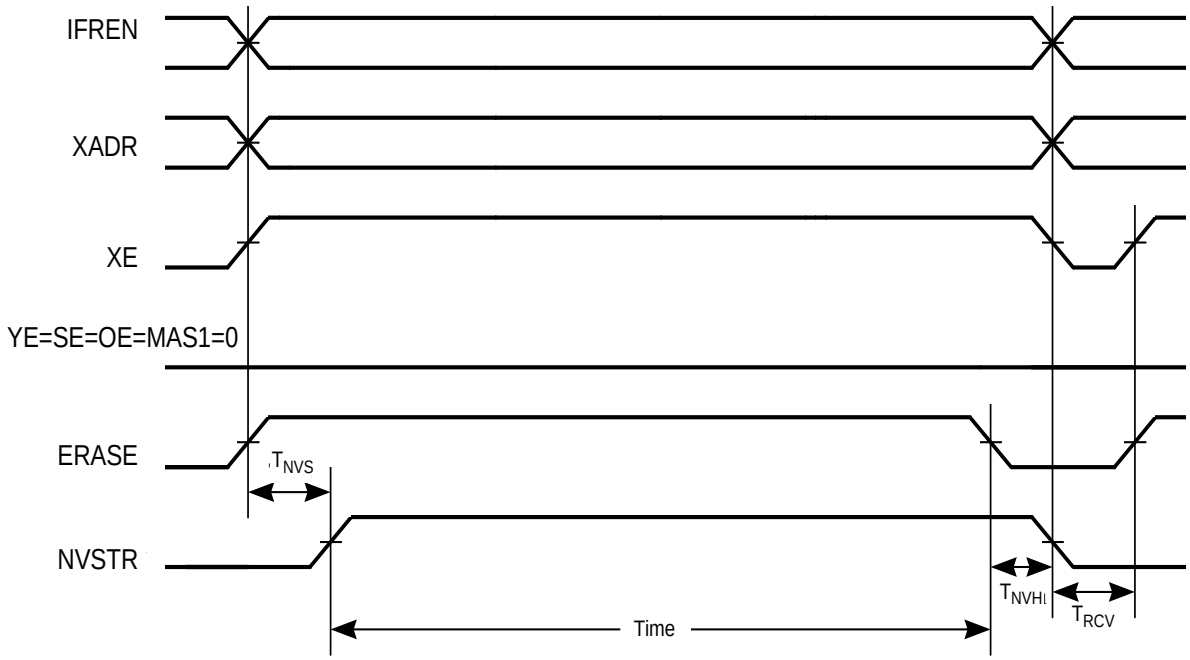


Figure 5-5. Flash Page Erase Cycle

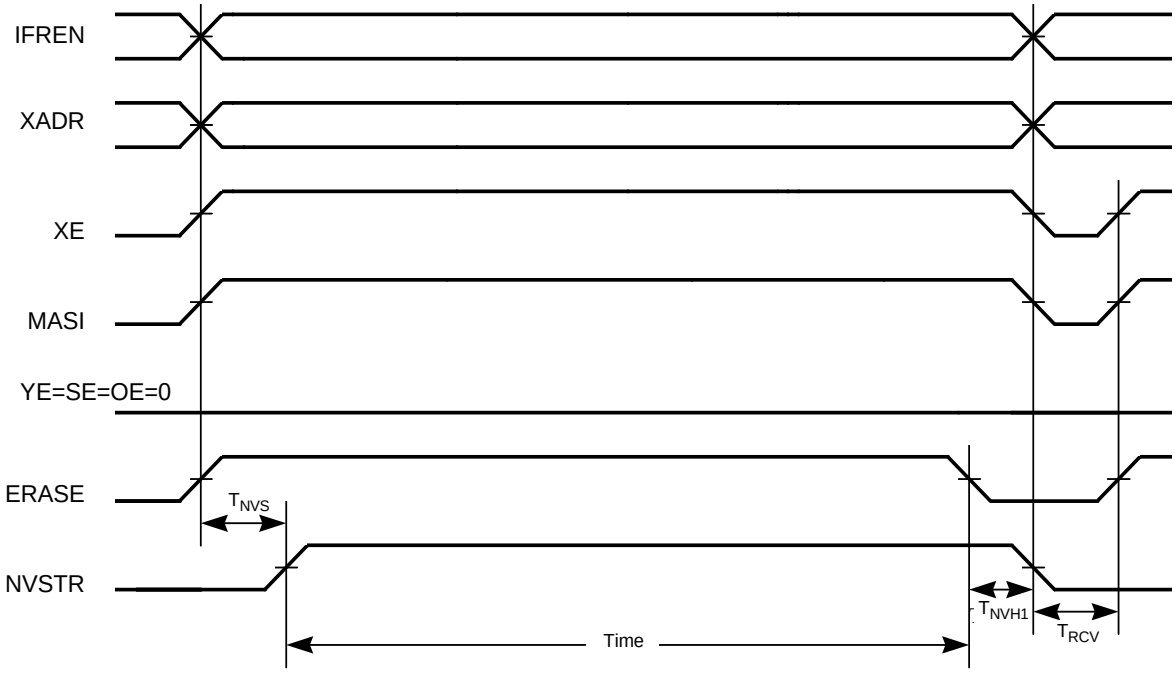


Figure 5-6. Flash Mass Erase Cycle

5.11 Register Definitions

Table 5-10. FLASH Memory Map

Device	Peripheral	Address
801/802/ 803/805	PFIU_BASE	$0F40
	DFIU_BASE	$0F60
	BFIU_BASE	$0F80
807	PFIU_BASE	$1340
	PFIU2_BASE	$1040
	DFIU_BASE	$1360
	BFIU_BASE	$1380

The Memory Map and Register Definitions are identical for Program Flash Interface Units (PFIU, PFIU2), Data Flash Interface Unit (DFIU), and Boot Flash Interface Unit (BFIU).

For each Flash Interface Unit (FIU) use the corresponding letters:

- *P* = Program
- *D* = Data
- *B* = Boot

One of the above letters must be included in front of FIU in *both the register name and base address to indicate the Flash type*. For example, the FIU_CNTL definition is same for PFIU_CNTL at memory location PFUI_BASE+0, DFIU_CNTL at memory location DFUI_BASE+0, and BFIU_CNTL at memory location BFUI_BASE+0.

For more information about PFIU registers, please refer to **Table 3-30.** Also, please refer to **Table 3-31** for DFIU registers, and **Table 3-32** for BFIU registers. For PFIU2 registers used *only* for the 56F807, please refer to **Table 3-13.**

The address of a register is the sum of a base address and an address offset. The base address is defined as the corresponding FIU_BASE and the address. Offset is defined for each register for the PFIU_BASE, PFIU2_BASE, DFIU_BASE, and BFIU_BASE definition in **Table 3-11.**

The interface also block-write protects all Flash registers except the active one. The Flash interface units are the same for Program Flash Interface Unit (PFIU). Since the 56F807 has 64K of Program Flash, 60K are usable while 4K is reserved, an extra Program Flash Interface Unit (PFIU2) is required to access the second half of Flash.

Table 5-11. Flash Register Summary

Address Offset	Register Acronym	Register Name	Access Type	Chapter Location
Base + $0	FIU_CNTL	Control Register	Read/Write	Section 5.11.1, “Flash Control Register (FIU_CNTL)”
Base + $1	FIU_PE	Program Enable Register	Read/Write	Section 5.11.2, “Flash Program Enable Register (FIU_PE)”
Base + $2	FIU_EE	Erase Enable Register	Read/Write	Section 5.11.3, “Flash Erase Enable Register (FIU_EE)”
Base + $3	FIU_ADDR	Address Register	Read/Write	Section 5.11.4, “Flash Address Register (FIU_ADDR)”
Base + $4	FIU_DATA	Data Register	Read/Write	Section 5.11.5, “Flash Data Register (FIU_DATA)”
Base + $5	FIU_IE	Interrupt Enable Register	Read/Write	Section 5.11.6, “Flash Interrupt Enable Register (FIU_IE)”

Table 5-11. Flash Register Summary (Continued)

Address Offset	Register Acronym	Register Name	Access Type	Chapter Location
Base + $6	FIU_IS	Interrupt Source Register	Read/Write	Section 5.11.7, “Flash Interrupt Source Register (FIU_IS)”
Base + $7	FIU_IP	Interrupt Pending Register	Read/Write	Section 5.11.8, “Flash Interrupt Pending Register (FIU_IP)”
Base + $8	FIU_CKDIVISOR	Clock Divisor Register	Read/Write	Section 5.11.9, “Flash Clock Divisor Register (FIU_CLKDIVISOR)”
Base + $9	FIU_TERASEL	TERASE Limit Register	Read/Write	Section 5.11.10, “Flash TERASE Limit Register (FIU_TERASEL)”
Base + $A	FIU_TMEL	TME Limit Register	Read/Write	Section 5.11.11, “Flash TME Limit Register (FIU_TMEL)”
Base + $B	FIU_TNVSL	TNVS Limit Register	Read/Write	Section 5.11.12, “Flash TNVS Limit Register (FIU_TNVSL)”
Base + $C	FIU_TPGSL	TPGS Limit Register	Read/Write	Section 5.11.13, “Flash TPGS Limit Register (FIU_TPGSL)”
Base + $D	FIU_TPROGL	TPROG Limit Register	Read/Write	Section 5.11.14, “Flash TPROG Limit Register (FIU_TPROGL)”
Base + $E	FIU_TNVHL	TNVH Limit Register	Read/Write	Section 5.11.15, “Flash TNVH Limit Register (FIU_TNVHL)”
Base + $F	FIU_TNVH1L	TNVH1L Limit Register	Read/Write	Section 5.11.16, “Flash TNVH1 Limit Register”
Base + $10	FIU_TRCVL	TRCV Limit Register	Read/Write	Section 5.11.17, “Flash TRCV Limit Register (FIU_TRCVL)”

Bit fields of each of the 17 registers are summarized in **Figure 5-7.** Details of each follow.

Add. Offset	Register Name		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$0	CNTL	R	BUSY	0	0	0	0	0	0	0	0	IFREN	XE	YE	PROG	ERASE	MAS1	NVSTR
		W																
$1	PE	R	DPE	IPE	0	0	0	0	ROW									
		W																
$2	EE	R	DEE	IEE	0	0	0	0	0	0	0	PAGE						
		W																
$3	ADDR	R	A															
		W																
$4	DATA	R	D]															
		W																
$5	IE	R	0	0	0	0	IE											
		W																
$6	IS	R	0	0	0	0	IS											
		W																
$7	IP	R	0	0	0	0	IP											
		W																
$8	CKDIVISOR	R	0	0	0	0	0	0	0	0	0	0	0	0	N			
		W																
$9	TERASEL	R	0	0	0	0	0	0	0	0	0	TERASEL						
		W																
$A	TMEL	R	0	0	0	0	0	0	0	0	TMEL							
		W																
$B	TNVSL	R	0	0	0	0	0	TNVSL										
		W																
$C	TPGSL	R	0	0	0	0	TPGSL											
		W																
$D	TPROGL	R	0	0	TPROGL													
		W																
$E	TNVHL	R	0	0	0	0	0	TNVHL										
		W																
$F	TNVH1L	R	0	TNVH1L														
		W																
$10	TRCVL	R	0	0	0	0	0	0	0	TRCVL								
		W																

R	0	Read as 0
W		Reserved

Figure 5-7. Flash Register Map Summary

5.11.1 Flash Control Register (FIU_CNTL)

When the BUSY bit is asserted in Intelligent Programming mode, *no register can receive writing*. In the Dumb Programming mode, *only* the address and data registers can be written.

FIU_BASE + $0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	BUSY	0	0	0	0	0	0	0	0	IFREN	XE	YE	PROG	ERASE	MAS1	NVSTR
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 5-8. Flash Control Register (FIU_CNTL)

See Programmer Sheet on Appendix page A-26

5.11.1.1 Busy (BUSY)—Bit 15

When the BUSY bit is asserted in Intelligent Programming mode, *no register can receive writing*. In the Dumb Programming mode, *only* the Address and Data registers can receive writing. The BUSY bit in the FIU_CNTL register will be set if any of the following are true:

- If PROG is asserted
- If NVSTR is asserted
- If ERASE is asserted

If the t_{RCV} counter is running Intelligent Program or Intelligent Erase modes, Flash reads should not be attempted while BUSY is asserted. The core should be prevented from entering Sleep or Low Power mode if BUSY is asserted OR FIU_IS is not equal to $0000. The BUSY bit is cleared once PROG, NVSTR, ERASE, XE, and YE bits are cleared.

5.11.1.2 Reserved—Bits 14–7

This bit field is reserved or not implemented. It can be read as 0 and cannot be modified by writing.

5.11.1.3 Information Block Enable (IFREN)—Bit 6

When IFREN is asserted, the Information Block of the Flash is enabled. The bit permits write in all modes when the BUSY bit is clear. The IFREN bit has the following effect. The IFREN bit has the following effect.

Table 5-12. IFREN Bit Effect

Activities	IFREN=1	IFREN=0
READ	Read Information Block	Read Main Memory Block
WORD PROGRAM	Program Information Block	Program Main Memory Block
PAGE ERASE	Erase Information Block	Erase Main Memory Block
MASS ERASE	Erase Both Blocks	Erase Main Memory Block

5.11.1.4 X Address Enable (XE)—Bit 5

This bit enables the X address when asserted. It can be written *only* in the Dumb Programming mode when the IS[0] bit is clear. The bit can be read *only* in the Intelligent Programming mode, reflecting the state of the Flash XE pin.

5.11.1.5 Y Address Enable (YE)—Bit 4

This bit enables the Y Address when asserted. It can be written *only* in Dumb Programming mode when the IS[0] bit is clear. The bit can be read *only* in Intelligent Programming mode, reflecting the state of the Flash YE pin.

5.11.1.6 Program Cycle Definition (PROG)—Bit 3

The Program Cycle is enabled when the PROG bit is asserted. It can be written *only* when the DPE bit is set and the IS[0] bit is clear. The bit can be read *only* in Intelligent Programming mode, reflecting the state of the Flash PROG pin. This bit *cannot* be set when either ERASE or MAS1 is set.

5.11.1.7 Erase Cycle Definition (ERASE)—Bit 2

The Erase Cycle is enabled when the ERASE bit is asserted. It can be written *only* when the DEE bit is set and the IS[0] bit is clear. The bit can be read *only* in Intelligent Programming mode, reflecting the state of the Flash ERASE pin. This bit *cannot* be set when PROG is set.

5.11.1.8 Mass Erase Cycle Definition (MAS1)—Bit 1

When the MAS1 bit is asserted, the Mass Erase Cycle is enabled, causing all pages to be automatically enabled. It can be written *only* when IEE or DEE is set and the BUSY bit is clear. The read value in the Intelligent Programming mode is a reflection of the state of the Flash MAS1 pin. It may not represent what was last written to the MAS1 bit. This bit *cannot* be set when PROG is set.

5.11.1.9 Non-Volatile Store Cycle Definition (NVSTR)—Bit 0

The Non-Volatile Store Cycle is enabled when the NVSTR bit is asserted. It can be modified *only* in Dumb Programming mode when the IS[0] bit is clear. The bit can be read *only* in the Intelligent Programming mode, reflecting the state of the Flash NVSTR pin.

5.11.2 Flash Program Enable Register (FIU_PE)

This register is reset upon any system reset. This register *cannot* be modified while the BUSY bit is asserted. IS[11] is asserted when this occurs.

FIU_BASE + $1	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	DPE	IPE	0	0	0	0	ROW									
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 5-9. Flash Program Enable Register (FIU_PE)

See Programmer Sheet on Appendix page A-27

5.11.2.1 Dumb Program Enable (DPE)—Bit 15

Dumb Programming is enabled when DPE is asserted. This bit *cannot* be set if any of IPE, DEE, or IEE bits are already active. Before DPE can be cleared, the following must be cleared: PROG, NVSTR, XE, and YE bits in the FIU_CNTL.

5.11.2.2 Intelligent Program Enable (IPE)—Bit 14

Intelligent Programming is enabled when IPE is asserted. This bit *cannot* be set if any of DPE, DEE, or IEE bits are already active.

5.11.2.3 Reserved—Bits 13–10

This bit field is reserved or not implemented. It is read as 0 and cannot be modified by writing.

5.11.2.4 Row Number (ROW)—Bits 9–0

ROW represents the number of the row to be programmed. The ROW number is calculated by forcing the MSB of the target address to zero, then right shift the result five places.

To program a row, the row number must be written as ROW[9:0]. A *write must* be performed to the same row to begin the programming sequence. Writing to a different row will *not* start the state machine, nor will it enable PROG and NVSTR. The Interrupt Source bit IS[0] is set to indicate an error. This is intended as a double check against accidental programming. These checks also apply for the Dumb Programming mode.

5.11.3 Flash Erase Enable Register (FIU_EE)

This register is reset upon any system reset. The register may *not* be modified while the BUSY bit is asserted. Should this happen, IS[11] is asserted.

FIU_BASE + $2	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	DEE	IEE	0	0	0	0	0	0	0	PAGE						
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 5-10. Flash Erase Enable Register (FIU_EE)

See Programmer Sheet on Appendix page A-27

5.11.3.1 Dumb Erase Enable (DEE)—Bit 15

Dumb Erase is enabled when the DEE bit is asserted. This bit *cannot* be set if any of DPE, IPE, or IEE bits are already active. ERASE, NVSTR, XE, and YE must be cleared before DEE can be cleared.

5.11.3.2 Intelligent Erase Enable (IEE)—Bit 14

Intelligent Erase is enabled when the IEE bit is asserted. This bit *cannot* be set if any of DPE, IPE, or DEE bits are already active.

5.11.3.3 Reserved—Bits 13–7

This bit field is reserved or not implemented. It is read as 0 and cannot be modified by writing.

5.11.3.4 Page Number (PAGE)—Bits 6–0

The PAGE bit must be set to the page number about to be erased. The page number is calculated by forcing the MSB of the target address to zero, then right shift the result eight places.

To erase a page, when MAS1 = 0 in FIU_CNTL, that page number must be written to PAGE. A *write must* be performed to the same page in order to begin the erase sequence. Writing to a different page will *not* start the intelligent erase state machine, nor will it enable ERASE and NVSTR. The Interrupt Source bit IS[0] is set to indicate an error. This is intended as a double check against accidental erasure.

Similarly, set PAGE to be $00 when MAS1 = 1. An exception is when mass erasing Boot Flash of 56F807. In that case, set the page to $78. These checks also apply for the Dumb Erase mode.

5.11.4 Flash Address Register (FIU_ADDR)

This register can be read as a register on the IPBus. It is cleared upon any system reset. This register is designed for program development and error analysis.

FIU_BASE + $3	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	A															
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 5-11. Flash Address Register (FIU_ADDR)

See Programmer Sheet on Appendix page A-28

This value of A[15:0] represents the current program or erase address. It can be read at any time. This register is set by attempting to write to memory space occupied by the Flash Memory.

When this write is enabled, the event can begin the Intelligent Program/Erase state machines through their paces. This register can *only* be set once at the start of an Intelligent Program or Intelligent Erase operation. Until the BUSY bit clears, subsequent writes have no effect on A[15:0], and an interrupt source (IS[1] or IS[2]) is set to indicate an error. In the Dumb Program or Erase modes, this register will be latched whenever FIU_CNTL bits PROG, ERASE, or NVSTR first go true and may *only* be changed after a negative edge of YE.

5.11.5 Flash Data Register (FIU_DATA)

This register is cleared upon any system reset. The register is designed for program development and error analysis.

FIU_BASE + $4	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	D															
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 5-12. Flash Data Register (FIU_DATA)

See Programmer Sheet on Appendix page A-28

Data bits (D[15:0]) reflect the last value programmed, or the value written to initiate an erase. FIU_ADDR and FIU_DATA are set simultaneously by writing to Flash Memory space. This value can be read at any time.

This register can only be set *once* at the start an Intelligent Program or Intelligent Erase operation. Until the BUSY bit clears, subsequent writes have no effect on D[15:0], and an interrupt source (IS[1] or IS[2]) will be set to indicate an error.

5.11.6 Flash Interrupt Enable Register (FIU_IE)

This register is reset upon any system reset. It may *only* be written when the BUSY bit is clear. IS[11] is asserted should this occur.

FIU_BASE + $5	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	IE											
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 5-13. Flash Interrupt Enable Register (FIU_IE)

See Programmer Sheet on Appendix page A-29

5.11.6.1 Reserved—Bits 15–12

This bit field is reserved or not implemented. It is read as 0 and cannot be modified by writing.

5.11.6.2 Interrupt Enable (IE)—Bits 11–0

Interrupt enables for IS[11:0] respectively. For example, if IE[3] is enabled, it can be activated when its *interrupt source* is triggered. It is bit wise and of IE[3] and IS[3] IP[11:0] = IE[11:0] AND IS[11:0]. To see a specific list of interrupt sources, refer to Section 5.11.7 or details about each bit.

5.11.7 Flash Interrupt Source Register (FIU_IS)

Under normal operation, *only* bits in FIU_IS can be cleared. The IS source bits can be cleared at any time. This register is reset upon any system reset.

FIU_BASE + $6	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	IS											
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 5-14. Flash Interrupt Source Register (FIU_IS)

See Programmer Sheet on Appendix page A-30

5.11.7.1 Reserved—Bits 15–12

This bit field is reserved or not implemented. It is read as 0 and cannot be modified by writing.

5.11.7.2 Interrupt Source (IS)—Bit 11

This Interrupt Source bit is asserted when a write to a register is attempted and while the BUSY bit is asserted.

5.11.7.3 Interrupt Source (IS)—Bit 10

This Interrupt Source bit is asserted when a write is attempted to FIU_CNTL during Intelligent Program or Intelligent Erase.

5.11.7.4 Interrupt Source (IS)—Bit 9

This Interrupt Source bit is asserted when there is a t_{ERASE} or t_{MEL} (Erase Time) timeout.

5.11.7.5 Interrupt Source (IS)—Bit 8

This Interrupt Source bit is asserted when there is a t_{RCV} (Recovery Time) timeout.

5.11.7.6 Interrupt Source (IS)—Bit 7

This Interrupt Source bit is asserted when there is a t_{PROG} (Program Time) timeout.

5.11.7.7 Interrupt Source (IS)—Bit 6

This Interrupt Source bit is asserted when there is a t_{PGS} (NVSTR to Program Setup Time) timeout.

5.11.7.8 Interrupt Source (IS)—Bit 5

This Interrupt Source bit is asserted when there is a t_{NVHL} (NVSTR Hold Time) timeout.

5.11.7.9 Interrupt Source (IS)—Bit 4

This Interrupt Source bit is asserted when there is a t_{NVH} (NVSTR Hold Time) timeout.

5.11.7.10 Interrupt Source (IS)—Bit 3

This Interrupt Source bit is asserted when there is a t_{NVS} (PROG/ERASE to NVSTR Setup Time) timeout.

5.11.7.11 Interrupt Source (IS)—Bit 2

This Interrupt Source bit is asserted when an illegal Flash read/write access is attempted during erase.

5.11.7.12 Interrupt Source (IS)—Bit 1

This Interrupt Source bit is asserted when an illegal Flash read/write access is attempted during program.

5.11.7.13 Interrupt Source (IS)—Bit 0

This Interrupt Source bit is asserted when a Flash Program or Flash Erase access out-of-range is attempted.

5.11.8 Flash Interrupt Pending Register (FIU_IP)

This register is read *only* and reset upon any system reset. Use this register for indirectly controlling the interrupt service routine.

FIU_BASE + $7	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	IP											
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 5-15. Flash Interrupt Pending Register (FIU_IP)

See Programmer Sheet on Appendix page A-31

5.11.8.1 Reserved—Bits 15–12

This bit field is reserved or not implemented. It is read as 0 and cannot be modified by writing.

5.11.8.2 Interrupt Pending (IP)—Bits 11–0

Flash Interface Unit is asserting an interrupt to the core when any of IP[11:0] are asserted. Interrupts are cleared by terminating the appropriate bit(s) in FIU_IS write 1 to bits not affected. IP[x] is a bit wise logical *and* operation of IE[x] and IS[x].

For IP[3] to be asserted, IE[3] has to be enabled *and* IS[3] must be asserted. This is because of the corresponding interrupt source being triggered. IP[11:0] = IE[11:0] *and* IS[11:0]. To see a specific list of interrupt sources, please refer to **Section 5.11.7** for details about each bit.

5.11.9 Flash Clock Divisor Register (FIU_CLKDIVISOR)

This register is reset during any system reset. It may *only* be changed when the BUSY bit is clear. IS[11] is asserted should this occur.

FIU_BASE+$8	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	0	0	0	0	0	0	0	N			
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

Figure 5-16. Flash Clock Divisor Register (FIU_CKDIVISOR)

See Programmer Sheet on Appendix page A-32

5.11.9.1 Reserved—Bits 15–4

This bit field is reserved or not implemented. It is read as 0 and cannot be modified by writing.

5.11.9.2 Clock Divisor (N)—Bits 3–0

The bus clock is divided by $2^{(N+1)}$ prior to clocking the t_{ERASE} and t_{ME} timers. The bus clock prescale value in ns is:

Bus Clock Period) $\times$ $2^{(N+1)}$
The Default Prescale Clock Value is $2^{16} \times 25ns = 16.384 \times 10^5$ ns

A Bus Clock Period of 25ns is typical. The Bus Clock Prescale Value in MHz is (bus clock frequency) divided by $2^{(N+1)}$. This prescale value is *only* used by the t_{ERASE} and t_{ME} timers.

5.11.10 Flash T_{ERASE} Limit Register (FIU_TERASEL)

This register is reset during any system reset. The register may *only* be changed when the BUSY bit is clear. IS[11] is asserted should this occur.

FIU_BASE + $9	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	0	0	0	TERASEL							
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

Figure 5-17. Flash T_{ERASE} Limit Register (FIU_TERASEL)

See Programmer Sheet on Appendix page A-32

5.11.10.1 Reserved—Bits 15–7

This bit field is reserved or not implemented. It is read as 0 and cannot be modified by writing.

5.11.10.2 Timer Erase Limit (TERASEL)—Bits 6–0

The bus clock is divided by $2^{(N+1)}$ prior to clocking the t_{ERASE} timer. TERASEL is the maximum erase time expressed as a multiple of this divided clock. Therefore, the maximum erase time is:

$2^{16} \times 2^7 \times 25ns = 210ms$
The default t_{ERASE} time = $2^{16} \times 2^4 \times 25ns = 26.2ms$

5.11.11 Flash T_{ME} Limit Register (FIU_TMEL)

This register is reset during any system reset. This register may *only* be changed when the BUSY bit is clear. IS[11] is asserted should this occur.

FIU_BASE + $A	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	0	0	0	TMEL							
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

Figure 5-18. Flash T_{ME} Limit Register (FIU_TMEL)

See Programmer Sheet on Appendix page A-32

5.11.11.1 Reserved—Bits 15–8

This bit field is reserved or not implemented. It is read as 0 and cannot be modified by writing.

5.11.11.2 Timer Mass Erase Limit (TMEL)—Bit 7–0

The bus clock is divided by $2^{(N+1)}$ prior to clocking the t_{ME} timer. TMEL is the maximum mass erase time expressed as a multiple of this divided clock. Therefore, the maximum erase time is:

$$2^{16} \times 2^{8} \times 25\text{ns} = 420\text{ms}$$

The default t_{ME} time = $2^{16} \times 2^{4} \times 25\text{ns} = 26.2\text{ms}$

5.11.12 Flash T_{NVS} Limit Register (FIU_TNVSL)

This register is reset during any system reset. It may *only* be changed when the BUSY bit is clear. IS[11] is asserted should this occur.

FIU_BASE +$B	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	TNVSL										
Write																
Reset	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Figure 5-19. Flash T_{NVS} Limit Register (FIU_TNVSL)

See Programmer Sheet on Appendix page A-33

5.11.12.1 Reserved—Bits 15–11

This bit field is reserved or not implemented. It is read as 0 and cannot be modified by writing.

5.11.12.2 Timer Non-Volatile Storage Limit (TNVSL)—Bits 10–0

Timeout value for the t_{NVS} counter in terms of bus clocks. This counter controls the PROG/ERASE to NVSTR Setup Time of the Flash. At a 40MHz bus clock, the maximum value of t_{NVS} is:

$$25ns \times 2^{11} = 51.2\mu s$$

The default value of t_{NVS} is $25ns \times 2^{8} = 6.4\mu s$

5.11.13 Flash T_{PGS} Limit Register (FIU_TPGSL)

This register is reset during any system reset. It may *only* be changed when the BUSY bit is clear. IS[11] is asserted should this occur.

FIU_BASE +$C	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	TPGSL											
Write																
Reset	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1

Figure 5-20. Flash T_{PGS} Limit Register (FIU_TPGSL)

See Programmer Sheet on Appendix page A-33

5.11.13.1 Reserved—Bits 15–12

This bit field is reserved or not implemented. It is read as 0 and cannot be modified by writing.

5.11.13.2 Timer Program Setup Limit (TPGSL)—Bits 11–0

Timeout value for the t_{PGS} counter in terms of bus clocks. This counter controls the NVSTR to Program Setup Time of the Flash. At a 40MHz bus clock, the maximum value of t_{PGS} is:

$$25ns \times 2^{12} = 102.4\mu s$$

The default value of t_{PGS} is 25 ns $\times 2^{9}$ = 12.8 s

5.11.14 Flash T_{PROG} Limit Register (FIU_TPROGL)

FIU_BASE+$D	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	TPROGL													
Write																
Reset	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1

Figure 5-21. Flash T_{PROG} Limit Register (FIU_TPROGL)

See Programmer Sheet on Appendix page A-33

5.11.14.1 Reserved—Bits 15–14

This bit field is reserved or not implemented. It is read as 0 and cannot be modified by writing.

5.11.14.2 Timer Program Limit (TPROGL)—Bits 13–0

Timeout value for the t_{PROG} counter in terms of bus clocks. This counter controls the Program Time of the Flash. At a 40MHz bus clock, the maximum value of t_{PROG} is:

$$25ns \times 2^{14} = 409.6\mu s$$

The default t_{PROG} value is $25\ ns \times 2^{10} = 25.6\mu s$

5.11.15 Flash T_{NVH} Limit Register (FIU_TNVHL)

This register is reset during any system reset. The register may *only* be changed when the BUSY bit is clear. IS[11] is asserted should this occur.

FIU_BASE+$E	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	TNVHL										
Write																
Reset	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Figure 5-22. Flash T_{NVH} Limit Register (FIU_TNVHL)

See Programmer Sheet on Appendix page A-34

5.11.15.1 Reserved—Bits 15–11

This bit field is reserved or not implemented. It is read as 0 and cannot be modified by writing.

5.11.15.2 Timer Non-Volatile Hold Limit (TNVHL)—Bits 10–0

Timeout value for the T_{NVHL} counter during page erase or program in terms of bus clocks. This counter controls the NVSTR Hold Time of the Flash. At a 40MHz bus clock, the maximum value of t_{NVH} is:

$$25\ ns \times 2^{11} = 51.2\mu s$$

The default value of t_{NVH} is $25ns \times 2^{8} = 6.4\mu s$

The counter controlled by this register is used for Page Erase mode *only*.

5.11.16 Flash T_{NVH1} Limit Register

This register is reset during any system reset. It may *only* be changed when the BUSY bit is clear. IS[11] is asserted should this occur.

FIU_BASE+$F	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	TNVH1L														
Write																
Reset	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1

Figure 5-23. Flash T_{NVH1} Limit Register (FIU_TNVH1L)

See Programmer Sheet on Appendix page A-34

5.11.16.1 Reserved—Bit 15

This bit is reserved or not implemented. It is read as 0 and cannot be modified by writing.

5.11.16.2 Timer Non-Volatile Hold 1 Limit (TNVH1L[14:0])—Bits 14–0

Timeout value for the t_{NVH1} counter during mass erase in terms of bus clocks. This counter controls the NVSTR Hold Time of the Flash. At a 40MHz bus clock, the maximum value of t_{NVHL} is:

$$25ns \times 2^{15} = 819.2\mu s$$

The default value of $t_{NVH1}$is $25ns \times 2^{12} = 102.4\mu s$

The counter controlled by this register is used for Mass Erase mode *only*.

5.11.17 Flash T_{RCV} Limit Register (FIU_TRCVL)

This register is reset during any system reset. The register may *only* be changed when the BUSY bit is clear. IS[11] is asserted should this occur.

FIU_BASE+$10	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	0	0	TRCVL								
Write																
Reset	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1

Figure 5-24. Flash T_{RCV} Limit Register (FIU_TRCVL)

See Programmer Sheet on Appendix page A-34

5.11.17.1 Reserved—Bits 15–9

This bit field is reserved or not implemented. It is read as 0 and cannot be modified by writing.

5.11.17.2 Timer Recovery Limit (TRCVL[8:0])—Bits 8–0

Timeout value for the t_{RCV} counter in terms of bus clocks. This counter controls the recovery time of the Flash. At an 40MHz bus clock, the maximum value of t_{RCV} is:

$$25\text{ns} \times 2^9 = 12.8\mu\text{s}$$

The default value of t_{RCV} is $25\text{ ns} \times 2^6 = 1.6\mu\text{s}$

The counter controlled by this register is used for both Page and Mass Erase modes.

5.11.18 Flash Interface Unit Timeout Registers

Timeout Limit registers default to the correct values for 40MHz bus operation and will require reprogramming for other bus frequencies. If programming is required, it would occur during the power-up sequence, or as an integral part of the programming algorithm. Program/erase times and their registers should never require reprogramming by the end user.

Timeout Limit registers are not allowed to be written while BUSY is asserted. IS[11] is asserted should this occur. IS[3] through IS[9] will be set when the various timers reach the values specified in the Timeout Limit registers.

5.12 Reset

A reset asserted for any reason should switch the Flash into the Standby mode. If a program/erase is interrupted in progress by a reset, data within the enabled area of the Flash will be indeterminate.

5.13 Interrupts

The FIU OR’s all bits in the Interrupt Pending (FIU_IP) register to determine the value of the single interrupt signal presented to the host processor interrupt controller.

The FIU_IP register is equal to the bit-wise *and*ing of the Interrupt Enable (FIU_IE) register and the Interrupt Source (FIU_IS) register. Interrupt sources may be polled at any time in the FIU_IS register, even if those sources are disabled. The FIU_IP register reflects only those sources causing an interrupt to the host processor.

- Interrupts must be cleared by clearing the appropriate bit in the FIU_IS register. This is achieved by the core under software control.
- The FIU is responsible for generating read out-of-range interrupts when the Information Block is enabled. But an attempt must also be made to access past the two rows of that particular block, but within the overall size of the Main Block. The FIU is *not* responsible for generating interrupts for accesses outside of this range.

- If the host processor attempts a read or write to the Flash while it is being erased or programmed, the FIU will generate an interrupt. It will not affect the contents of the Limit register in question if the corresponding enable bit in FIU_IE is set.
- Writes to the FIU_CNTL register are prohibited once the Intelligent Program or Intelligent Erase sequences are begun by the FIU state machine. It is illegal to change these settings until after the program/erase task is complete and the BUSY bit is clear. Any attempt to do so will not affect the register contents, and may cause an error interrupt to be posted if the corresponding enable bit in FIU_IE is set.

Table 5-13. Document Revision History for Chapter 5

Version History	Description of Change
Rev. 8	Formatting, layout, spelling, and grammar corrections. Added revision history table.

Chapter 6
External Memory Interface (EMI)

6.1 Introduction

Digital signal controllers 56F803, 56F805, and 56F807 provide a port for external memory interfacing. This chapter details the pins and programming specifics for an External Memory Interface (EMI), also known as Port A. This port provides 16 pins for an external address bus, 16 pins for an external data bus, and four pins for bus control. Together, these 36 pins comprise Port A. Devices 56F801/802 have no port for external memory interfacing.

6.2 Block Diagram

6.2.1 External Memory Port Architecture

Figure 6-1 illustrates the general Block Diagram of the 56F803/805/807 input/output.

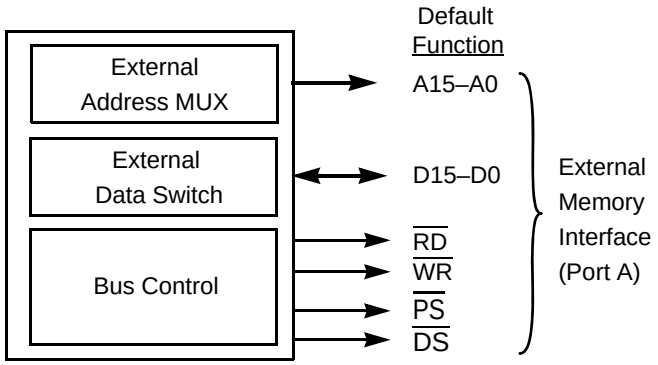


Figure 6-1. 56F803/805/807 Input/Output Block Diagram

6.3 Pin Descriptions

The names of the 56F803/805/807 External Memory port pins are as follows:

- Address Bus Output (A15–0)
- Data Bus (D15–0)
- Read Enable ($\overline{RD}$)

- Write Enable ($\overline{WR}$)
- Program Memory Select ($\overline{PS}$)
- Data Memory Select ($\overline{DS}$)

6.4 Register Definition

The External Memory Interface (EMI), also referred to as Port A, is the port through which all accesses to external memories and external memory-mapped peripherals are made. This port contains a 16-bit address bus, a 16-bit data bus, and four-bus control pins for strobes. The EMI uses one programmable register for bus control, the Bus Control Register (BCR).

Software-controlled wait states can be introduced when accessing slower memories, or peripherals. Wait states are programmable using the BCR register. **Figure 6-3** and **Figure 6-4** illustrate bus cycles with and without wait states.

Table 6-1. EMI Register Summary

Address	Register Acronym	Register Name	Access Type	Chapter Location
X:$FFF9	BCR	Bus Control Register	Read/Write	Section 6.4.1

6.4.1 Bus Control Register (BCR)

The BCR, located at X:$FFF9, is a 16-bit, read/write register used for inserting software wait states on accesses to external program or data memory. Two, 4-bit wait state fields are provided, each capable of specifying from 0, 4, 8, or 12-wait states. On processor reset, each wait state field is set to $C so 12-wait states are inserted, allowing slower memory to be used immediately after reset. Bits 0, 1, 4, and 5 are ignored in determining the wait states. Therefore, writing values other than 0, 4, 8, and 12 will result in the nearest lower count wait state.

BCR—$FFF9	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	0	DRV	0	WAITSTATE FIELD for EXTERNAL WSX[4:7]				WAIT STATE FIELD for EXTERNAL WSP[0:3]			
Write																
Reset	0	0	0	0	0	0	0	7	1	1	0	0	1	1	0	0

Figure 6-2. Bus Control Register (BCR)

See Programmer Sheet on Appendix page A -19

6.4.1.1 Reserved—Bits 15–10

This bit field is reserved or not implemented. It is read/written using 0.

6.4.1.2 Drive (DRV)—Bit 9

The DRIVE (DRV) control bit is used to specify what occurs on the external memory port pins when no external access is performed—whether the pins remain driven or are placed in tri-state. Accessing external memory when DRV = 0 may cause unintentional results. Therefore, it is recommended DRV = 1 be defined by the user. Please see **Section 6.4.2, "State of Pins in Different Processing States".**

6.4.1.3 Reserved—Bit 8

This bit is reserved or not implemented. It is read/written using 0.

6.4.1.4 Wait State X Data Memory (WSX)—Bits 7–4

The wait state X data memory (WSX[3:0]) control bits allow programming of the wait states of external X data memory. **Table 6-2** illustrates the wait states provided with these bits. The WSX[3:0] and the WSP[3:0] bits are programmed in the same fashion, but do not need to be set to the same value.

Table 6-2. Programming WSP[3:0] and WSX[3:0] Bits for Wait States

Bit String	Hex Value	Number of Wait States
0000	$0	0
0100	$4	4
1000	$8	8
1100	$C	12

6.4.1.5 Wait State P Program Memory (WSP)—Bits 3–0

The Wait state P program memory (WSP[3:0]) control bits allow for programming of the wait states for external program memory. These bits are programmed as shown in **Table 6-2.**

Note: Bits zero and one in the 4-bit string of **Table 6-2** for both WSP[3:0] and WSX[3:0] are ignored in evaluating the number of wait states to be inserted. That is, writing a value of seven (0111) will result in only four (0100) four wait states.

Figure 6-3 illustrates an example of the bus cycles without wait states. This timing relationship will not function ideally at an 80MHz frequency, but it will function according to the figure at lower frequencies. The critical factor is the data must have stabilized before the beginning of the T3 period, demonstrated by the dark gray area in **Figure 6-3.**

Figure 6-4 exhibits an example of the bus cycles with wait states. For more information on wait states, see the corresponding chip's technical data sheet.

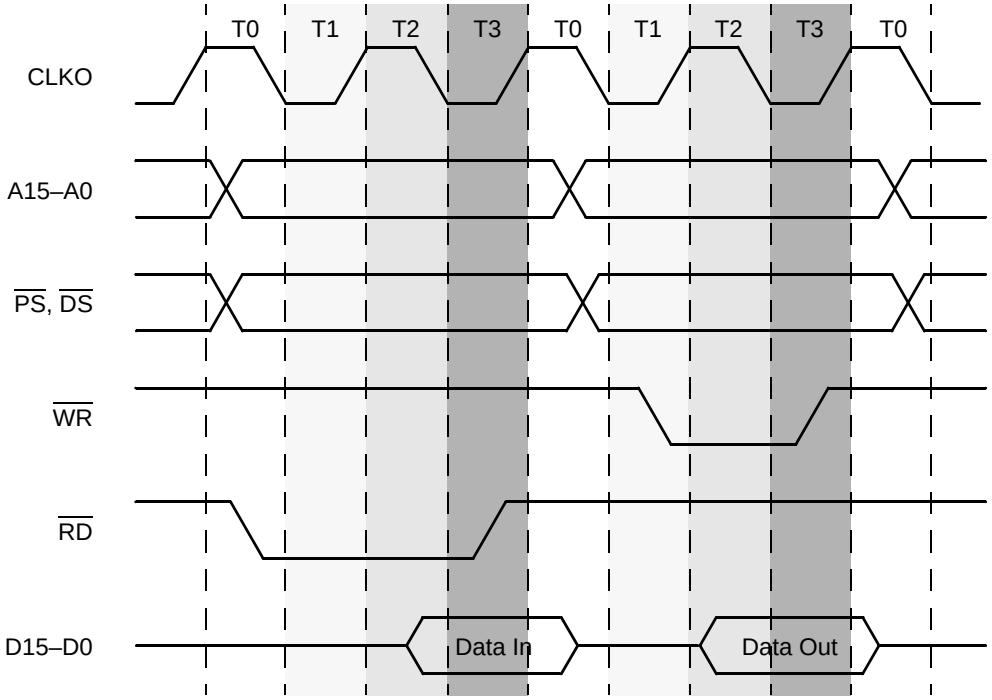


Figure 6-3. Bus Operation (Read/Write–Zero Wait States)

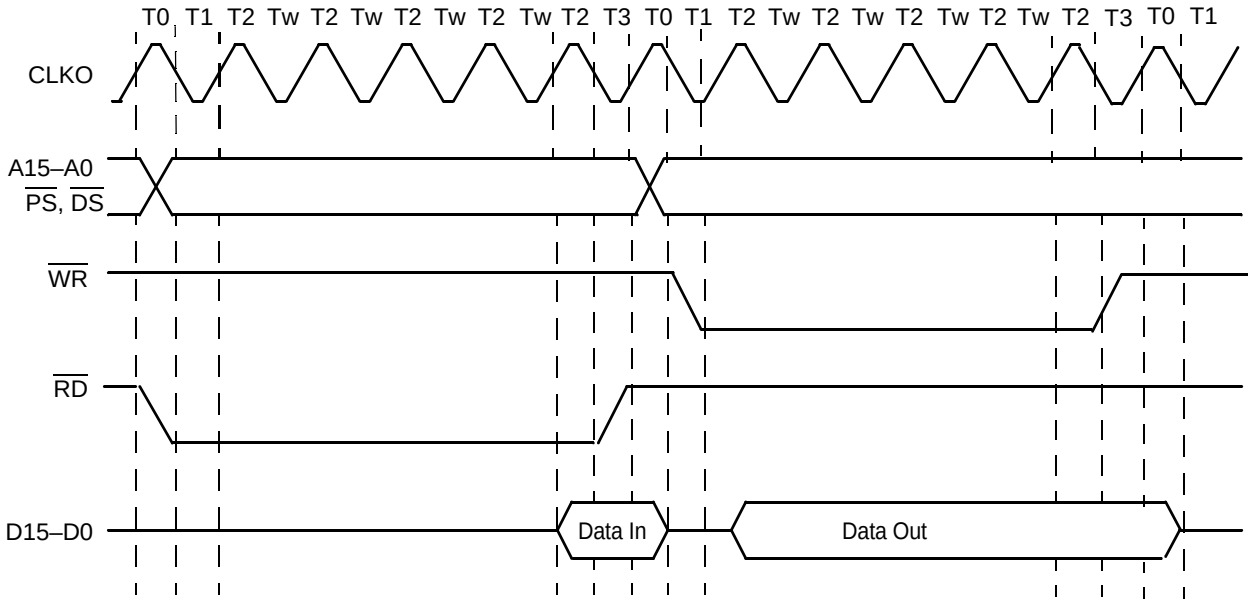


Figure 6-4. Bus Operation (Read/Write–Four Wait States)

6.4.2 State of Pins in Different Processing States

The DSP56800 core can be in one of six processing states, also called modes:

1. Normal
2. Exception
3. Reset
4. Wait
5. Stop
6. Debug

In Normal mode, each instruction cycle has two possible cases—either the processor needs to perform an external access, or all accesses are accomplished internally. Exception processing is similar. In the case of an external access, the Address and bus control pins are driven to perform a normal bus cycle. The data bus is also driven if the access is in a write cycle.

In the case where there is no external access on a particular instruction cycle, one of two things may occur:

1. The external address bus and control pins can remain driven with their previous values.
2. They can be tri-stated.

Note: The pull-ups for the address, data, and control pins are normally enabled by the state of the DRV bit. The data, control, and address [5:0] pull-ups may be disabled by setting the appropriate bits in the SYS_CNTL register in the reset wait module. The pull-ups for address bits [15:8] are controlled by the GPIOA PUR register. The pull-ups for address bits [7:6] are controlled by the GPIOE PUR register bits[3:2].

Reset mode *always* tri-states the external address bus and internally pulls control pins high. Stop and Wait modes, however, either tri-state the address bus and control pins or let them remain driven with their previous values. This selection is made based on the value of the DRV bit in the BCR. **Table 6-3** and **Table 6-4** describe the operation of the external memory port in these different modes. Debug mode is a special state used to debug and test programming code. This state is detailed in **Section 9** of the *DSP56800 Family Manual*, DSP56800FM. The state is not described in the following tables.

Table 6-3. Port A Operation with DRV Bit = 0

Mode	Pins		
	A0–A15	$\overline{PS}$, $\overline{DS}$, $\overline{RD}$, $\overline{WR}$	D0–D15
Normal mode, external access	Driven	Driven	Driven
Normal mode, internal access	Tri-stated	Tri-stated	Tri-stated
Stop mode	Tri-stated	Tri-stated	Tri-stated
Wait mode	Tri-stated	Tri-stated	Tri-stated
Reset mode	Tri-stated	Pulled high internally	Tri-stated

Table 6-4. Port A Operation with DRV Bit = 1

Mode	Pins		
	A0–A15	$\overline{PS}$, $\overline{DS}$, $\overline{RD}$, $\overline{WR}$	D0–D15
Normal mode, external access	Driven	Driven	Driven
Normal mode, internal access	Driven	Driven	Tri-stated
Stop mode	Driven	Driven	Tri-stated
Wait mode	Driven	Driven	Tri-stated
Reset mode	Tri-stated	Pulled high internally	Tri-stated

During a Normal mode external access, data lines are driven *only* during an external write cycle.

Table 6-5. Document Revision History for Chapter 6

Version History	Description of Change
Rev. 8	Formatting, layout, spelling, and grammar corrections. Added revision history table. In Table 6-1, changed the first column heading (was "Address Offset", is "Address") and the associated value (was "Base+$9", is "X:$FFF9").

Chapter 7
General Purpose Input/Output (GPIO)

7.1 Introduction

The 56F80x General Purpose Input/Output (GPIO) pins are designed to share package pins with other peripherals on the chip. If a peripheral is not required, the pin may be programmed as input, output, or level-sensitive interrupt input. GPIOs are placed on the chip in groups of eight bits. GPIO pins and their respective connections with some peripheral devices for the 56F805 are illustrated in **Figure 7-3.** Logic associated with just one of those eight bits is set out in **Figure 7-4.** For information about which GPIO pins are multiplexed and shared with other peripherals, please refer to Chapter Two, *Pin Descriptions*.

7.2 Block Diagrams

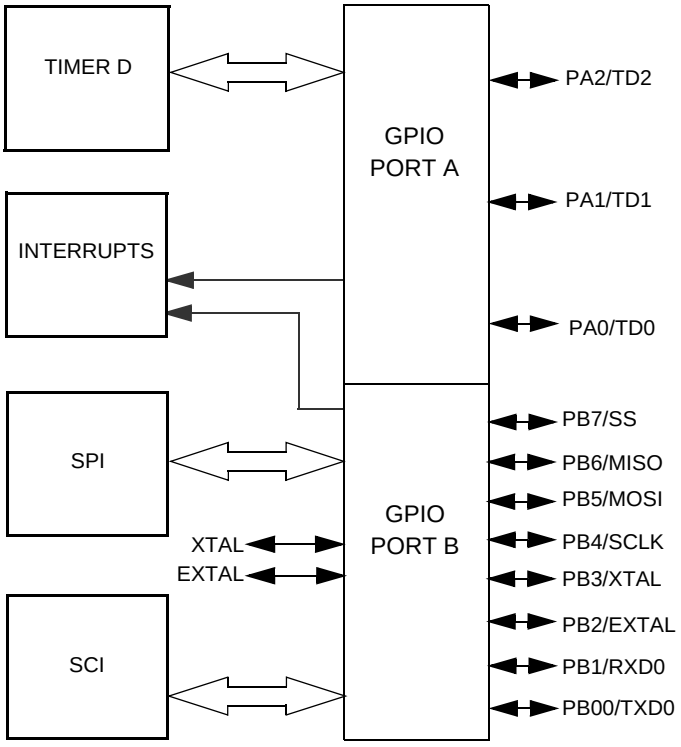


Figure 7-1. Block Diagram Showing 56F801 GPIO Connections

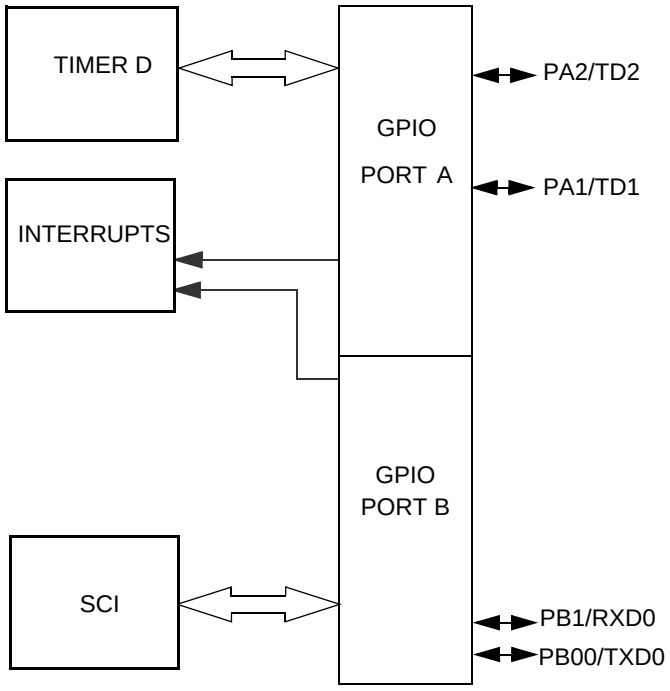


Figure 7-2. Block Diagram Showing 56F802 GPIO Connections

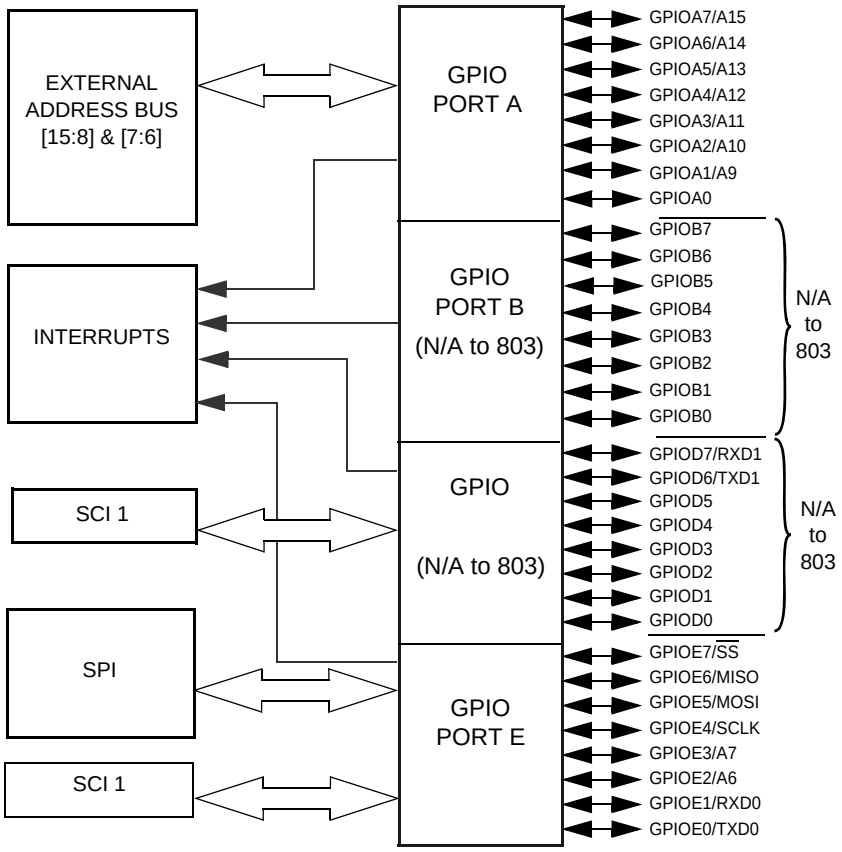


Figure 7-3. Block Diagram Showing 56F803/805/807 GPIO Connections

7.2.1 Summary: Dedicated and Multiplexed GPIOs

- 56F801—Has no dedicated GPIOs; but 11 multiplexed GPIOs on Ports A and B
- 56F802—Has no dedicated GPIOs; but four multiplexed GPIOs on Ports A and B
- 56F803—Has no dedicated GPIOs; but 16 multiplexed GPIOs on Ports A and E
- 56F805—Has eight dedicated pins on Port B while the lower six pins of Port D are dedicated GPIO. There are 18 multiplexed GPIOs available on Ports A, D, and E
- 56F807—Has 14 dedicated pins on Ports B and D while there are 18 multiplexed GPIOs available on Ports A, D, and E

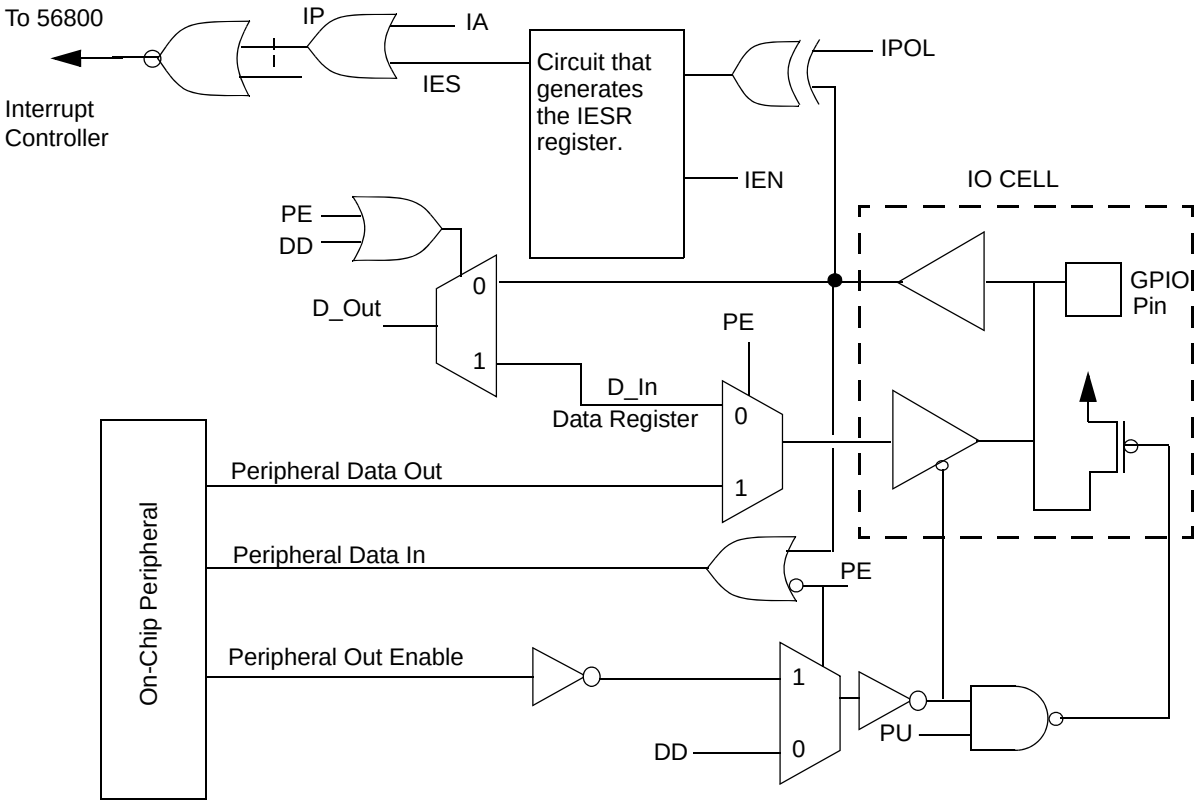


Figure 7-4. Bit-Slice View of the GPIO Logic

Each GPIO pin can be configured in three ways:

1. An input, with or without pull-up
2. An interrupt
3. An output

56F803/805/807 GPIO's pull-ups are configured by writing the Pull-Up Enable Control Register (PUR) and the Data Direction Register (DDR) when the Peripheral Enable Register (PER) is set to zero. When PER is set to one the pull-ups are controlled by the PUR and the direction of the peripheral used. In any case, if the I/O is set to be an output, the pull-up is disabled. The 56F803/805/807 GPIO interfaces with the following on-chip devices:

- External address bus
- SCI0
- SCI1
- SPI

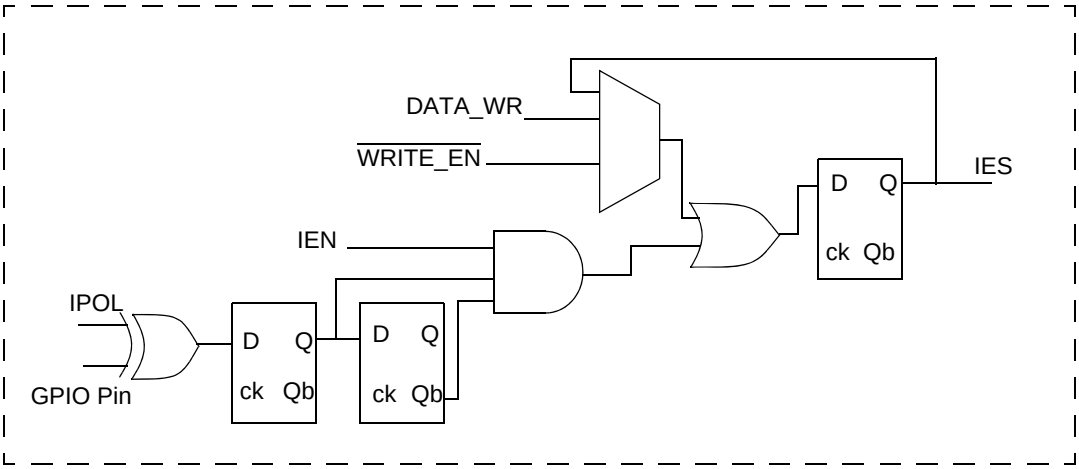


Figure 7-5. Edge Detector Circuit

7.3 GPIO Interrupts

The GPIO has two types of interrupts:

1. Software interrupt for testing purposes
2. Hardware interrupt from the corresponding GPIO pin

The software Interrupt Assert Register (IAR) can be tested by writing ones to the IAR. The IPR associated with the GPIO (GPIO_IPR) will record the value of IAR. The GPIO_IPR can be cleared by writing zeros into the IAR during the IAR testing.

Note: When testing the IAR, the Interrupt Polarity Register (IPOLR), Interrupt Edge Sensitive Register (IESR) and Interrupt Enable Register (IENR) must be set to zero guaranteeing the interrupt registered in the GPIO_IPR is due to IAR only.

When a GPIO is used as an interrupt, the IAR must be set to zero. When the GPIO pin signal is asserted, polarity is detected going through the edge detection mechanism. Refer to **Figure 7-4** for an illustration. The value will be seen at the IESR and recorded by the GPIO_IPR. The GPIO_IPR can be cleared by writing zeros into the IESRs. If IPOLR is set to zero, the interrupt at the GPIO pin is active high. If IPOLR is set to one, the interrupt at the GPIO pin is active low.

To present only a single interrupt to the 56800 core, the eight interrupt signals receive an overrun flag. The interrupt service routine must then check the contents of the IPR determining which pin(s) caused the interrupt.

External interrupt sources do not need to remain asserted due to the edge sensitive nature of the detection mechanism.

Table 7-1. GPIO Interrupt Assert Functionality

IPOLR	Interrupt Asserted	Remark
0	High	If the IENR is set to 1, as the GPIO pin goes to high an interrupt will be recorded by the GPIO_IPR register.
1	Low	If the IENR is set to 1, as the GPIO pin goes to low an interrupt will be recorded by the GPIO_IPR.

7.4 Register Definitions

Table 7-2. GPIO Memory Map

Device	Peripheral	Address
801/802/ 803/805	GPIOA_BASE	$0FB0
	GPIOB_BASE	$0FC0
	GPIOD_BASE	$0FE0
	GPIOE_BASE	$0FF0
807	GPIOA_BASE	$13B0
	GPIOB_BASE	$13C0
	GPIOD_BASE	$13E0
	GPIOE_BASE	$13F0

The actual memory address of a register is the sum of a base address and the registers' address offset. The base address is defined at the core. The address offset is defined at the module level. Please reference **Table 3-34 - Table 3-37.**

Table 7-3. GPIO Registers with Their Reset Values

Register	Description	Binary Reset State for Lower 8 bits	Remark
PUR	Pull-up Enable Register	0b11111111	Pull-ups are enabled. (See **Table 7-4**)
DR	Data Register	0b00000000	DR is used for data interface between the GPIO pin and the IPBus.
DDR	Data Direction Register	0b00000000	GPIO is set to an input. If DDR is one the GPIO becomes an output.
PER	Peripheral Enable Register	0b11111111	Peripheral controls the GPIO. The PER does not determine the direction of the I/O
IAR	Interrupt Assert Register	0b00000000	No interrupt.
IENR	Interrupt Enable Register	0b00000000	Interrupt is disabled.

Table 7-3. GPIO Registers with Their Reset Values (Continued)

Register	Description	Binary Reset State for Lower 8 bits	Remark
IPOLR	Interrupt Polarity Register	0b00000000	When set to 1, the interrupts are active low, and when set to 0, interrupts are active high.
GPIO_IPR	Interrupt Pending Register	0b00000000	No interrupt is registered. A 1 indicates an interrupt.
IESR	Interrupt Edge Sensitive Register	0b00000000	A 1 indicates that an edge has been detected.

For more information on GPIO registers on Port A, please refer to **Table 3-34.** GPIO registers on Port B are illustrated in **Table 3-35.** GPIO registers on Port D are referenced in **Table 3-36,** while GPIO registers on Port E are illustrated in **Table 3-37.**

Table 7-4 illustrates the state of the GPIO pin and pull-up resistor.

Table 7-4. GPIO Pull-Up Enable Functionality

Peripheral Out Disable	PER	PUR	DDR	GPIO Pin State	GPIO Pin Pull-Up
x	0	0	0	Input	Disabled
x	0	0	1	Output	Disabled
x	0	1	0	Input	Enabled
x	0	1	1	Output	Disabled
0	1	x	x	Output	Disabled
1	1	0	x	Input	Disabled
1	1	1	x	Input	Enabled

Note: When the Periphery Enable Register (PER) is a zero, the Pull-Up Enable Register (PUR) and the Data Direction Register (DDR) control the GPIO pin pull-up. When the PER is one, the GPIO pin pull-up is controlled by the PUR and peripheral output enable. The PUR value is recognized when the GPIO pin is configured as an input only.

The Data Register (DR) is used to pass data to the GPIO pin or the IPBus. It is also used to store data from the GPIO pin and the IPBus.

Table 7-5. GPIO Data Transfers Between GPIO Pin and IPBus

Peripheral Out Enable	PER	DDR	GPIO Pin State	Access	Data Access Result
X	0	0	Input	Write to DR	Data is written into DR by IPBus. No effect on the GPIO pin value.
X	0	1	Output	Write to DR	Data is written into the DR by the IPBus. DR value seen at GPIO pin.
X	0	0	Input	Read from DR	GPIO pin state is read by the IPBus. No effect on DR value.
X	0	1	Output	Read from DR	DR value is read by the IPBus. DR value seen at GPIO pin.
1	1	X	Input	Write to DR	Data is written into the DR by IPBus. No effect on GPIO pin value.
0	1	X	Output	Write to DR	Data is written into the DR by the IPBus. p_mp_odata is seen at GPIO pin.
1	1	X	Input	Read from DR	DR value is read by the IPBus. No effect on the GPIO pin or DR value.
0	1	X	Output	Read from DR	DR value is read by the IPBus. p_mp_odata is seen at the GPIO pin.

7.5 Chip Specific Configurations

Table 7-6. GPIO Assignments

	Port A	Port B	Port D	Port E
56F801	—	8 Shared	3 Shared	—
56F802	—	8 Shared	3 Shared	—
56F803	8 Shared	—	—	8 Shared
56F805	8 Shared	8 Dedicated	6 Dedicated, 2 Shared	8 Shared
56F807	8 Shared	8 Dedicated	6 Dedicated, 2 Shared	8 Shared

Dedicated GPIOs are intended only for use as GPIOs. Shared or muxed indicates pins may alternately be used as GPIO; however they are typically used for other functions. The programming model is identical for dedicated versus shared GPIO. Dedicated GPIOs have simply had their peripheral data out and enable inputs tied to V_{DD}, meaning the peripheral data out is disabled.

7.6 Register Definitions

Table 7-7. GPIO Memory Map

Device	Peripheral	Address
801/802/ 803/805	GPIOA_BASE	$0FB0
	GPIOB_BASE	$0FC0
	GPIOD_BASE	$0FE0
	GPIOE_BASE	$0FF0
807	GPIOA_BASE	$13B0
	GPIOB_BASE	$13C0
	GPIOD_BASE	$13E0
	GPIOE_BASE	$13F0

Each GPIO module contains nine, 8-bit registers each performing an identical function for one of the eight GPIO pins controlled by its port. The address of a register is the sum of a base address and an address offset. The base address is defined at the MCU level. The address offset is defined at the module level. Please refer to **Table 3-3** for GPIOA_BASE, GPIOB_BASE, GPIOD_BASE, and GPIOE_BASE definitions.

For example, the Pull-Up Enable Register (PUR) for GPIO on Port A, is called GPIO_A_PUR and is found at memory location GPIOA_BASE+$0.

Note: Be certain to consider the specific chip's available GPIO ports. Not all GPIO ports are available on all chips. 56F801/802 have GPIO on Ports A and B. 56F803 has GPIO on Ports A and E. 56F805 and 56F807 have GPIO on Ports A, B, D, and E.

Table 7-8. GPIO Register Summary

Address Offset	Register Acronym	Register Name	Access Type	Register Location
Base + $0	GPIOX_PUR	Pull-Up Enable Register	Read/Write	**Section 7.6.1**
Base + $1	GPIOX_DR	Data Register	Read/Write	**Section 7.6.2**
Base + $2	GPIOX_DDR	Data Direction Register	Read/Write	**Section 7.6.3**
Base + $3	GPIOX_PER	Peripheral Enable Register	Read/Write	**Section 7.6.4**
Base + $4	GPIOX_IAR	Interrupt Assert Register	Read/Write	**Section 7.6.5**
Base + $5	GPIOX_IENR	Interrupt Enable Register	Read/Write	**Section 7.6.6**
Base + $6	GPIOX_IPOLAR	Interrupt Polarity Register	Read/Write	**Section 7.6.7**
Base + $7	GPIOX_IPR	Interrupt Pending Register	Read/Write	**Section 7.6.8**
Base + $8	GPIOX_IESR	Interrupt Edge-Sensitive Register	Read/Write	**Section 7.6.9**

Each GPIO module has nine, 8-bit registers provided in **Figure 7-6.**

Add. Offset	Register Name		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$0	PUR	R	0	0	0	0	0	0	0	0	PU							
		W																
$1	DR	R	0	0	0	0	0	0	0	0	D							
		W																
$2	DDR	R	0	0	0	0	0	0	0	0	DD							
		W																
$3	PER	R	0	0	0	0	0	0	0	0	PE							
		W																
$4	IAR	R	0	0	0	0	0	0	0	0	IA							
		W																
$5	IENR	R	0	0	0	0	0	0	0	0	IEN							
		W																
$6	IPOLR	R	0	0	0	0	0	0	0	0	IPOL							
		W																
$7	IPR	R	0	0	0	0	0	0	0	0	IP							
		W																
$8	IESR	R	0	0	0	0	0	0	0	0	IES							
		W																

R	0	Read as 0
W		Reserved

Figure 7-6. GPIO Register Map Summary

7.6.1 Pull-Up Enable Register (PUR)

The PUR read/write register is for pull-up enabling and disabling. If an GPIO pin is configured as an input, the PU will enable the pull-ups when set to one. If the PUR is set to zero, the pull-ups will be disabled. If the GPIO pin is configured as an output, the PUR has no effect. See **Table 7-4** for all possible combinations. The PUR is set to one on processor reset.

GPIO_BASE+$0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	0	0	0	PU							
Write																
Reset	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Figure 7-7. Pull-Up Enable Register (PUR)

See Table A-8, List of Programmer's Sheets

7.6.2 Data Register (DR)

The purpose of Data register (DR) is for holding data coming either from the GPIO pin or the IPBus. Meaning the D is the data interface between the GPIO pin and the IPBus. See **Table 7-5** for data transfers between the GPIO pin and the IPBus.

GPIO_BASE+$1	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	0	0	0	D							
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 7-8. Data Register (DR)

See Table A-8, List of Programmer's Sheets

7.6.3 Data Direction Register (DDR)

When the Peripheral Enable register (PER), a read/write register, is set to zero, the GPIO pin is configured either as input or output by the DDR. When DDR is set to zero, the GPIO pin is an input, with pull-up device while PUR is set to one, or without the pull-up device if PU is set to zero. When the DDR is set to one the GPIO pin is an output. See **Table 7-4** for more information.

GPIO_BASE+$2	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	0	0	0	DD							
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 7-9. Data Direction Register (DDR)

See Table A-8, List of Programmer's Sheets

7.6.4 Peripheral Enable Register (PER)

The Peripheral Enable Register (PER) is read and write register. This register determines the GPIO's configuration. When the PE value is one, the peripheral manages the GPIO pin. This mastery includes configuring the GPIO pin as a required input, with or without pull-up. It also may be an output, depending on the status of the peripheral output enable. It includes data transfers from the GPIO pin to the peripheral. See **Table 7-4** for more details.

When the PER value is zero, the DDR determines the direction of data flow. When DD is zero, the GPIO pin is input only. When DDR is one the GPIO pin is an output only.

GPIO_BASE+$3	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	0	0	0	PE							
Write																
Reset	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Figure 7-10. Peripheral Enable Register (PER)

See Table A-8, List of Programmer's Sheets

7.6.5 Interrupt Assert Register (IAR)

This is a read/write register. The Interrupt Assert register (IAR) is only for software testing. When the IAR is one, an interrupt is asserted and can be cleared by writing zeros into the IAR.

GPIO_BASE+$4	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	0	0	0	IA							
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 7-11. Interrupt Assert Register (IAR)

See Table A-8, List of Programmer's Sheets

7.6.6 Interrupt Enable Register (IENR)

This is a read/write register. It enables or disables the edge detection for any incoming (external) interrupt from the GPIO pin. This register is set to one for interrupt detection. The interrupts are recorded in the IPR.

GPIO_BASE+$5	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	0	0	0	IEN							
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 7-12. Interrupt Enable Register (IENR)

See Table A-8, List of Programmer's Sheets

7.6.7 Interrupt Polarity Register (IPOLR)

This read/write register is used to configure external interrupt's polarity detection. When this register is set to one, the interrupt at the GPIO pin is active low. When set to zero, the interrupt seen at the GPIO pin is active high. This is true, however, only when the IENR is set to one. There is no effect on the interrupt if the IEN is set to zero.

GPIO_BASE+$6	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	0	0	0	IPOL							
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 7-13. Interrupt Polarity Register (IPOLR)

See Table A-8, List of Programmer's Sheets

7.6.8 Interrupt Pending Register (IPR)

This *read-only* register is used to record any incoming interrupts. This register is read to determine which pin has caused an interrupt. The register is cleared by writing zeros into the IAR

when the interrupt is caused by software. For external interrupts, the IPR is cleared by writing zeros into the Interrupt Edge Sensitive register (IESR).

GPIO_BASE+$7	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	0	0	0	IP							
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 7-14. Interrupt Pending Register (IPR)

See Table A-8, List of Programmer's Sheets

7.6.9 Interrupt Edge Sensitive Register (IESR)

The IESR records the interrupt, detecting an edge by the edge detector circuit, and the corresponding bit in the IPR will be set to one. This is a read/write register; however, the only time the register can receive writing is to *clear* the IPR. Only write zeros into this register to clear the IP.

Note: Writing ones into this register will result in false interrupts.

GPIO_BASE+$8	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	0	0	0	IES							
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 7-15. Interrupt Edge-Sensitive Register (IESR)

See Table A-8, List of Programmer's Sheets

7.7 GPIO Programming Algorithms

Example 7-1. Algorithm for Port B Pins as Inputs Used for Receiving Data

- INPUT for Port B
  - START EQU $0080 ;Start of program
  - PBPUR EQU $0FC0 ;Port B pull-up register
  - PBDR EQU $0FC1 ;Port B data register
  - PBDDR EQU $0FC2 ;Port B data direction register
  - PBPER EQU $0FC3 ;Port B peripheral register
  - PBIAR EQU $0FC4 ;Port B interrupt assert register
  - PBIENR EQU $0FC5 ;Port B interrupt enable register
  - PBIPOLR EQU $0FC6 ;Port B interrupt polarity register

```
    — PBIPR       EQU     $0FC7     ;Port B interrupt pending register
    — PBIESR      EQU     $0FC8     ;Port B interrupt edge sensitive register
    — data_i      EQU     $0001     ;data input
• Vector Setup
    — ORG         P:$0000           ;Cold Boot
    — JMP         START             ;Hardware RESET vector
    — ORG         P:START           ;Start of program
• General Setup
    — MOVE        #$0200,X:BCR      ;External memory has 0 states
• Port B Setup
    — MOVE        #$0000,X:PBIAR    ;Disable Port B interrupts
    — MOVE        #$0000,X:PBIENR   ;Disable Port B interrupts
    — MOVE        #$0000,X:PBIPOLR  ;Disable Port B interrupts
    — MOVE        #$0000,X:PBIESR   ;Disable Port B interrupts
    — MOVE        #$0000,X:PBPER    ;Configures Port B pins as dedicated GPIOs
    — MOVE        #$0000,X:PBDDR    ;Selects Port B pins as inputs (default)
• Main Routine
    — INPUT                         ;Input Loop
    — MOVE        X:PBDR,X0         ;Read PB0-PB7 into bits 0-7 of "data_i"
    — MOVE        X0,X:data_i       ;Memory to memory move requires 2 moves
    — BRA         INPUT
```

Example 7-2. Algorithm for Port B Pins as Outputs Used for Sending Data

```
• INPUT for Port B
    — START       EQU     $0080     ;Start of program
    — PBPUR       EQU     $0FC0     ;Port B pull-up register
    — PBDR        EQU     $0FC1     ;Port B data register
    — PBDDR       EQU     $0FC2     ;Port B data direction register
    — PBPER       EQU     $0FC3     ;Port B peripheral register
    — PBIAR       EQU     $0FC4     ;Port B interrupt assert register
```

```
    – PBIENR      EQU     $0FC5     ;Port B interrupt enable register
    – PBIPOLR     EQU     $0FC6     ;Port B interrupt polarity register
    – PBIPR       EQU     $0FC7     ;Port B interrupt pending register
    – PBIESR      EQU     $0FC8     ;Port B interrupt edge sensitive register
    – data_o      EQU     $0001     ;data input
• Vector Setup
    – ORG         P:$0000           ;Cold Boot
    – JMP         START             ;Hardware RESET vector
    – ORG         P:START           ;Start of program
• General Setup
    – MOVE        #$0200,X:BCR      ;External memory has 0 states
• Port B Setup
    – MOVE        #$0000,X:PBIAR    ;Disable Port B interrupts
    – MOVE        #$0000,X:PBIENR   ;Disable Port B interrupts
    – MOVE        #$0000,X:PBIPOLR  ;Disable Port B interrupts
    – MOVE        #$0000,X:PBIESR   ;Disable Port B interrupts
    – MOVE        #$0000,X:PBPER    ;Configures Port B pins as dedicated GPIOs
    – MOVE        #$FFFF,X:PBDDR    ;Selects Port B pins as outputs
• Main Routine
    – OUTPUT                        ;Output Loop
    – MOVE        X:data_o,X0       ;Put bits 0-7 of "data_o" on pins PB0-PB7.
    – MOVE        X0,X:PBDR         ;Memory to memory move requires 2 moves
    – BRA         OUTPUT
```

Table 7-9. Document Revision History for Chapter 7

Version History	Description of Change
Rev. 8	Formatting, layout, spelling, and grammar corrections. Added revision history table. Removed trailing "R" from the bit field names in the register figures.

Chapter 8
Controller Area Network (CAN)

8.1 Introduction

The scalable Controller Area Network (CAN) is available on three of the four digital signal controller core-based family chips: 56F803, 56F805, and 56F807. The 56F801 and 56F802 devices have no CAN.

The CAN definition is based on the CAN12 definition, the specific implementation of the Freecale scalable CAN concept, originally targeted for the MC68HC12 micro controller family.

The module is a communication controller implementing the CAN 2.0 A/B protocol, as defined in the *Bosch Specification* dated September 1991. To fully understand the CAN specification, recommended first reading the *Bosch Specification* will provide adaptation to terms and concepts contained within the document.

The CAN protocol was primarily designed to be a vehicle serial data bus, meeting the specific requirements of that field such as:

- Real-time processing
- Reliable operation in the EMI environment of a vehicle
- Cost-effectiveness
- Required bandwidth

The transmission frames in CAN is similar to the ethernet protocol, except for different collision avoidance. To send a ready frame, each station listens for the idle bus state. Once recognized, each node starts sending its frame. If a node sends a 1 but reads back a 0, it stops transmission and goes into Idle Listen mode. Each frame is marked by a unique CAN protocol header, so after transmission of 11 bits, 29 for CAN 2.0B, only one node always wins arbitration. The side effect of the CAN arbitration schema is a unique priority for each frame deemed very important for critical processing.

CAN utilizes an advanced buffer arrangement resulting in a predictable real-time behavior, simplifying the application software.

8.2 Features

Basic features of the CAN:

- Modular architecture
- Implementation of the CAN protocol—Version 2.0A/B
  - — Standard and extended data frames
  - — Zero to eight bytes data length
  - — Programmable bit rate up to 1Mbps
  - — Support for remote frames
- Double-buffered receive storage scheme
- Triple-buffered transmit storage scheme with internal prioritization using a *local priority* concept
- Flexible identifier filter capable of masking supports two full-size extended identifier filters: two 32-bit, four 16-bit, or eight 8-bit filters
- Programmable wake-up functionality with integrated low-pass filter
- Programmable Loop Back mode supports self-test operation
- Separate signalling and interrupt capabilities for all CAN receiver and transmitter error states: warning, error passive, bus off
- Programmable CAN clock source, either IPBus clock or crystal oscillator output
- Three low power modes: Sleep, Soft Reset and Power-Down

8.3 Block Diagram

The following figure illustrates the CAN organization functions.

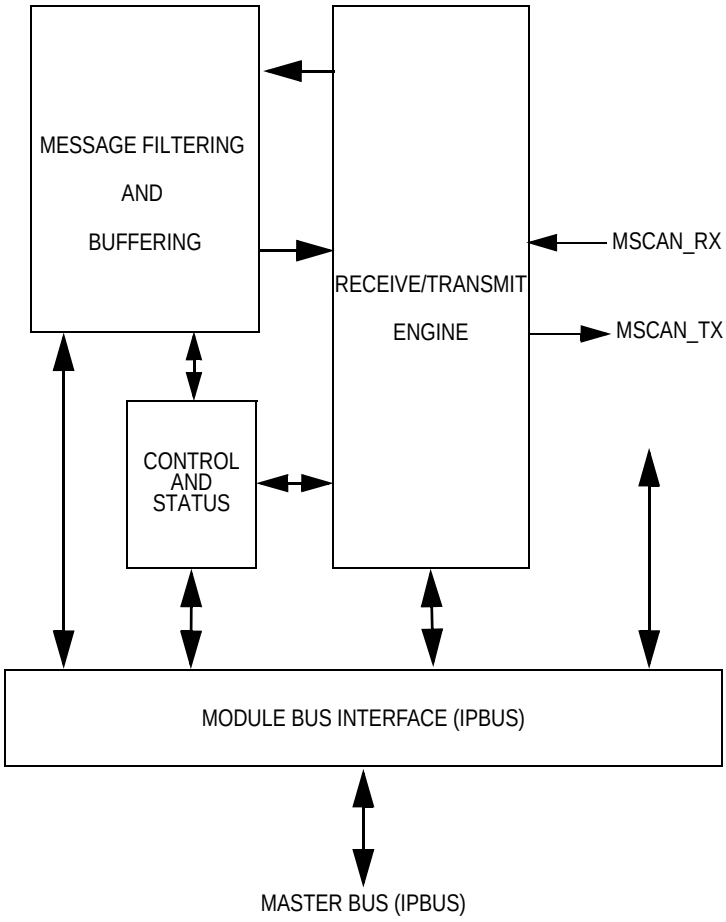


Figure 8-1. CAN Block Diagram

8.4 Functional Description

8.4.1 Message Storage

CAN facilitates a sophisticated message storage system, addressing the requirements of a broad range of network applications.

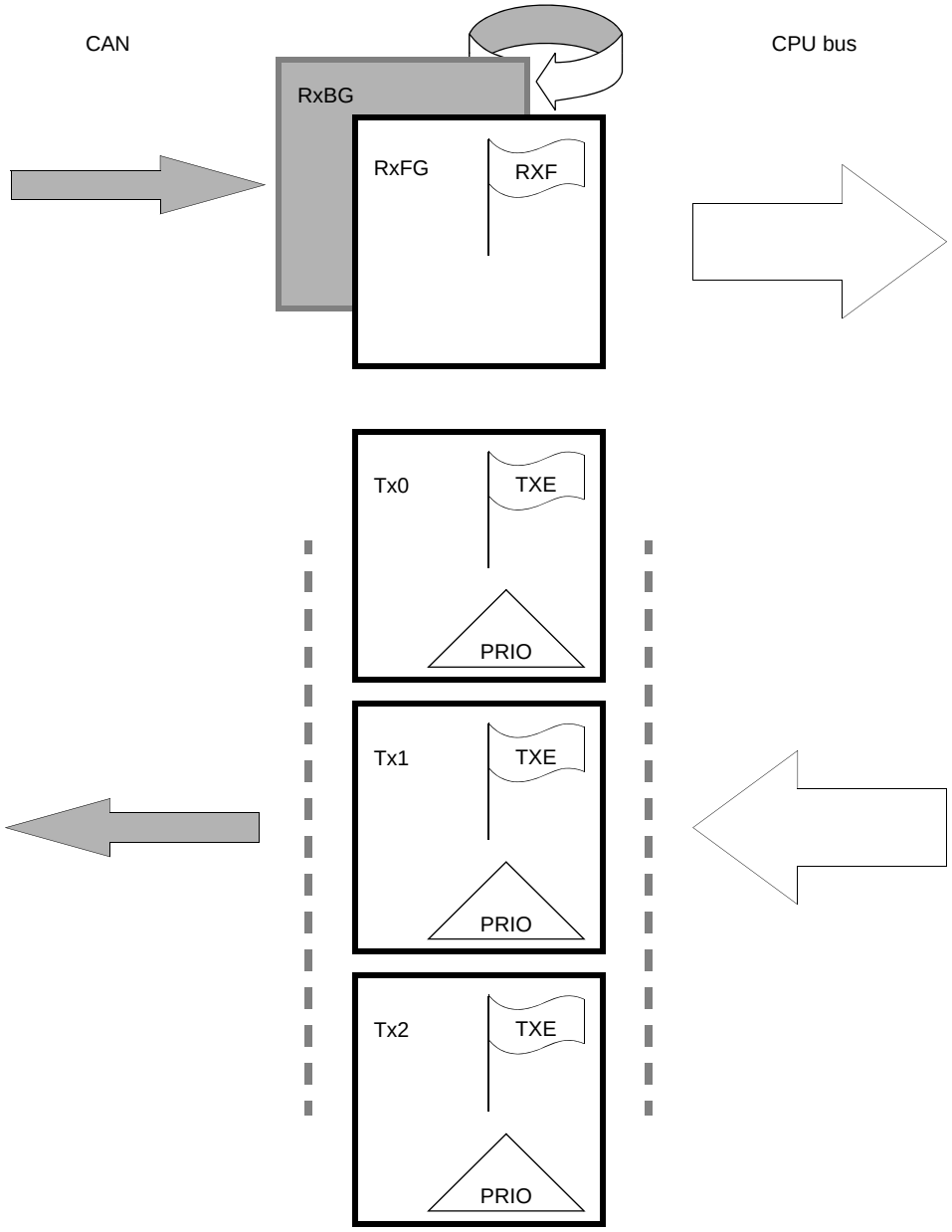


Figure 8-2. User Model for Message Buffer Organization

8.4.1.1 Message Transmit Background

Modern application layer software is built upon two fundamental assumptions:

1. Any CAN node is able to send out a stream of scheduled messages without releasing the bus between the two messages. Such nodes arbitrate for the bus immediately after sending the previous message and only release the bus in case of lost arbitration.
2. The internal message queue within any CAN node is organized so the highest priority message is sent out first when more than one message is ready to be sent.

This behavior *cannot* be achieved with a single transmit buffer. That buffer must be reloaded immediately after the previous message is sent. The loading process lasts a finite amount of time. It must be completed within the inter-frame sequence (IFS)[1] in order to send an uninterrupted stream of messages. Even if this is feasible for limited CAN bus speeds, it requires the CPU react with short latencies to the transmit interrupt.

A double-buffer scheme uncouples the reloading of the transmit buffer from the actual message sending thereby reducing the reactiveness requirements on the CPU. It is possible problems could arise if a sent message has finished while the CPU is reloading the second buffer. In this instance there would be no buffer ready for transmission, releasing the bus.

At least three transmit buffers are required to meet the first of the above requirements under all circumstances. The CAN has three transmit buffers.

The second requirement calls for internal CAN prioritization, implementing the *local priority* concept described in the next section.

1. Reference the *Bosch CAN 2.0A/B Protocol Specification*, September 1991.

8.4.1.2 Transmit Structures

The CAN has a triple transmit buffer scheme. The transmit scheme allows multiple messages to be established in advance, achieving an optimized real-time performance. The three buffers are arranged as shown in **Figure 8-2.**

All three buffers have a 13-byte data structure similar to the outline of the receive buffers. See **Section 8.7.14, "Programmer's Model of Message Storage".** An additional Transmit Buffer Priority Register (TBPR) contains an 8-bit local Priority (PRIO) field. See **Section 8.7.18, "Transmit Buffer Priority Register (TBPR)".**

To transmit a message, the CPU must identify an available transmit buffer indicated by a set Transmitter Buffer Empty (TXE[2:0]) flag. See **Section 8.7.7, "CAN Transmitter Flag Register (CANTFLG)".**

The CPU then stores the identifier, the control bits, and the data content into one of the transmit buffers. Finally, the buffer is flagged as ready for transmission by clearing the associated TXE flag.

Next, the CAN schedules the message for transmission. Signals are sent to the successful transmission of the buffer by setting the associated TXE flag. A transmit interrupt is generated[2] when TXE[2:0] is set and can be used to drive the application software to re-load the buffer. Refer to **Section 8.9, "Interrupt Operation"** for interrupt operation.

If more than one buffer is scheduled for transmission when the CAN bus becomes available for arbitration, the CAN uses the local priority setting of the three buffers to determine the prioritization. For this purpose, every transmit buffer has an 8-bit PRIO. The application software programs this field when the message is set-up. The local priority reflects the order of this particular message relative to the set of messages being transmitted from this node. The *lowest binary value* of the PRIO is defined to be the *highest priority*. The internal scheduling process takes place whenever the CAN arbitrates for the bus. This is also the case after the occurrence of a transmission error.

When a high priority message is scheduled by the application software, it may become necessary to abort a lower priority message not yet transmitted, in one of the three transmit buffers. Messages already in transmission *cannot* be aborted, so termination must be requested by setting the corresponding Abort Request (ABTRQ) bit. See **Section 8.7.8, "CAN Transmitter Control Register (CANTCR)".** The CAN then permits the request, when possible, by:

- Setting the corresponding abort acknowledge flag (ABTAK) in the CANTFLG register
- Setting the associated TXE flag to release the buffer

2. The transmit interrupt occurs only if not masked. A polling scheme can be applied on TXEx also.

- Generating a transmit interrupt

The transmit interrupt handler software can tell from the setting of the ABTAK flag whether the message was aborted ABTAK equals 1, or when sent ABTAK equals 0.

8.4.1.3 Receive Structures

Received messages are stored in a two-stage input FIFO. The two message buffers are alternately mapped into a single memory area. See **Figure 8-4.** While the Background Receive Buffer (RxBG) is exclusively associated with the CAN, the Foreground Receive Buffer (RxFG) is addressed by the CPU. This scheme simplifies the handler software because only one address area is applicable for the receive process. Users *cannot* read/write to the RxBG buffer.

Both buffers have a size of 13 bytes for storage of the CAN control bits:

- Identifier
- Standard
- Extended
- Data Contents

See details in **Section 8.7.14, "Programmer's Model of Message Storage".**[3]

The Receiver Full (RXF) flag signals the status of the Foreground Receive Buffer. See **Section 8.7.5, "CAN Receiver Flag Register (CANRFLG)".** A flag is set when the buffer contains a correctly received message with a matching identifier.

Each message is checked to see if it passes the reception filter. See **Section 8.4.1.4, "Identifier Acceptance Filter".** It is written in parallel into RxBG. CAN copies the content of RxBG into RxFG,[4] sets the RXF flag, and generates a receive interrupt to the CPU.[5] See **Section 8.9, "Interrupt Operation".** Receive handler will read the received message from RxFG, before resetting the RXF flag. Reset of the RXF flag acknowledges the interrupt, in order to release the foreground buffer. A new message can follow immediately after the IFS field of the CAN frame is received into RxBG. The over-writing of the background buffer is independent of the identifier filter function.

When the CAN module is transmitting, the CAN receives its own transmitted messages into the background receive buffer RxBG. However, it does *not:*

- Overwrite RxFG
- Generate a receive interrupt

3. *Bosch CAN 2.0A/B Protocol Specification*, September 1991.
4. Only if the RXF Flag is not set.
5. The receive interrupt occurs only if not masked. A polling scheme can be applied on RXF also.

- Acknowledge its own messages on the CAN bus

There is an exception to this rule. It is in Loop Back mode where the CAN treats its own messages exactly like all other incoming messages. See **Section 8.7.3, "CAN Bus Timing Register 0 (CANBTR0)".** The CAN receives its own transmitted messages in the event it loses arbitration.[6] If arbitration is lost, the CAN must be prepared to become a receiver.

An overrun condition occurs when both message buffers are filled. Overrun messages are discarded, indicating an error interrupt. The CAN is still able to transmit messages with both receive message buffers filled; however, all incoming messages are discarded. See **Section 8.9, "Interrupt Operation"**.

8.4.1.4 Identifier Acceptance Filter

The CAN Identifier Acceptance registers define the acceptable patterns of the standard or extended identifier, ID[10:0] or ID[28:0] bits. See **Section 8.7.12, "CAN Identifier Acceptance Registers (CANIDAR0–7)"**. Any of these bits can be marked *don't care* in the CAN Identifier Mask registers. See **Section 8.7.13, "CAN Identifier Mask Registers (CANIDMR0–7)".**

A filter hit is indicated to the application software by setting the Receive Buffer Full (RXF) flag and the identifier hit flags in the CANIDAC register. See **Section 8.7.9, "CAN Identifier Acceptance Control Register (CANIDAC)"**. These identifier hit F flags (IDHIT[2:0]) clearly identify the filter section causing the acceptance. They simplify the application software's task to identify the cause of the receiver interrupt. In case more than one hit occurs, two or more filters match, the lower hit has priority.

A very flexible programmable, and generic identifier acceptance filter has been introduced reducing the CPU interrupt loading. The filter is programmable to operate in four different modes:[7]

1. Two identifier acceptance filters, each to be applied to:
   — The full 29 bits of the extended identifier and to the following bits of the CAN 2.0B frame: Remote Transmission Request (RTR), Identifier Extension (IDE), and Substitute Remote Request (SRR)
   — The 11 bits of the standard identifier plus the RTR and IDE bits of the CAN 2.0A/B messages. This mode implements two filters for a full length CAN 2.0B-compliant extended identifier [8]

6. Reference the *Bosch CAN 2.0A/B Protocol Specification,* September 1991.
7. For a better understanding of references made within Filter mode description, reference the *Bosch Protocol Specification,* September 1991 detailing the CAN 2.0A/B protocol.
8. Although this mode can be used for standard identifiers, it is recommended to use the four or eight identifier acceptance filters for standard identifiers.

Figure 8-5 illustrates how the first 32-bit filter bank (CANIDAR0–3 and CANIDMR0–3) produces a filter 0 hit. Similarly, the second filter bank (CANIDAR4–7 and CANIDMR4–7) produces a filter 1 hit

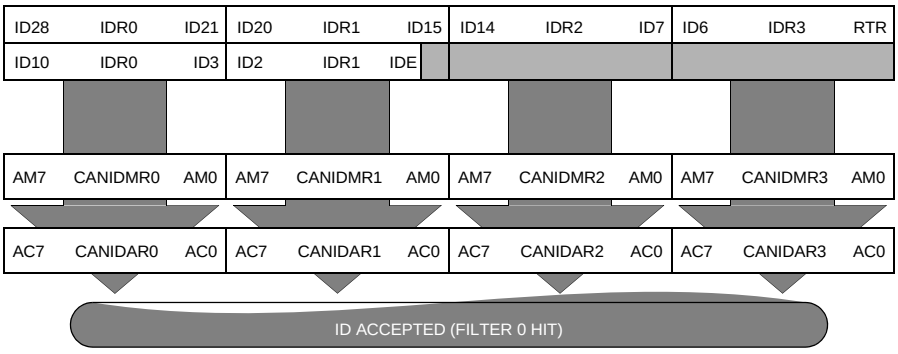


Figure 8-3. 32-Bit Mask Identifier Acceptance Filter

2. Four identifier acceptance filters, each to be applied to:
   - The 14MSBs of the extended identifier plus the SRR and IDE bits of CAN 2.0B messages
   - The 11 bits of the standard identifier, the RTR and IDE bits of CAN 2.0A/B messages

Figure 8-6 illustrates how the first 32-bit filter bank (CANIDAR0–3 and CANIDMR0–3) produces filter 0 and 1 hit. Similarly, the second filter bank (CANIDAR4–7 and CANIDMR4–7) produces filter two and three hits.

3. Eight identifier acceptance filters, each to be applied to the first eight bits of the identifier. This mode implements eight independent filters for the first eight bits of a CAN 2.0B-compliant standard identifier or a CAN 2.0B-compliant extended identifier.

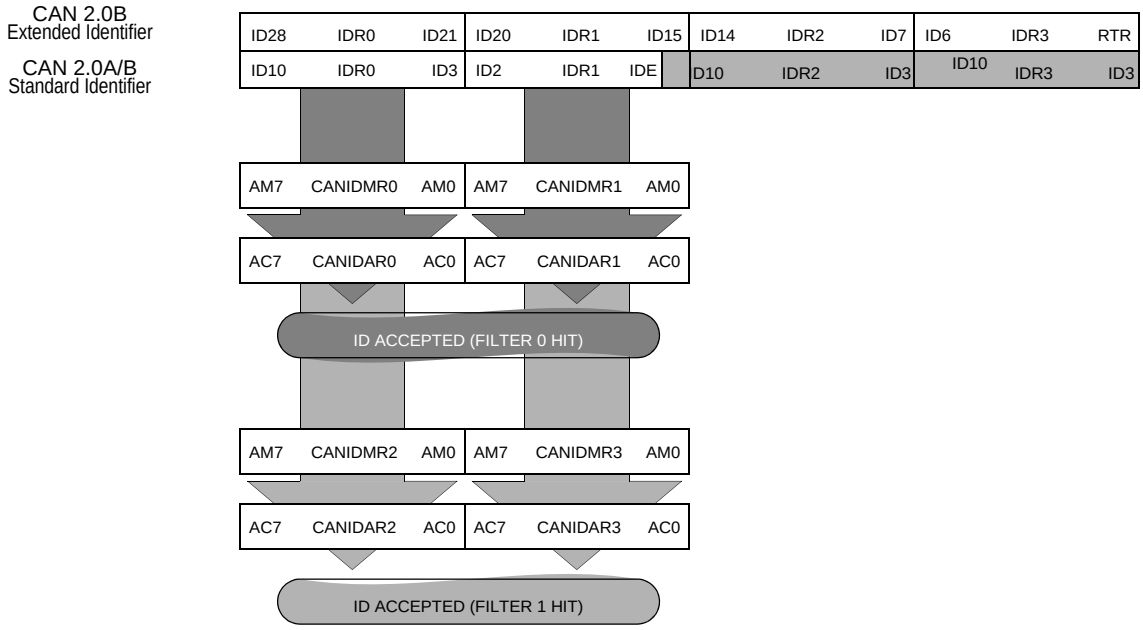


Figure 8-4. 16-Bit Maskable Identifier Acceptance Filters

Figure 8-7 displays how the first 32-bit filter bank (CANIDAR0–3 and CANIDMR0–3) produces filter zero to three hits. Similarly, the second filter bank (CANIDAR4–7 and CANIDMR4–7) produces filter four to seven hits.

4. Closed filter. No CAN message is copied into the foreground buffer RxFG. The RXF flag is never set.

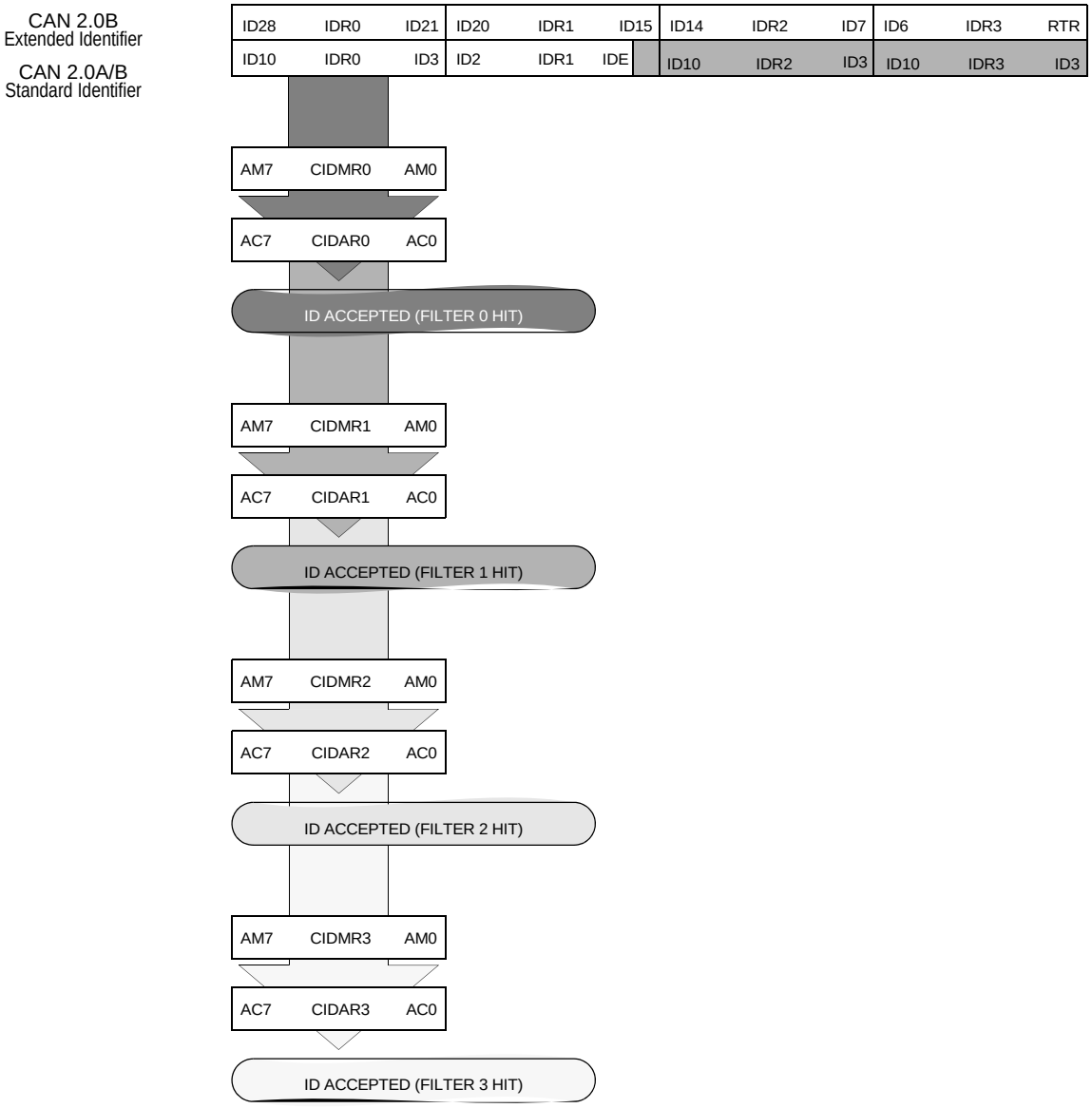


Figure 8-5. 8-Bit Maskable Identifier Acceptance Filters

8.4.2 Protocol Violation Protection

CAN protects from accidentally violating the CAN protocol through programming errors. The protection logic implements these features:

- Receive and transmit error counters *cannot* be modified or otherwise manipulated
- Registers controlling the configuration of the CAN *cannot* be modified while the CAN is online. The SFTRES bit in the CANCTL0 register serves as a lock protecting the following registers: See **Section 8.7, "Register Definitions".**

  — CAN Control 1 (CANCTL1) register

— CAN Bus Timing 0 and 1 (CANBTR0, CANBTR1) registers

— CAN Identifier Acceptance Control (CANIDAC) register

— CAN Identifier Acceptance (CANIDAR0–7) registers

— CAN Identifier Mask(CANIDMR0–7) registers

- The MSCAN_TX pin is forced to a recessive state when the CAN goes into any of the low power modes. See **Section 8.8.4, "Sleep Mode"** through **Section 8.8.6.**
- MSCAN_TX pin is an open-drain output. System designers using CAN should have external pull-up on MSCAN_TX
- MSCAN_RX input has an internal pull-up
- As further protection against inadvertently disabling the CAN, the enable bit, CANE, is only write-capable once in normal modes

8.4.3 Clock System

Figure 8-8 illustrates the structure of the CAN clock generation circuitry. With this flexible clocking scheme, the CAN is able to handle CAN bus rates ranging from 10 Kbps up to 1Mbps.

The Clock Source (CLKSRC) bit in the CANCTL1 register defines whether the CAN is connected to the output of the crystal oscillator, EXTALi, or to the IPBus clock. See **Section 8.7.2.** The muxing of these two clocks is performed inside CAN. The clock source and generation must be selected to meet the tight oscillator tolerance requirements of the CAN protocol, up to 0.4 percent accumulated jitter. Additionally, for high CAN bus rates of 1Mbps, a 50 percent duty cycle of the clock is required. When using the IPBus clock these tolerances are met.

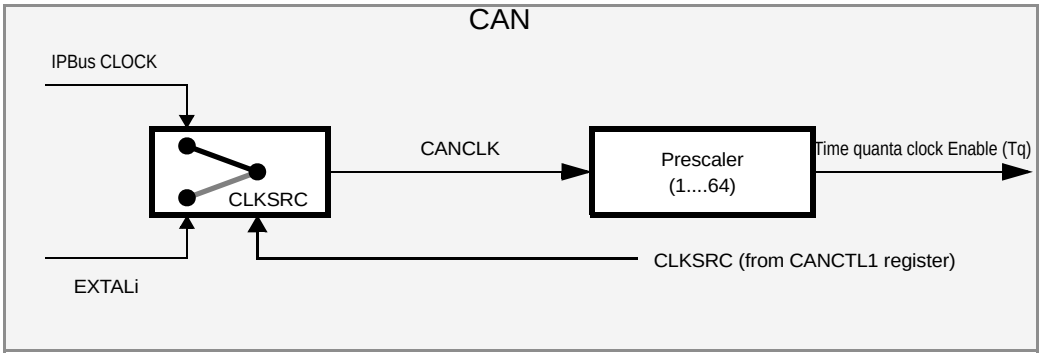


Figure 8-6. CAN Clocking Scheme

Note: Selecting the IPBus clock source will result in the least amount of clock jitter, greater noise immunity, and the most reliable data transfer. The IPBus clock is the recommended clock source for all rates. It must be selected for rates above 500Kbit/sec. The CLKSRC bit in the CANTCL1 register allows a choice between the IPBus or EXTEL clocks as the CAN clock.

A programmable prescaler generates the time quanta (Tq) clock from CANCLK. A time quantum is the atomic unit of time handled by the CAN.

$$f_{Tq} = \frac{f_{CANCLK}}{(Prescaler\ \ value)}$$

Note: For the 56F803, 56F805, and 56F807, the time quanta signal is internally used as a clock enable. CANCLK is the primary clock of the CAN operation.

A bit time is subdivided into three segments. Please reference **Figure 8-10:**

- SYNC_SEG—This segment has a fixed length of one time quantum. Signal edges are expected to happen within this section.
- Time segment 1—This segment includes the PROP_SEG and the PHASE_SEG1 of the CAN standard. It can be programmed by setting the parameter TSEG1 to consist of four to 16 time quanta.
- Time segment 2—This segment represents the PHASE_SEG2 of the CAN standard. It can be programmed by setting the TSEG2 parameter to be two to eight time quanta long.

$$Bit\ \ Rate = \frac{f_{Tq}}{(number\ \ of\ \ Time\ \ Quanta)}$$

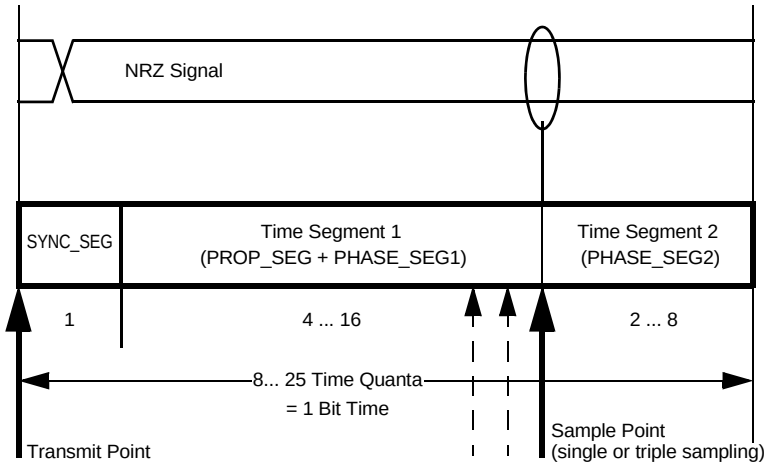


Figure 8-7. Segments Within the Bit Time

Table 8-1. 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 in Receive mode 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.

The synchronization jump width[9] (SJW) can be programmed in a range of one to four time quanta by setting the SJW parameter.

The above parameters are set by programming the CAN bus timing registers (CANBTR0 and CANBTR1). See **Section 8.7.3, "CAN Bus Timing Register 0 (CANBTR0)"** and **Section 8.7.6.**

Figure 8-2 provides an overview of the CAN-compliant segment settings and their related parameter values.

Note: Assure bit time settings are in compliance with the CAN standard.

Table 8-2. CAN Time Segment Settings When CLKSRC=0 (EXTAL_CLK)

(CLKSRC = 0) EXTAL_CLK ~8MHz)					
Prescaler Value (P (= BRP + 1))	**Time Segment 1**	**TSEG1**	**Time Segment 2**	**TSEG2**	**Synchronization Jump Width**
> 1	5 .. 10	4 .. 9	2	1	1 .. 2
> 1	4 .. 11	3 .. 10	3	2	1 .. 3
> 1	5 .. 12	4 .. 11	4	3	1 .. 4
> 1	6 .. 13	5 .. 12	5	4	1 .. 4
> 1	7 .. 14	6 .. 13	6	5	1 .. 4
> 1	8 .. 15	7 .. 14	7	6	1 .. 4

9. Reference the *Bosch CAN 2.0A/B Protocol Specification*, September 1991 for bit timing.

Table 8-3. CAN Time Segment Settings when CLKSRC = 1 (IPBus Clock)

CLKSRC = 1 (IPBus Clock >= 8MHz)						
Prescaler Value (P (= BRP + 1))	**Time Segment 1**	**TSEG1**	**Time Segment 2**	**TSEG2**	**Synchronization Jump Width**	**SJW**
= 1	4 .. 11	3 .. 10	3	2	1 .. 3	0 .. 2
> 1	5 .. 10	4 .. 9	2	1	1 .. 2	0 .. 1
Any	4 .. 11	3 .. 10	3	2	1 .. 3	0 .. 2
Any	5 .. 12	4 .. 11	4	3	1 .. 4	0 .. 3
Any	6 .. 13	5 .. 12	5	4	1 .. 4	0 .. 3
Any	7 .. 14	6 .. 13	6	5	1 .. 4	0 .. 3
Any	8 .. 15	7 .. 14	7	6	1 .. 4	0 .. 3
Any	9 .. 16	8 .. 15	8	7	1 .. 4	0 .. 3

8.5 Operating Modes

8.5.1 Normal Modes

The CAN module behaves as described within this specification in all Normal modes.

8.5.2 Special Modes

The CAN module has no special accessible modes.

8.5.3 Emulation Modes

In all emulation modes, the CAN module behaves just like Normal modes as described within this specification.

8.5.4 Security Modes

The CAN module has no security feature.

8.6 Pin Definitions

The CAN uses two external pins: Receive Input MSCAN_RX and Transmit output MSCAN_TX. The MSCAN_TX output pin represents the logic level on the CAN:

- 0 = Dominant state
- 1 = Recessive state

When the CAN is enabled, (CANE = 1) via the CANCTL1 register, MSCAN_RX is the dedicated input pin and MSCAN_TX is the dedicated output pin, or open-drain output. The MSCAN_RX pin has an internal pull-up.

A typical CAN system with CAN is illustrated in **Table 8-8.** Each CAN station is connected physically to the CAN bus lines through a transceiver chip. The transceiver is capable of driving the large current required by the CAN bus. It has current protection against defective CAN or defective stations.

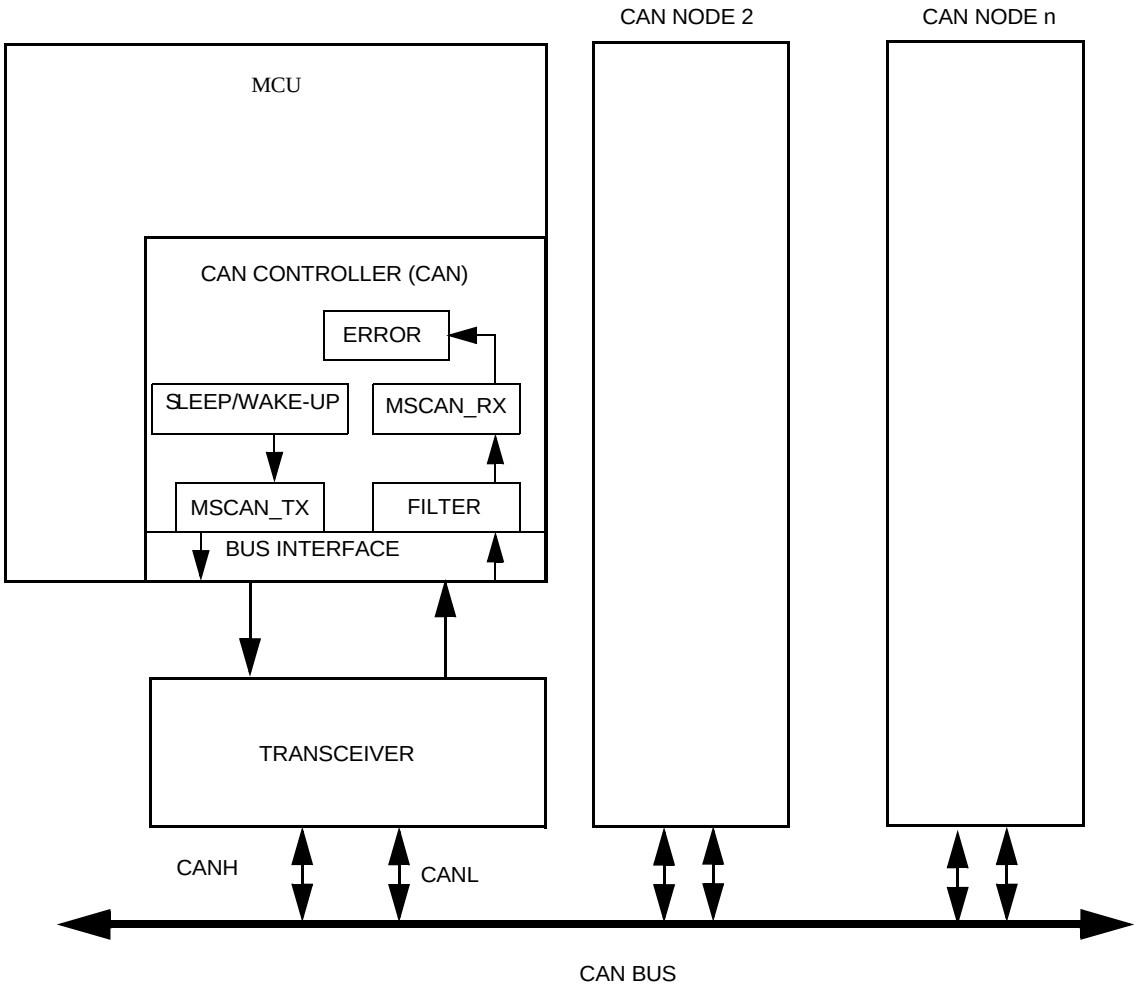


Figure 8-8. CAN System

8.7 Register Definitions

The CAN occupies 128 words in the memory space starting at CAN_BASE, defined in **Table 3-11.** The register decode map is fixed, beginning at the first address of the module address offset. **Table 8-4** illustrates the organization of the registers.

Note: All CAN registers are 16-bits wide with the most significant eight bits tied to zero. Therefore, note the lower eight significant bits as the relevant configuration of status bits.

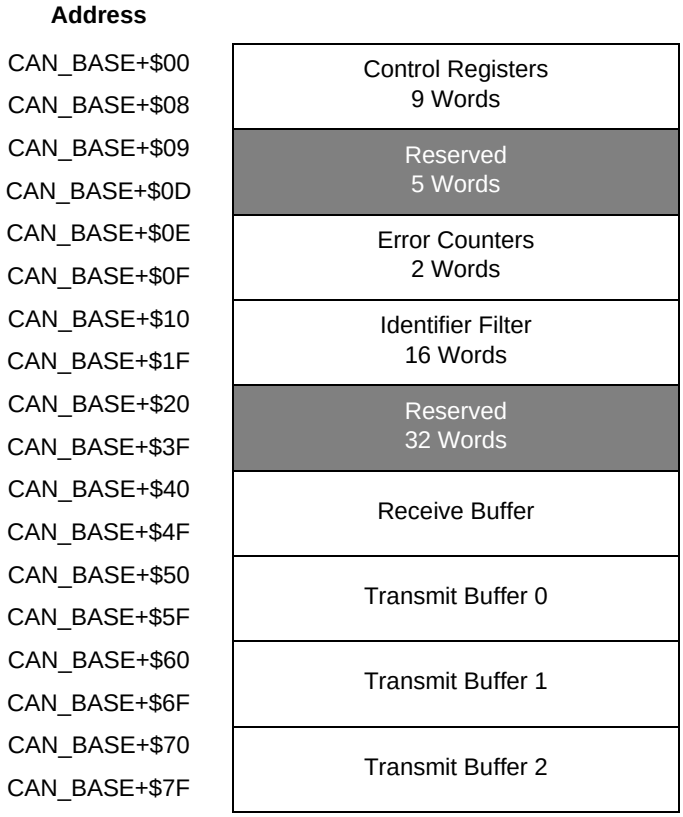


Figure 8-9. CAN Register Organization

Section 8-9 exhibits individual registers associated with the CAN, and their relative offset from the base address. The detailed register descriptions follow in the order they appear in the register map.

Reserved bits within a register are always read as 0 and a write is not implemented. Reserved functions are indicated by shaded bits.

Table 8-4. CAN Memory Map

Device	Peripheral	Address
803/805	CAN_BASE	$0D80
807	CAN_BASE	$1180

This section details all registers and register bits in the CAN module.

Note: All bits of all registers in this module are completely synchronized to internal clocks during a register read.

Table 8-5. CAN Register Summary

Address Offset	Register Acronym	Register Name	Access Type	Register Location
Base + $0	CANCTL0	Control Register 0	Read/Write	Section 8.7.1
Base + $1	CANCTL1	Control Register 1	Read/Write	Section 8.7.2
Base + $2	CANBTR0	Bus Timing Register 0	Read/Write	Section 8.7.3
Base + $3	CANBTR1	Bus Timing Register 1	Read/Write	Section 8.7.4
Base + $4	CANRFLG	Receiver Flag Register	Read/Write	Section 8.7.5
Base + $5	CANRIER	Receiver Interrupt Enable Register	Read/Write	Section 8.7.6
Base + $6	CANTFLG	Transmitter Flag Register	Read/Write	Section 8.7.7
Base + $7	CANTCR	Transmitter Control Register	Read/Write	Section 8.7.8
Base + $8	CANIDAC	Identifier Acceptance Control Reg.	Read/Write	Section 8.7.9
Base + $E	CANRXERR	Receive Error Counter Register	Read/Write	Section 8.7.10
Base + $F	CANTXERR	Transmit Error Counter Register	Read/Write	Section 8.7.11
Base + $10 – $13	CANIDAR0-3	Identifier Acceptance Reg (1st Bank)	Read/Write	Section 8.7.12
Base + $14 – $17	CANIDMR0-4	Identifier Mask Register (1st Bank)	Read/Write	Section 8.7.13
Base + $18 – $1B	CANIDAR4-7	Identifier Acceptance Reg (2nd Bank)	Read/Write	Section 8.7.12
Base + $1C – $1F	CANIDMR4-7	Identifier Mask Register (2nd Bank)	Read/Write	Section 8.7.13
Base + $40 – $43	CANRBIDR0-3	Receive Buffer Identifier Registers 0-3	Read/Write	Section 8.7.15
Base + $44 – $4B	CANRBDSR0-7	Receive Buffer Data Segment Reg.0-7	Read/Write	Section 8.7.16
Base + $4C	CANRBDLR	Receive Buffer Data Length Register	Read/Write	Section 8.7.17
Base + $50 – $53	CANTB0IDR0-3	Transmit Buffer 0 Identifier Registers 0-3	Read/Write	Section 8.7.18
Base + $54 – $5B	CANTB0DSR0 -7	Transmit Buffer 0 Data Segment Regs. 0-7	Read/Write	Section 8.7.15
Base + $5C	CANTB0DLR	Transmit Buffer 0 Data Length Register	Read/Write	Section 8.7.17
Base + $5D	CANTB0TBPR	Transmit Buffer 0 Transmit Buffer Priority	Read/Write	Section 8.7.18
Base + $60 – $63	CANTB1IDR0-3	Transmit Buffer 1Identifier Registers 0-3	Read/Write	Section 8.7.15

Table 8-5. CAN Register Summary (Continued)

Address Offset	Register Acronym	Register Name	Access Type	Register Location
Base + $64 – $6B	CANTB1DSR0-7	Transmit Buffer 1 Data Segment Regs. 0-7	Read/Write	Section 8.7.16
Base + $6C	CANTB1DLR	Transmit Buffer 1 Data Length Register	Read/Write	Section 8.7.17
Base + $6D	CANTB1TBPR	Transmit Buffer 1 Transmit Buffer Priority	Read/Write	Section 8.7.18
Base + $70 – $73	CANTB2IDR0-3	Transmit Buffer 2 Identifier Registers 0-3	Read/Write	Section 8.7.15
Base + $74 – $7B	CANTB2DSR0-7	Transmit Buffer 2 Data Segment Regs. 0-3	Read/Write	Section 8.7.16
Base + $7C	CANTB2DLR	Transmit Buffer 2 Data Length Register	Read/Write	Section 8.7.17
Base + $7D	CANTB2TBPR	Transmit Buffer 2 Transmit Buffer Priority	Read/Write	Section 8.7.18

Bit fields of the CAN 98 registers are illustrated in **Table 8-10.** Details of each follow.

Add. Offset	Register Name		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$0	CANCTL0	R	0	0	0	0	0	0	0	0	RXFRM	RXACT	CSWAI	SYNCH	0	SLPAK	SLPRQ	SFTRES
		W																
$1	CANCTL1	R	0	0	0	0	0	0	0	0	CANE	0	0	0	0	LOOPB	WUPM	CLKSRC
		W																
$2	CANBTR0	R	0	0	0	0	0	0	0	0	SJW1	SJW0	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0
		W																
$3	CANBTR1	R	0	0	0	0	0	0	0	0	SAMP	TSEG 22	TSEG 21	TSEG 20	TSEG 13	TSEG 12	TSEG 11	TSEG 10
		W																
$4	CANRFLG	R	0	0	0	0	0	0	0	0	WUPIF	RWRNIF	TWRNIF	RERRIF	TERRIF	BOFFIF	OVRIF	RXF
		W																
$5	CANRIER	R	0	0	0	0	0	0	0	0	WUPIE	RWRNIE	TWRNIE	RERRIE	TERRIE	BOFFIE	OVRIE	RXFIE
		W																
$6	CANTFLG	R	0	0	0	0	0	0	0	0	0	ABTAK2	ABTAK1	ABTAK0	0	TXE2	TXE1	TXE0
		W																
$7	CANTCR	R	0	0	0	0	0	0	0	0	0	ABTRQ2	ABTRQ1	ABTRQ0	0	TXEIE2	TXEIE1	TXEIE0
		W																
$8	CANIDAC	R	0	0	0	0	0	0	0	0	0	0	IDAM1	IDAM0	0	IDHIT2	IDHIT1	IDHIT0
		W																
Reserved $9 - $D																		
$E	CANRXERR	R	0	0	0	0	0	0	0	0	RXERR 7	RXERR 6	RXERR 5	RXERR 4	RXERR 3	RXERR 2	RXERR 1	RXERR 0
		W																
$F	CANTXERR	R	0	0	0	0	0	0	0	0	TXERR 7	TXERR 6	TXERR 5	TXERR 4	TXERR 3	TXERR 2	TXERR 1	TXERR 0
		W																
$10	CANIDAR0	R	0	0	0	0	0	0	0	0	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
		W																
$11	CANIDAR1	R	0	0	0	0	0	0	0	0	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
		W																
$12	CANIDAR2	R	0	0	0	0	0	0	0	0	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
		W																
$13	CANIDAR3	R	0	0	0	0	0	0	0	0	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
		W																
$14	CANIDMR0	R	0	0	0	0	0	0	0	0	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
		W																

Figure 8-10. CAN Register Map

Add. Offset	Register Name		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$15	CANIDMR1	R	0	0	0	0	0	0	0	0	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
		W																
$16	CANDIMR2	R	0	0	0	0	0	0	0	0	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
		W																
$17	CANDIMR3	R	0	0	0	0	0	0	0	0	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
		W																
$18	CANIDAR4	R	0	0	0	0	0	0	0	0	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
		W																
$19	CANIDAR5	R	0	0	0	0	0	0	0	0	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
		W																
$1A	CANIDAR6	R	0	0	0	0	0	0	0	0	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
		W																
$1B	CANIDAR7	R	0	0	0	0	0	0	0	0	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
		W																
$1C	CANIDMR4	R	0	0	0	0	0	0	0	0	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
		W																
$1D	CANIDMR5	R	0	0	0	0	0	0	0	0	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
		W																
$1E	CANIDMR6	R	0	0	0	0	0	0	0	0	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
		W																
$1F	CANIDMR7	R	0	0	0	0	0	0	0	0	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
		W																
RESERVED $20 - $3F																		
Standard Identifier Mapping Receive Buffer, Identifier Registers 0-3																		
$40	RB_IDR0	R	0	0	0	0	0	0	0	0	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3
		W																
$41	RB_IDR1	R	0	0	0	0	0	0	0	0	ID2	ID1	ID0	RTR	IDE (=0)			
		W																
$42	RB_IDR2	R	0	0	0	0	0	0	0	0								
		W																
$43	RB_IDR3	R	0	0	0	0	0	0	0	0								
		W																
Extended Identifier Mapping Receive Buffer, Identifier Registers 0-3																		
$40	RB_IDR0	R	0	0	0	0	0	0	0	0	ID28	ID27	ID26	ID25	ID24	ID23	ID22	ID21
		W																
$41	RB_IDR1	R	0	0	0	0	0	0	0	0	ID20	ID19	ID18	SRR(=1)	IDE(=1)	ID17	ID16	ID15
		W																
$42	RB_IDR2	R	0	0	0	0	0	0	0	0	ID14	ID13	ID12	ID11	ID10	ID9	ID8	ID7
		W																
$43	RB_IDR3	R	0	0	0	0	0	0	0	0	ID6	ID5	ID4	ID3	ID2	ID1	ID0	RTR
		W																
Receive Buffer Data Segment Registers 0-7																		
$44	RB_DSR0	R	0	0	0	0	0	0	0	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
		W																
$45	RB_DSR1	R	0	0	0	0	0	0	0	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
		W																
$46	RB_DSR2	R	0	0	0	0	0	0	0	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
		W																
$47	RB_DSR3	R	0	0	0	0	0	0	0	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
		W																

Figure 8-10. CAN Register Map (Continued)

Add. Offset	Register Name		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$48	RB_DSR4	R	0	0	0	0	0	0	0	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
		W																
$49	RB_DSR5	R	0	0	0	0	0	0	0	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
		W																
$4A	RB_DSR6	R	0	0	0	0	0	0	0	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
		W																
$4B	RB_DSR7	R	0	0	0	0	0	0	0	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
		W																
$4C	RB_DLR	R	0	0	0	0	0	0	0	0	0	0	0	0	DLC3	DLC2	DLC1	DLC0
		W																
Standard Identifier Mapping Transmit Buffer 0, Identifier Registers 0-3																		
$50	TB0_IDR0	R	0	0	0	0	0	0	0	0	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3
		W																
$51	TB0_IDR1	R	0	0	0	0	0	0	0	0	ID2	ID1	ID0	RTR	IDE (=0)			
		W																
$52	TB0_IDR2	R	0	0	0	0	0	0	0	0								
		W																
$53	TB0_IDR3	R	0	0	0	0	0	0	0	0								
		W																
Extended Identifier Mapping, Transmit Buffer 0 Identifier Registers 0-3																		
$50	TB0_IDR0	R	0	0	0	0	0	0	0	0	ID28	ID27	ID26	ID25	ID24	ID23	ID22	ID21
		W																
$51	TB0_IDR1	R	0	0	0	0	0	0	0	0	ID20	ID19	ID18	SRR(=1)	IDE(=1)	ID17	ID16	ID15
		W																
$52	TB0_IDR2	R	0	0	0	0	0	0	0	0	ID14	ID13	ID12	ID11	ID10	ID9	ID86	ID7
		W																
$53	TB0_IDR3	R	0	0	0	0	0	0	0	0	ID6	ID5	ID4	ID3	ID2	ID1	ID0	RTR
		W																
Transmit Buffer 0 Data Segment Registers 0-7																		
$54	TB0_DSR0	R	0	0	0	0	0	0	0	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
		W																
$55	TB0_DSR1	R	0	0	0	0	0	0	0	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
		W																
$56	TB0_DSR2	R	0	0	0	0	0	0	0	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
		W																
$57	TB0_DSR3	R	0	0	0	0	0	0	0	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
		W																
$58	TB0_DSR4	R	0	0	0	0	0	0	0	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
		W																
$59	TB0_DSR5	R	0	0	0	0	0	0	0	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
		W																
$5A	TB0_DSR6	R	0	0	0	0	0	0	0	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
		W																
$5B	TB0_DSR7	R	0	0	0	0	0	0	0	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
		W																
$5C	TB0_DLR	R	0	0	0	0	0	0	0	0	0	0	0	0	DLC3	DLC2	DLC1	DLC0
		W																
$5D	TB0_TBPR	R	0	0	0	0	0	0	0	0	PRIO7	PRIO6	PRIO5	PRIO4	PRIO3	PRIO2	PRIO1	PRIO0
		W																
Standard Identifier Mapping, Transmit Buffer 1 Identifier Registers 0-3																		

Figure 8-10. CAN Register Map (Continued)

Add. Offset	Register Name		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$60	TB1-IDR0	R	0	0	0	0	0	0	0	0	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3
		W																
$61	TB1_IDR1	R	0	0	0	0	0	0	0	0	ID2	ID1	ID0	RTR	IDE (=0)			
		W																
$62	TB1_IDR2	R	0	0	0	0	0	0	0	0								
		W																
$63	TB1_IDR3	R	0	0	0	0	0	0	0	0								
		W																
Extended Identifier Mapping, Transmit Buffer 1 Identifier Registers 0-3																		
$60	TB1-IDR0	R	0	0	0	0	0	0	0	0	ID28	ID27	ID26	ID25	ID24	ID23	ID22	ID21
		W																
$61	TB1_IDR1	R	0	0	0	0	0	0	0	0	ID20	ID19	ID18	SRR(=1)	IDE(=1)	ID17	ID16	ID15
		W																
$62	TB1_IDR2	R	0	0	0	0	0	0	0	0	ID14	ID13	ID12	ID11	ID10	ID9	ID86	ID7
		W																
$63	TB1_IDR3	R	0	0	0	0	0	0	0	0	ID6	ID5	ID4	ID3	ID2	ID1	ID0	RTR
		W																
Transmit Buffer 1 Data Segment Registers 0-7																		
$64	TB1_DSR0	R	0	0	0	0	0	0	0	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
		W																
$65	TB1_DSR1	R	0	0	0	0	0	0	0	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
		W																
$66	TB1_DSR2	R	0	0	0	0	0	0	0	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
		W																
$67	TB1_DSR3	R	0	0	0	0	0	0	0	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
		W																
$68	TB1_DSR4	R	0	0	0	0	0	0	0	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
		W																
$69	TB1_DSR5	R	0	0	0	0	0	0	0	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
		W																
$6A	TB1_DSR6	R	0	0	0	0	0	0	0	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
		W																
$6B	TB1_DSR7	R	0	0	0	0	0	0	0	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
		W																
$6C	TB1_DLR	R	0	0	0	0	0	0	0	0	0	0	0	0	DLC3	DLC2	DLC1	DLC0
		W																
$6D	TB1_TBPR	R	0	0	0	0	0	0	0	0	PRIO7	PRIO6	PRIO5	PRIO4	PRIO3	PRIO2	PRIO1	PRIO0
		W																
Standard Identifier Mapping, Transmit Buffer 2 Identifier Registers 0-3																		
$70	TB2_IDR0	R	0	0	0	0	0	0	0	0	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3
		W																
$71	TB2_IDR1	R	0	0	0	0	0	0	0	0	ID2	ID1	ID0	RTR	IDE (=0)			
		W																
$72	TB2_IDR2	R	0	0	0	0	0	0	0	0								
		W																
$73	TB2_IDR3	R	0	0	0	0	0	0	0	0								
		W																
Extended Identifier Mapping, Transmit Buffer 2 Identifier Registers 0-3																		
$70	TB2_IDR0	R	0	0	0	0	0	0	0	0	ID28	ID27	ID26	ID25	ID24	ID23	ID22	ID21
		W																

Figure 8-10. CAN Register Map (Continued)

Add. Offset	Register Name		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$71	TB2_IDR1	R	0	0	0	0	0	0	0	0	ID20	ID19	ID18	SRR(=1)	IDE(=1)	ID17	ID16	ID15
		W																
$72	TB2_IDR2	R	0	0	0	0	0	0	0	0	ID14	ID13	ID12	ID11	ID10	ID9	ID86	ID7
		W																
$73	TB2_IDR3	R	0	0	0	0	0	0	0	0	ID6	ID5	ID4	ID3	ID2	ID1	ID0	RTR
		W																
Transmit Buffer 2 Data Segment Registers 0-7																		
$74	TB2_DSR0	R	0	0	0	0	0	0	0	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
		W																
$75	TB2_DSR1	R	0	0	0	0	0	0	0	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
		W																
$76	TB2_DSR2	R	0	0	0	0	0	0	0	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
		W																
$77	TB2_DSR3	R	0	0	0	0	0	0	0	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
		W																
$78	TB2_DSR4	R	0	0	0	0	0	0	0	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
		W																
$79	TB2_DSR5	R	0	0	0	0	0	0	0	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
		W																
$7A	TB2_DSR6	R	0	0	0	0	0	0	0	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
		W																
$7B	TB2_DSR7	R	0	0	0	0	0	0	0	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
		W																
$7C	TB2_DLR	R	0	0	0	0	0	0	0	0	0	0	0	0	DLC3	DLC2	DLC1	DLC0
		W																
$7D	TB2_TBPR	R	0	0	0	0	0	0	0	0	PRIO7	PRIO6	PRIO5	PRIO4	PRIO3	PRIO2	PRIO1	PRIO0
		W																

R	0	Read as 0
W		Reserved

Figure 8-10. CAN Register Map (Continued)

Note: Register address = base address + address offset, where the base address is defined in **Table 3-11** and the address offset is defined at the module level.

8.7.1 CAN Control Register 0 (CANCTL0)

The CANCTL0 register provides for various control of the CAN module.

These bits are read/write at any time, except when bits RXACT, SYNCH, and SLPAK are reserved. With bit RXFRM, a write of 1 clears the flag and a write of 0 is ignored.

CAN_BASE+$0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	0	0	0	RXFRM	RXACT	CSWAI	SYNCH	0	SLPAK	SLPRQ	SFTRES
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 8-11. CAN Control Register 0 (CANCTL0)

See Programmer Sheet on Appendix page A-41

Note: The CANCTL0 register is held in the Reset state when the SFTRES bit is set.

8.7.1.1 Reserved—Bits 15–8

This bit field is reserved or not implemented. It is read as 0 and cannot be modified by writing.

8.7.1.2 Received Frame Flag (RXFRM)—Bit 7

The Received Frame Flag (RXFRM) bit is read and clear only. It is set when a receiver has accepted a valid message correctly and independently of the filter configuration. Once set, it remains set until it is cleared by software or reset. To clear, write 1 to the bit. This bit is not valid in a Loop Back mode.

- 0 = No valid message was received since last clearing this flag
- 1 = A valid message was received since last clearing of this flag

Note: CAN must be in Run mode for this bit to set.

8.7.1.3 Receiver Active Status (RXACT)—Bit 6

Receiver active status flag indicates CAN is receiving a message. The flag is controlled by the receiver front end. This bit is not valid in Loop Back mode.

- 0 = CAN is transmitting or idle
- 1 = CAN is receiving a message, including when arbitration is lost

8.7.1.4 CAN Stops in Wait Mode (CSWAI)—Bit 5

Enabling this bit uses lower power consumption in Wait mode by disabling all the clocks at the bus interface to the CAN module.

- 0 = The module is not affected during Wait mode
- 1 = The module ceases to be clocked during Wait mode

8.7.1.5 Synchronized Status (SYNCH)—Bit 4

This flag indicates whether CAN is synchronized to the CAN bus. When it is synchronized, it is able to participate in the communication process. It is set and cleared by CAN.

- 0 = CAN is not synchronized to the CAN bus
- 1 = CAN is synchronized to the CAN bus

8.7.1.6 Reserved—Bit 3

This bit is reserved or not implemented. It can be read/written as 0.

8.7.1.7 Sleep Acknowledge (SLPAK)—Bit 2

The Sleep Acknowledge (SLPAK) flag indicates whether the CAN module is in internal Sleep mode. Refer to **Section 8.8.4.** It is used as a handshake for Sleep mode request. If the CAN detects bus activity while in Sleep mode, it clears the flag.

- 0 = Wake-Up – The CAN is not in Sleep mode
- 1 = Sleep – The CAN is in Sleep mode

8.7.1.8 Sleep Request–Go Into Sleep Mode (SLPRQ)—Bit 1

Sleep Request (SLPRQ) bit requests the CAN goes into an internal Power-Saving mode. Please see **Section 8.8.4.**

- 0 = Wake-Up – The CAN functions normally
- 1 = Sleep request – The CAN enters Sleep mode

8.7.1.9 Soft Reset (SFTRES)—Bit 0

When this bit is set by the CPU, the CAN immediately enters the Soft Reset State. Any ongoing transmission, or reception is quit and synchronization to the bus is lost. Please see **Section 8.8.5, "Soft Reset Mode"** for further details.

The following registers enter and stay in their hard Reset state:

- CANCTL0
- CANRFLG
- CANRIER
- CANTFLG
- CANTCR

These registers can only be modified by the CPU when the CAN is in Soft Reset state. Values of the error counters are not affected by soft reset.

- CANCTL1
- CANBTR0
- CANBTR1
- CANIDAC
- CANIDAR0–7
- CANIDMR0–7

When the Soft Reset (SFTRST) bit is cleared by the CPU, the CAN attempts synchronization to the CAN bus. If the CAN is not in Bus Off state, it synchronizes after 11 recessive bits on the bus; if the CAN is in Bus Off state it continues to wait for 128 occurrences of 11 recessive bits.

Clearing SFTRES and writing to other bits in CANCTL0 must be fulfilled in separate instructions.

- 0 = Normal operation
- 1 = CAN in Soft Reset state

8.7.2 CAN Control Register 1 (CANCTL1)

The CAN Control 1 (CANCTL1) register provides for various control of the CAN module as described below.

These bits are read/write at any time when SFTRES = 1, except CANE. It is write once in Normal modes and any time in special modes when SFTRES = 1.

CAN_BASE+$1	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	0	0	0	CANE	0	0	0	0	LOOPB	WUPM	CLKSRC
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-12. CAN Control Register 1 (CANCTL1)

See Programmer Sheet on Appendix page A-42

8.7.2.1 Reserved—Bits 15–8

This bit field is reserved or not implemented. It is read as 0 and cannot be modified by writing.

8.7.2.2 CAN Enable (CANE)—Bit 7

- 0 = The CAN module is disabled
- 1 = The CAN module is enabled

Note: Care should be taken when the CAN is disabled. It is recommended to have a pull-up on the MSCAN_TX pin either internally or externally ensuring the MSCAN_TX pin is in a recessive state (Logic 1) and avoiding violation of the CAN protocol.

8.7.2.3 Reserved—Bits 6–3

This bit field is reserved or not implemented. It is read as 0 and cannot be modified by writing.

8.7.2.4 Loop Back Self Test Mode (LOOPB)—Bit 2

When this bit is set, the CAN performs an internal loop back used for self test operation. The bit stream output of the transmitter is fed back to the receiver internally. The MSCAN_RX input pin is ignored while the MSCAN_TX output goes to the recessive state (Logic 1). The CAN behaves as it does when normally transmitting. It treats its own transmitted message as a message received from a remote node. In this state, the CAN ignores the bit sent during the ACK slot in the CAN Frame Acknowledge field ensuring proper reception of its own message. Both transmit and receive interrupts are generated.

8.7.2.5 Wake-Up Mode (WUPM)—Bit 1

This bit defines whether the integrated low-pass filter is applied to protect the CAN from artificial wake-up. Refer to **Section 8.8.4.**

- 0 = CAN wakes the CPU after any recessive to dominant edge on the CAN bus
- 1 = CAN wakes the CPU only in case of a dominant pulse on the bus with a length of T_{wup}

8.7.2.6 CAN Clock Source (CLKSRC)—Bit 0

The CAN Clock Source (CLKSRC) bit defines the clock source for the CAN module. It selects between IPBus clock and crystal oscillator clock EXTAL_CLK. Refer to **Section 8.4.3, "Clock System"** and **Table 8-6**

- 0 = The CAN clock source is EXTAL_CLK
- 1 = The CAN clock source is IPBus clock

Note: Selecting the IPBus clock source will result in the least amount of clock jitter, greater noise immunity, and the most reliable data transfer. The IPBus clock is the recommended clock source for all rates. It must be selected for rates above 500Kbit/sec.

8.7.3 CAN Bus Timing Register 0 (CANBTR0)

The CANBTR0 register provides various bus timing control of the CAN module. These bits are read/write at any time SFTRES = 1.

CAN_BASE+$2	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	0	0	0	SJW1	SJW0	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-13. CAN Bus Timing Register 0 (CANBTR0)

See Programmer Sheet on Appendix page A-43

8.7.3.1 Reserved—Bits 15–8

This bit field is reserved or not implemented. It is read as 0 and cannot be modified by writing.

8.7.3.2 Synchronization Jump Width (SJW)—Bits 7–6

The synchronization jump width defines the maximum number of time quanta (Tq) clock cycles a bit can be shortened or lengthened to achieve resynchronization to data transitions on the bus. Please refer to **Table 8-6**

Table 8-6. Synchronization Jump Width

SJW1	SJW0	Synchronization Jump Width
0	0	1 Tq clock cycle
0	1	2 Tq clock cycles
1	0	3 Tq clock cycles
1	1	4 Tq clock cycles

8.7.3.3 Baud Rate Prescaler (BRP)—Bits 5–0

These bits determine the time quanta (Tq) clock, used to build the individual bit timing. Please see **Table 8-4.** Prescaler (P) is equal to the binary representation of BRP5, BRP4, BRP3, BRP2, BRP1, and BRP0 plus 1. See examples in **Table 8-7.**

Table 8-7. Examples of Baud Rate Prescaler

BRP5	BRP4	BRP3	BRP2	BRP1	BRP0	Prescaler value (P)
0	0	0	0	0	0	1
0	0	0	0	0	1	2
0	0	0	0	1	0	3
0	0	0	0	1	1	4
1	1	1	1	1	0	63
1	1	1	1	1	1	64

Note: When the baud rate prescaler value (P) = 1(e.g.: CANBTR0 = binary bits: 00000000_xx000000) for getting a bit time of eight time quanta (8 Tq). For correct operation program for CANBTR1 = $0023 or $00A3 *with no other possible combinations* of $0014 or $0094. The correct combination translates to time segment 1 (TSEG1) = 4 Tq and time segment 2 (TSEG2) = 3 Tq.

8.7.4 CAN Bus Timing Register 1 (CANBTR1)

The CANBTR1 register provides various bus timing controls of the CAN module. See **Table 8-14.** It can read/write at any time when SFTRES = 1.

CAN_BASE+$3	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	0	0	0	SAMP	TSEG 22	TSEG 21	TSEG 20	TSEG 13	TSEG 12	TSEG 11	TSEG 10
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-14. CAN Bus Timing Register 1 (CANBTR1)

See Programmer Sheet on Appendix page A-44

8.7.4.1 Reserved—Bits 15–8

This bit field is reserved or not implemented. It is read as 0 and cannot be modified by writing.

8.7.4.2 Sampling (SAMP)—Bit 7

This bit determines the number of samples of the serial bus to be taken per bit time. If set, three samples per bit are taken; the regular one, a sample point, and two preceding samples using a majority rule. For higher bit rates, it is recommended SAMP be cleared, meaning only one sample is taken per bit.

- 0 = One sample per bit
- 1 = Three samples per bit[10]

8.7.4.3 Time Segment 2 (TSEG22–TSEG20)—Bits 6–4

Time segments within the bit time fix the number of clock cycles per bit time and the location of the sample point. Please see **Section 8-10.** Time Segment 2 (TSEG2) values are programmable as illustrated in **Table 8-8.**

Table 8-8. Time Segment 2 Values

TSEG22	TSEG21	TSEG20	Time segment 2
0	0	0	1 Tq clock cycle[1]
0	0	1	2 Tq clock cycles
0	1	0	3 Tq clock cycles
0	1	1	4 Tq clock cycles
1	0	0	5 Tq clock cycles
1	0	1	6 Tq clock cycles
1	1	0	7 Tq clock cycles
1	1	1	8 Tq clock cycles

1.This setting is not valid. Please refer to **Table 8-2. CAN Standard Compliance Bit Time Segment Settings** for valid settings.

10.In this case, PHASE_SEG1 must be at least two time quanta.

8.7.4.4 Time Segment 1 (TSEG13–TSEG10)—Bits 3–0

Time segments within the bit time fix the number of clock cycles per bit time and the location of the sample point. Please refer to **Section 8-10.**

Time Segment1 (TSEG1) values are programmable as shown in **Table 8-9.**

Table 8-9. Time Segment 1 Values

TSEG13	TSEG12	TSEG11	TSEG10	Time segment 1
0	0	0	0	1 Tq clock cycle[1]
0	0	0	1	2 Tq clock cycles[1]
0	0	1	0	3 Tq clock cycles[1]
0	0	1	1	4 Tq clock cycles
0	1	0	0	5 Tq clock cycles
0	1	0	1	6 Tq clock cycles
0	1	1	0	7 Tq clock cycles
0	1	1	1	8 Tq clock cycles
1	0	0	0	9 Tq clock cycles
1	0	0	1	10 Tq clock cycles
1	0	1	0	11 Tq clock cycles
1	0	1	1	12 Tq clock cycles
1	1	0	0	13 Tq clock cycles
1.	1	0.	1	14 Tq clock cycles
1	1	1	0	15 Tq clock cycles
1	1	1	1	16 Tq clock cycles

1.This setting is not valid. Please refer to **Table 8-2** for valid settings.

The bit time is determined by the oscillator frequency, the baud rate prescaler, and the number of time quanta (Tq) clock cycles per bit, shown in **Table 8-8** and **8-9.**

$$Bit\ \ Time = \frac{(Prescaler\ \ value)}{f_{CANCLK}} x(number\ \ of\ \ Time\ \ Quanta)$$

8.7.5 CAN Receiver Flag Register (CANRFLG)

A flag can only be cleared when the condition causing the setting is no longer valid and can only be cleared by software; writing 1 to the corresponding bit position. Every flag has an associated interrupt enable bit in the CAN Receiver Interrupt Enable (CANRIER) register.

This register is read/write at any time. A write of 1 clears flag; a write of 0 is ignored.

CAN_BASE+$4	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	0	0	0	WUPIF	RWRNIF	TWRNIF	RERRIF	TERRIF	BOFFIF	OVRIF	RXF
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-15. CAN Receiver Flag Register (CANRFLG)

See Programmer Sheet on Appendix page A-45

Note: The CANRFLG register is held in the reset state when the SFTRES bit in CAN Control 0 (CANCTL0) register is set.

8.7.5.1 Reserved—Bits 15–8

This bit field is reserved or not implemented. It is read as 0 and cannot be modified by writing.

8.7.5.2 Wake-Up Interrupt Flag (WUPIF)—Bit 7

If the CAN detects bus activity while in Sleep mode it sets the WUPIF flag. If not masked, a wake-up interrupt is pending while this flag is set. Please see **Table 8.8.4.**

- 0 = No wake-up activity observed while in Sleep mode
- 1 = CAN detected activity on the bus and requested wake-up

8.7.5.3 Receiver Warning Interrupt Flag (RWRNIF)—Bit 6

This flag is set when the CAN enters warning status due to the Receive Error Counter (REC) exceeding 96 and none of the error interrupt flags, or the Bus Off Interrupt Flag (BOFFIF) is set. If not masked, an error interrupt is pending while this flag is set. When the CPU writes 1 into this flag, the error interrupt signal to CPU is denied, even though the condition to set, the flag continues to remain active. That means a write of 1 by the CPU will inhibit only the interrupt signal to the CPU and does not update the status of the flag.

- 0 = No receiver warning status reached
- 1 = CAN entered receiver warning status

8.7.5.4 Transmitter Warning Interrupt Flag (TWRNIF)—Bit 5

The Transmitter Warning Interrupt Flag (TWRNIF) is set when the CAN enters a warning status due to the Transmit Error Counter (TEC) exceeding 96 when the error interrupt flags or the Bus Off Interrupt Flag (BOFFIF) were not set. If not masked, an error interrupt is pending while this flag is set. When the CPU writes 1 into this flag, the error interrupt signal to CPU is denied, even though the condition to set the flag continues to remain active. That means a write of 1 by the CPU will deny only the interrupt signal to the CPU without updating the status of the flag.

- 0 = No transmitter warning status reached
- 1 = CAN entered transmitter warning status

8.7.5.5 Receiver Error Passive Interrupt Flag (RERRIF)—Bit 4

This flag is set when the CAN enters error passive status due to the receive error counter exceeding 127 and the Bus Off Interrupt Flag (BOFFIF=0) is clear. If not masked, an error interrupt is pending while this flag is set. When the CPU writes 1 into this flag, the error interrupt signal to CPU is denied, even though the condition to set the flag continues to remain active; or a write of 1 by the CPU will deny only the interrupt signal to the CPU without updating the status of the flag.

- 1 = CAN entered receiver error passive status
- 0 = No receiver error passive status reached

8.7.5.6 Transmitter Error Passive Interrupt Flag (TERRIF)—Bit 3

This flag is set when the CAN enters error passive status due to the transmit error counter exceeding 127 and the Bus Off Interrupt Flag (BOFFIF = 0) is clear. If not masked, an error interrupt is pending while this flag is set. When the CPU writes 1 into this flag, the error interrupt signal to CPU is denied, even though the condition to set the flag continues to remain active. For example, a write of 1 by the CPU will deny only the interrupt signal to the CPU without updating the status of the flag.

- 0 = No transmitter error passive status reached
- 1 = CAN entered transmitter error passive status

8.7.5.7 Bus Off Interrupt Flag (BOFFIF)—Bit 2

The Bus Off Interrupt Flag (BOFFIF) is set when the CAN enters bus off status, resulting in the transmit error counter exceeding 255. It cannot be cleared before the CAN has monitored 128 times 11 consecutive recessive bits on the bus. If not masked, an error interrupt is pending while this flag is set. When the CPU writes 1 into this flag, the error interrupt signal to CPU is denied,

even though the condition to set the flag continues to remain active. That means a write of 1 by the CPU will deny only the interrupt signal to the CPU without updating the status of the flag.

- 0 = No bus off status reached
- 1 = CAN entered into bus off status

8.7.5.8 Overrun Interrupt Flag (OVRIF)—Bit 1

This flag is set when a data overrun condition occurs. If not masked, an error interrupt is pending if this flag is set.

- 0 = No data overrun condition
- 1 = A data overrun detected

8.7.5.9 Receive Buffer Full (RXF)—Bit 0

The RXF flag is set by the CAN when a new message is available in the foreground receive buffer. This flag indicates whether the buffer is loaded with a correctly received message, matching Cyclic Redundancy Code (CRC) and no other errors detected. After the CPU reads that message from the Receive Buffer, the RXF flag must be cleared to release the buffer. A set RXF flag prohibits the exchange of the Background Receive
Buffer into the Foreground Buffer. If not masked, a receive interrupt is pending while this flag is set.

- 0 = The receive buffer is released: not full
- 1 = The receive buffer is full and a new message is available

Note: To ensure data integrity, do not read the Receive Buffer registers while the RXF flag is cleared. For MCUs with dual CPUs, reading the Receive Buffer registers while the RXF flag is cleared may result in a CPU fault condition.

8.7.6 CAN Receiver Interrupt Enable Register (CANRIER)

This register contains the interrupt enable bits for the interrupt flags described above. This register can be read/write at any time.

CAN_BASE+$5	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	0	0	0	WUPIE	RWRNIE	TWRNIE	RERRIE	TERRIE	BOFFIE	OVRIE	RXFIE
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-16. CAN Receiver Interrupt Enable Register (CANRIER)

See Programmer Sheet on Appendix page A-46

Note: The CANRIER register is held in the Reset state when the SFTRES bit in the CAN Control 0 (CANCTL0) register is set.

8.7.6.1 Reserved—Bits 15–8

This bit field is reserved or not implemented. It is read as 0 and cannot be modified by writing.

8.7.6.2 Wake-Up Interrupt Enable (WUPIE)—Bit 7

- 0 = No interrupt request is generated from this event
- 1 = A wake-up event causes a wake-up interrupt request

8.7.6.3 Receiver Warning Interrupt Enable (RWRNIE)—Bit 6

- 0 = No interrupt request is generated from this event
- 1 = A receiver warning status event causes an error interrupt request

8.7.6.4 Transmitter Warning Interrupt Enable (TWRNIE)—Bit 5

- 0 = No interrupt request is generated from this event
- 1 = A transmitter warning status event causes an error interrupt request

8.7.6.5 Receiver Error Passive Interrupt Enable (RERRIE)—Bit 4

- 0 = No interrupt request is generated from this event
- 1 = A receiver error passive status event causes an error interrupt request

8.7.6.6 Transmitter Error Passive Interrupt Enable (TERRIE)—Bit 3

- 0 = No interrupt request is generated from this event
- 1 = A transmitter error passive status event causes an error interrupt request

8.7.6.7 Bus Off Interrupt Enable (BOFFIE)—Bit 2

- 0 = No interrupt request is generated from this event
- 1 = A bus off event causes an error interrupt request

8.7.6.8 Overrun Interrupt Enable (OVRIE)—Bit 1

- 0 = No interrupt request is generated from this event
- 1 = An overrun event causes an error interrupt request

8.7.6.9 Receiver Full Interrupt Enable (RXFIE)—Bit 0

- 0 = No interrupt request is generated from this event
- 1 = A receive buffer full (successful message reception) event causes a receiver interrupt request

8.7.7 CAN Transmitter Flag Register (CANTFLG)

The transmit buffer empty flags each have an associated interrupt enable bit in the CAN Transmitter Control (CANTCR) register. This register can be read at any time. Write is reserved for ABTAK[2:0] flags; anytime for TXE[2:0] flags, write of 1 clears flag, a write of 0 is ignored.

CAN_BASE+$6	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	0	0	0	0	ABTAK2	ABTAK1	ABTAK0	0	TXE2	TXE1	TXE0
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Figure 8-17. CAN Transmitter Flag Register (CANTFLG)

See Programmer Sheet on Appendix page A-47

Note: The CANTFLG register is held in the Reset state when the SFTRES bit in CANCTL0 is set.

8.7.7.1 Reserved—Bits 15–7

This bit field is reserved or not implemented. It is read as 0 and cannot be modified by writing.

8.7.7.2 Abort Acknowledge (ABTAK)—Bits 6–4

This flag acknowledges a message was stopped due to a pending stop request from the CPU. After a particular message buffer is flagged empty, this flag can be used by the application software to identify whether the message was quit successfully or was sent anyway. The ABTAK[2:0] Flag is cleared whenever the corresponding TXE flag is cleared.

- 0 = The message was not stopped; it was sent out
- 1 = The message was stopped

8.7.7.3 Reserved—Bits 3

This bit is reserved or not implemented. It is read as 0 and cannot be modified by writing.

8.7.7.4 Transmitter Buffer Empty (TXE)—Bits 2–0

This flag indicates the associated transmit message buffer is empty, and thus not scheduled for transmission. The CPU must clear the flag after a message is set up in the Transmit Buffer and is due for transmission. The CAN sets the flag after the message is sent successfully. The flag is also set by the CAN when the transmission request is successfully ended due to a pending stop request. Please see **Section 8.7.8, "CAN Transmitter Control Register (CANTCR)".** If not masked, a transmit interrupt is pending while this flag is set.

Clearing a TXE[2:0] flag also clears the corresponding ABTAK[2:0]. When a TXE[2:0] flag is set, the corresponding ABTRQ[2:0] bit is cleared. Please refer to **Section 8.7.8, "CAN Transmitter Control Register (CANTCR)".**

- 0 = The associated message buffer is full, loaded with a message due for transmission
- 1 = The associated message buffer is empty, or not scheduled

Note: To ensure data integrity, do not write to the Transmit Buffer registers while the associated TXE flag is cleared.

8.7.8 CAN Transmitter Control Register (CANTCR)

The CANTCR register provides for various transmitter control. This register can be read/write at any time.

CAN_BASE+$7	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	0	0	0	0	ABTRQ2	ABTRQ1	ABTRQ0	0	TXEIE2	TXEIE1	TXEIE0
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-18. CANCAN Transmitter Control Register (CANTCR)

See Programmer Sheet on Appendix page A-47

Note: The CANTCR register is held in the Reset state when the SFTRES bit in CANCTL0 is set.

8.7.8.1 Reserved—Bits 15–7

This bit field is reserved or not implemented. It is read as 0 and cannot be modified by writing.

8.7.8.2 Abort Request (ABTRQ)—Bits 6–4

The CPU sets the ABTRQ[2:0] bit to request a scheduled message buffer (TXE[2:0] = 0) be aborted. The CAN consents to the request if the message has not already started transmission and if the transmission is not successful, meaning it lost arbitration or is an error. When a message is stopped, the associated TXE and Abort Acknowledge (ABTAK) flags are set and a transmit interrupt occurs if enabled. See **Section 8.7.7, "CAN Transmitter Flag Register (CANTFLG)".** The CPU *cannot* reset ABTRQ[2:0]. ABTRQ[2:0] is reset whenever the associated TXE flag is set.

- 0 = No abort request
- 1 = Abort request pending

Note: The software must not clear one or more of the TXE[2:0] Flags in CANTFLG and simultaneously set the respective ABTRQ bit(s).

8.7.8.3 Reserved—Bit 3

This bit is reserved or not implemented. It is read as 0 and cannot be modified by writing.

8.7.8.4 Transmitter Empty Interrupt Enable (TXEIE)—Bits 2–0

- 0 = No interrupt request is generated from this event
- 1 = Empty transmitter, transmit buffer available for transmission, event causes a transmitter empty interrupt request

8.7.9 CAN Identifier Acceptance Control Register (CANIDAC)

The CANIDAC register provides for identifier acceptance control. This register can be read/write at any time when SFTRES equals 1. The only exception is when bits IDHIT[2:0] are unfulfilled.

CAN_BASE+$8	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	0	0	0	0	0	IDAM1	IDAM0	0	IDHIT2	IDHIT1	IDHIT0
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-19. CAN Identifier Acceptance Control Register (CANIDAC)

See Programmer Sheet on Appendix page A-48

8.7.9.1 Reserved—Bits 15–6

This bit field is reserved or not implemented. It is read as 0 and cannot be modified by writing.

8.7.9.2 Identifier Acceptance Mode (IDAM)—Bits 5–4

The CPU sets these flags to define the Identifier Acceptance filter organization. Please refer to **Section 8.4.1.4, "Identifier Acceptance Filter".** **Table 8-10** summarizes the different settings. In Filter Closed mode, no message is accepted, so the foreground buffer is never reloaded.

Table 8-10. Identifier Acceptance Mode Settings

IDAM1	IDAM0	Identifier Acceptance Mode
0	0	Two 32-Bit Acceptance Filters
0	1	Four 16-Bit Acceptance Filters
1	0	Eight 8-Bit Acceptance Filters
1	1	Filter Closed

8.7.9.3 Reserved—Bit 3

This bit is reserved or not implemented. It is read as 0 and cannot be modified by writing.

8.7.9.4 Identifier Acceptance Hit Indicator (IDHIT)—Bits 2–0

The CAN sets these flags indicating an identifier acceptance hit **Table 8-11** summarizes the different settings. Please refer to **Section 8.4.1.4, "Identifier Acceptance Filter".**

Table 8-11. Identifier Acceptance Hit Indication

IDHIT2	IDHIT1	IDHIT0	Identifier Acceptance Hit
0	0	0	Filter 0 Hit
0	0	1	Filter 1 Hit
0	1	0	Filter 2 Hit
0	1	1	Filter 3 Hit
1	0	0	Filter 4 Hit
1	0	1	Filter 5 Hit
1	1	0	Filter 6 Hit
1	1	1	Filter 7 Hit

The IDHIT indicators are always related to the message in the Foreground Buffer. When a message gets copied from the Background to the Foreground Buffer, the indicators are updated as well. This register is always read as $00 in Normal modes; write is reserved in Normal modes.

Note: Writing to these registers when in special modes can alter the CAN functionality.

8.7.10 CAN Receive Error Counter Register (CANRXERR)

This register reflects the status of the CAN receive error counter. This register is *read- only* when in Sleep or Soft Reset mode. Write is reserved.

CAN_BASE+$E	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	0	0	0	RXERR7	RXERR6	RXERR5	RXERR4	RXERR3	RXERR2	RXERR1	RXERR0
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-20. CAN Receive Error Counter Register (CANRXERR)

See Programmer Sheet on Appendix page A-49

Note: Reading this register when in any other mode other than Sleep or Soft Reset, may return an incorrect value. For MCUs with dual CPUs may result in a CPU fault condition. Writing to this register when in special modes can alter the CAN functionality.

8.7.11 CAN Transmit Error Counter Register (CANTXERR)

This register reflects the status of the CAN transmit error counter. It is *read only* when in Sleep or Soft Reset mode. Write is reserved.

CAN_BASE+$F	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	0	0	0	TXERR7	TXERR6	TXERR5	TXERR4	TXERR3	TXERR2	TXERR1	TXERR0
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-21. CAN Transmit Error Counter Register (CANTXERR)

See Programmer Sheet on Appendix page A-49

Note: Reading this register when in any other mode other than Sleep or Soft Reset, may return an incorrect value. For MCUs with dual CPUs, this may result in a CPU fault condition. Writing to this register when in special modes can alter the CAN functionality.

8.7.12 CAN Identifier Acceptance Registers (CANIDAR0–7)

On reception, each message is written into the Background Receive Buffer (RXBG). The RXBG register *cannot* be read. The CPU is only signalled to read the message if it passes the criteria in the identifier acceptance and the identifier mask registers are accepted. Otherwise, the message is overwritten by the next message, or dropped.

The CANIDAR0–7 acceptance registers of the CAN are applied on the IDR0 to IDR3 registers of incoming messages in a bit-by-bit manner. Refer to **Section 8.7.15, "Receive/Transmit Buffer Identifier Registers (IDR0–3)".**

For extended identifiers, all four acceptance and mask registers are applied. For standard identifiers, only the first two registers, CANIDAR0/1 and CANIDMR0/1, are applied. These registers can be read at any time and can be modified only when SFTRES equals 1. For more information on two, four, and eight ID acceptance register functionality, see **Section 8.4.1.4, "Identifier Acceptance Filter".**

CAN Identifier Acceptance Register 0 (CANIDAR0) — Address: CAN_BASE + $10
CAN Identifier Acceptance Register 1 (CANIDAR1) — Address: CAN_BASE + $11
CAN Identifier Acceptance Register 2 (CANIDAR2) — Address: CAN_BASE + $12
CAN Identifier Acceptance Register 3 (CANIDAR3) — Address: CAN_BASE + $13

CAN_BASE+$F	15 - 8	7	6	5	4	3	2	1	0
Read	0	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
Write									
Reset	0	0	0	0	0	0	0	0	0

Figure 8-22. CAN Identifier Acceptance Registers - 1st Bank (CANIDAR0–3)

See Programmer Sheet on Appendix page A-50

CAN Identifier Acceptance Register 4 (CANIDAR4) — Address: CAN_BASE + $18
CAN Identifier Acceptance Register 5 (CANIDAR5) — Address: CAN_BASE + $19
CAN Identifier Acceptance Register 6 (CANIDAR6) — Address: CAN_BASE + $1A
CAN Identifier Acceptance Register 7 (CANIDAR7) — Address: CAN_BASE + $1B

CAN_BASE+$F	15 - 8	7	6	5	4	3	2	1	0
Read	0	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
Write									
Reset	0	0	0	0	0	0	0	0	0

Figure 8-23. CAN Identifier Acceptance Registers - 2nd Bank (CANIDAR4–7)

See Programmer Sheet on Appendix page A-50

8.7.12.1 Acceptance Code Bits (AC)—Bits 7–0

The Acceptance Code (AC[7:0]) bits comprise sequence of bits defined by the developer. The corresponding bits of the Related Identifier (DR0–IDR3) register of the Receive Message Buffer are compared. The result of this comparison is masked with the corresponding Identifier Mask register.

8.7.13 CAN Identifier Mask Registers (CANIDMR0–7)

The Identifier Mask register specifies which of the corresponding bits in the Identifier Acceptance register are relevant for acceptance filtering. To receive standard identifiers in 32-bit Filter mode, it is required to program the last three bits (AM[2:0]) in the mask registers,

CANIDMR1 and CANIDMR5 as 111. This is mandatory for the standard identifier. To receive standard identifiers in 16-bit Filter mode, it is necessary to program the last three bits (AM[2:0]) in the Mask registers (CANIDMR1, CANIDMR3, CANIDMR5, and CANIDMR7) to 111. This register can be read/write anytime when SFTRES equals 1.

CAN Identifier Mask Register 0 (CANIDMR0) — Address: CAN_BASE + $14
CAN Identifier Mask Register 1 (CANIDMR1) — Address: CAN_BASE + $15
CAN Identifier Mask Register 2 (CANIDMR2) — Address: CAN_BASE + $16
CAN Identifier Mask Register 3 (CANIDMR3) — Address: CAN_BASE + $17

CAN_BASE+$F	15 - 8	7	6	5	4	3	2	1	0
Read	0	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
Write									
Reset	0	0	0	0	0	0	0	0	0

Figure 8-24. CAN Identifier Mask Registers - 1st Bank (CANIDMR0–3)

See Programmer Sheet on Appendix page A-50

CAN Identifier Mask Register 5 (CANIDMR5) — Address: CAN_BASE + $1D
CAN Identifier Mask Register 6 (CANIDMR6) — Address: CAN_BASE + $1E
CAN Identifier Mask Register 7 (CANIDMR7) — Address: CAN_BASE + $1F
CAN Identifier Mask Register 4 (CANIDMR4) — Address: CAN_BASE + $1C

CAN_BASE+$F	15 - 8	7	6	5	4	3	2	1	0
Read	0	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
Write									
Reset	0	0	0	0	0	0	0	0	0

Figure 8-25. CAN Identifier Mask Registers - 2nd Bank (CANIDMR4–7)

See Programmer Sheet on Appendix page A-50

8.7.13.1 Acceptance Mask Bits (AM)—Bits 7–0

If any particular bit in this register is cleared, the corresponding bit in the Identifier Acceptance (AC) register must match the cleared Acceptance Mask (AM) bit. The message is accepted if all bits match. If a bit is set, it indicates the state of the corresponding bit in the identifier acceptance register, and does not affect the message being accepted:

- Ignore corresponding acceptance code register bit
- Match corresponding acceptance code register and identifier bits

8.7.14 Programmer's Model of Message Storage

The following section details the organization of the Receive and Transmit Message Buffers (RXB/TXB) and the Associated Control (AC) registers. For reasons of programmer interface simplification, the Receive and Transmit Message Buffers have the same outline. Each message buffer allocates 16 words in the memory map, containing a 13-word data structure. An additional Transmit Buffer Priority (TBPR) register is defined for the transmit buffers.

Table 8-12. Message Buffer Organization for Peripheral Address Locations

Register Name	Receive Buffer (RB)	Transmit Buffer 0 (TB0)	Transmit Buffer 1 (TB1)	Transmit Buffer 2 (TB2)
Identifier Register 0 (IDR0)	CAN_BASE+$40	CAN_BASE+$50	CAN_BASE+$60	CAN_BASE+$70
Identifier Register 1 (IDR1)	CAN_BASE+$41	CAN_BASE+$51	CAN_BASE+$61	CAN_BASE+$71
Identifier Register 2 (IDR2)	CAN_BASE+$42	CAN_BASE+$52	CAN_BASE+$62	CAN_BASE+$72
Identifier Register 3 (IDR3)	CAN_BASE+$43	CAN_BASE+$53	CAN_BASE+$63	CAN_BASE+$73
Data Segment Register 0 (DSR0)	CAN_BASE+$44	CAN_BASE+$54	CAN_BASE+$64	CAN_BASE+$74
Data Segment Register 1 (DSR1)	CAN_BASE+$45	CAN_BASE+$55	CAN_BASE+$65	CAN_BASE+$75
Data Segment Register 2 (DSR2)	CAN_BASE+$46	CAN_BASE+$56	CAN_BASE+$66	CAN_BASE+$76
Data Segment Register 3 (DSR3)	CAN_BASE+$47	CAN_BASE+$57	CAN_BASE+$67	CAN_BASE+$77
Data Segment Register 4 (DSR4)	CAN_BASE+$48	CAN_BASE+$58	CAN_BASE+$68	CAN_BASE+$78
Data Segment Register 5 (DSR5)	CAN_BASE+$49	CAN_BASE+$59	CAN_BASE+$69	CAN_BASE+$79
Data Segment Register 6 (DSR6)	CAN_BASE+$4A	CAN_BASE+$5A	CAN_BASE+$6A	CAN_BASE+$7A
Data Segment Register 7 (DSR7)	CAN_BASE+$4B	CAN_BASE+$5B	CAN_BASE+$6B	CAN_BASE+$7B
Data Length Register (DLR)	CAN_BASE+$4C	CAN_BASE+$5C	CAN_BASE+$6C	CAN_BASE+$7C
Transmit Buffer Priority Register[1] (TBPR)	CAN_BASE+$4D	CAN_BASE+$5D	CAN_BASE+$6D	CAN_BASE+$7D
Reserved	CAN_BASE+$4E	CAN_BASE+$5E	CAN_BASE+$6E	CAN_BASE+$7E
Reserved	CAN_BASE+$4F	CAN_BASE+$5F	CAN_BASE+$6F	CAN_BASE+$7F

1. Not applicable for receive buffers.

Note: CAN_BASE defined in Table 3-8.

Section 8-27 provides the common 13-word data structure of Receive and Transmit Buffers for extended identifiers. The mapping of standard identifiers into the IDRs is shown in **Section 8-26.** *All bits of the Receive and Transmit Buffers are 0 out of reset.*

Transmit buffers can be read at any time. Receive buffers content is only readable when the RXF flag in the CANRFLG register is set for receive. Please refer to **Section 8.7.5, "CAN Receiver Flag Register (CANRFLG)".**

Transmit buffers can be modified any time when the TXE[2:0] flag is set; it is reserved to receive buffers. Please refer to **Section 8.7.7, "CAN Transmitter Flag Register (CANTFLG)".**

8.7.15 Receive/Transmit Buffer Identifier Registers (IDR0–3)

The Identifier registers for an extended format identifier consist of a total of 32 bits: ID[28:0], SRR, IDE, and RTR bits.

The Identifier registers for a standard format identifier consist of a total of 13 bits: ID[10:0], RTR, and IDE bits.

CAN Receive Buffer Identifier Register 0 (CAN_RB_IDR0) — Address: CAN_BASE + $40
CAN Receive Buffer Identifier Register 1 (CAN_RB_IDR1) — Address: CAN_BASE + $41
CAN Receive Buffer Identifier Register 2 (CAN_RB_IDR2) — Address: CAN_BASE + $42
CAN Receive Buffer Identifier Register 3 (CAN_RB_IDR3) — Address: CAN_BASE + $43

CAN Transmit Buffer 0 Identifier Register 0 (CAN_TB0_IDR0) — Address: CAN_BASE + $50
CAN Transmit Buffer 0 Identifier Register 1 (CAN_TB0_IDR1) — Address: CAN_BASE + $51
CAN Transmit Buffer 0 Identifier Register 2 (CAN_TB0_IDR2) — Address: CAN_BASE + $52
CAN Transmit Buffer 0 Identifier Register 3 (CAN_TB0_IDR3) — Address: CAN_BASE + $53

CAN Transmit Buffer 1 Identifier Register 0 (CAN_TB1_IDR0) — Address: CAN_BASE + $60
CAN Transmit Buffer 1 Identifier Register 1 (CAN_TB1_IDR1) — Address: CAN_BASE + $61
CAN Transmit Buffer 1 Identifier Register 2 (CAN_TB1_IDR2) — Address: CAN_BASE + $62
CAN Transmit Buffer 1 Identifier Register 3 (CAN_TB1_IDR3) — Address: CAN_BASE + $63

CAN Transmit Buffer 2 Identifier Register 0 (CAN_TB2_IDR0) — Address: CAN_BASE + $70
CAN Transmit Buffer 2 Identifier Register 1 (CAN_TB2_IDR1) — Address: CAN_BASE + $71
CAN Transmit Buffer 2 Identifier Register 2 (CAN_TB2_IDR2) — Address: CAN_BASE + $72
CAN Transmit Buffer 2 Identifier Register 3 (CAN_TB2_IDR3) — Address: CAN_BASE + $73

Reset: $0000

Register Name	Bits	15–8	7	6	5	4	3	2	1	0
IDR0	Read	0	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3
	Write									
IDR1	Read	0	ID2	ID1	ID0	RTR	IDE (=0)			
	Write									
IDR2	Read									
	Write									
IDR3	Read									
	Write									

Figure 8-26. Standard Identifier Mapping Registers (IDR0–3)

See Programmer Sheet on Appendix page A-54

Register Name	Bits	15-8	7	6	5	4	3	2	1	0
IDR0	Read	0	ID28	ID27	ID26	ID25	ID24	ID23	ID22	ID21
	Write									
IDR1	Read	0	ID20	ID19	ID18	SRR (=1)	IDE (=1)	ID17	ID16	ID15
	Write									
IDR2	Read	0	ID14	ID13	ID12	ID11	ID10	ID9	ID8	ID7
	Write									
IDR3	Read	0	ID6	ID5	ID4	ID3	ID2	ID1	ID0	RTR
	Write									

0 Read as 0

Reserved

Figure 8-27. Extended Identifier Mapping Registers (IDR0–3)

See Programmer Sheet on Appendix page A-54

8.7.15.1 Extended Format Identifier—Bits ID[28:0]

The identifiers consist of 29 bits (ID28–ID0) for the extended format. ID28 is the MSB, transmitted first on the bus during the arbitration procedure. The priority of an identifier is defined to be highest for the smallest binary number.

8.7.15.2 Standard Format Identifier—Bits ID[10:0]

The identifiers consist of 11 bits (ID[10:0]) for the standard format. ID[10] is the MSB, transmitted first on the bus during the arbitration procedure. The priority of an identifier is defined to be highest for the smallest binary number.

8.7.15.3 Substitute Remote Request (SRR)—Bit 4 in IDR1 (Extended)

This fixed recessive bit is used only in extended format. The bit must be set to 1 for transmission buffers. It is stored as received on the CAN bus for receive buffers.

8.7.15.4 ID Extended (IDE)—Bit 3 in IDR1

This flag indicates whether the extended or standard identifier format is applied in this buffer. In the case of a receive buffer, the flag is set as received indicating to the CPU how to process the Buffer Identifier registers. In the case of a transmit buffer, the flag indicates to the CAN what type of identifier to send.

- 0 = Standard format (11-bit)
- 1 = Extended format (29-bit)

8.7.15.5 Remote Transmission Request (RTR)—Bit 4 in IDR1 (Standard)

This flag reflects the status of the Remote Transmission Request (RTR) bit in the CAN frame. In the case of a receive buffer, it indicates the status of the received frame and supports the transmission of an answering frame in software. In the case of a transmit buffer, this flag defines the setting of the RTR bit to be sent.

- 0 = Data frame
- 1 = Remote frame

8.7.16 Receive/Transmit Buffer Data Segment Registers (DSR0–7)

The eight Data Segment Registers (DSR0-7), each with bits DB[7:0], contain the data to be transmitted or received. The number of bytes to be transmitted or received is determined by the data length code in the corresponding DLR register.

CAN Receive Buffer Data Segment Register 0 (CAN_RB_DSR0) — Address: CAN_BASE + $44
CAN Receive Buffer Data Segment Register 1 (CAN_RB_DSR1) — Address: CAN_BASE + $45
CAN Receive Buffer Data Segment Register 2 (CAN_RB_DSR2) — Address: CAN_BASE + $46
CAN Receive Buffer Data Segment Register 3 (CAN_RB_DSR3) — Address: CAN_BASE + $47
CAN Receive Buffer Data Segment Register 4 (CAN_RB_DSR4) — Address: CAN_BASE + $48
CAN Receive Buffer Data Segment Register 5 (CAN_RB_DSR5) — Address: CAN_BASE + $49
CAN Receive Buffer Data Segment Register 6 (CAN_RB_DSR6) — Address: CAN_BASE + $4A
CAN Receive Buffer Data Segment Register 7 (CAN_RB_DSR7) — Address: CAN_BASE + $4B

CAN Transmit Buffer 0 Data Segment Register 0 (CAN_TB0_DSR0) — Address: CAN_BASE + $54
CAN Transmit Buffer 0 Data Segment Register 1 (CAN_TB0_DSR1) — Address: CAN_BASE + $55
CAN Transmit Buffer 0 Data Segment Register 2 (CAN_TB0_DSR2) — Address: CAN_BASE + $56
CAN Transmit Buffer 0 Data Segment Register 3 (CAN_TB0_DSR3) — Address: CAN_BASE + $57
CAN Transmit Buffer 0 Data Segment Register 4 (CAN_TB0_DSR4) — Address: CAN_BASE + $58
CAN Transmit Buffer 0 Data Segment Register 5 (CAN_TB0_DSR5) — Address: CAN_BASE + $59
CAN Transmit Buffer 0 Data Segment Register 6 (CAN_TB0_DSR6) — Address: CAN_BASE + $5A
CAN Transmit Buffer 0 Data Segment Register 7 (CAN_TB0_DSR7) — Address: CAN_BASE + $5B

CAN Transmit Buffer 1 Data Segment Register 0 (CAN_TB1_DSR0) — Address: CAN_BASE + $64
CAN Transmit Buffer 1 Data Segment Register 1 (CAN_TB1_DSR1) — Address: CAN_BASE + $65
CAN Transmit Buffer 1 Data Segment Register 2 (CAN_TB1_DSR2) — Address: CAN_BASE + $66
CAN Transmit Buffer 1 Data Segment Register 3 (CAN_TB1_DSR3) — Address: CAN_BASE + $67
CAN Transmit Buffer 1 Data Segment Register 4 (CAN_TB1_DSR4) — Address: CAN_BASE + $68
CAN Transmit Buffer 1 Data Segment Register 5 (CAN_TB1_DSR5) — Address: CAN_BASE + $69
CAN Transmit Buffer 1 Data Segment Register 6 (CAN_TB1_DSR6) — Address: CAN_BASE + $6A
CAN Transmit Buffer 1 Data Segment Register 7 (CAN_TB1_DSR7) — Address: CAN_BASE + $6B

CAN Transmit Buffer 2 Data Segment Register 0 (CAN_TB2_DSR0) — Address: CAN_BASE + $74
CAN Transmit Buffer 2 Data Segment Register 1 (CAN_TB2_DSR1) — Address: CAN_BASE + $75
CAN Transmit Buffer 2 Data Segment Register 2 (CAN_TB2_DSR2) — Address: CAN_BASE + $76
CAN Transmit Buffer 2 Data Segment Register 3 (CAN_TB2_DSR3) — Address: CAN_BASE + $77
CAN Transmit Buffer 2 Data Segment Register 4 (CAN_TB2_DSR4) — Address: CAN_BASE + $78
CAN Transmit Buffer 2 Data Segment Register 5 (CAN_TB2_DSR5) — Address: CAN_BASE + $79
CAN Transmit Buffer 2 Data Segment Register 6 (CAN_TB2_DSR6) — Address: CAN_BASE + $7A
CAN Transmit Buffer 2 Data Segment Register 7 (CAN_TB2_DSR7) — Address: CAN_BASE + $7B

Register Name	Bits	15-8	7	6	5	4	3	2	1	0
DSR0	Read	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	Write									
DSR1	Read	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	Write									
DSR2	Read	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	Write									
DSR3	Read	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	Write									
DSR4	Read	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	Write									
DSR5	Read	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	Write									
DSR6	Read	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	Write									
DSR7	Read	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	Write									

0 Read as 0

Reserved

Figure 8-28. Receive Buffer Data Segment Registers

See Programmer Sheet on Appendix page A-62

8.7.17 Data Length Register (DLR)

This register keeps the data length field of the CAN frame.

Data Length Register (DLR) code bits DLC[3:0] contain the number of bytes (data byte count) of the respective message. During the transmission of a remote frame, the data length code is transmitted as programmed while the number of transmitted data bytes is always 0 in a remote frame. The data byte count ranges from zero to eight for a data frame. **Table 8-13** demonstrates the effect of setting the DLC bits.

CAN Receive Buffer Data Length Register (CAN_RB_DLR) — Address: CAN_BASE + $4C
CAN Transmit Buffer 0 Data Length Register (CAN_TB0_DLR) — Address: CAN_BASE + $5C
CAN Transmit Buffer 1 Data Length Register (CAN_TB1_DLR) — Address: CAN_BASE + $6C

CAN Transmit Buffer 2 Data Length Register (CAN_TB2_DLR) — Address: CAN_BASE + $7C

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	0	0	0	0	0	0	0	DLC3	DLC2	DLC1	DLC0
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-29. Data Length Register (DLR)

See Programmer Sheet on Appendix page A-62

Table 8-13. Data Length Codes

Data Length Code				Data Byte Count
DLC3	DLC2	DLC1	DLC0	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8

8.7.18 Transmit Buffer Priority Register (TBPR)

A Transmit Buffer Priority Register (TBPR) defines the local priority of the associated message buffer. The local priority is used for the internal prioritization process of the CAN and is defined to be highest for the smallest binary number. The CAN implements the following internal prioritization mechanisms:

- All transmission buffers with a cleared TXE[2:0] flag participate in the prioritization immediately before the start of frame (SOF) is sent

- The transmission buffer with the lowest local priority field wins the prioritization
- In cases of more than one buffer having the same lowest priority, the message buffer with the lower index number wins

This register can be read/write at any time.

CAN Transmit Buffer 0 Priority Register (CAN_TB0_TBPR)— Address: CAN_BASE + $5D
CAN Transmit Buffer 1 Priority Register (CAN_TB1_TBPR) — Address: CAN_BASE + $6D
CAN Transmit Buffer 2 Priority Register (CAN_TB2_TBPR) — Address: CAN_BASE + $7D

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	0	0	0	PRIO7	PRIO6	PRIO5	PRIO4	PRIO3	PRIO2	PRIO1	PRIO0
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-30. Transmit Buffer Priority Register (TBPR)

See Programmer Sheet on Appendix page A-63

8.7.18.0.1 Reserved—Bits 15–8

This bit field is reserved or not implemented. It is read as 0 and cannot be modified by writing.

8.7.18.0.2 Local Priority (PRIO)—Bits 7–0

These bits define the Local Priority of the associated message buffer. The Local Priority is used for the internal prioritization of the transmit process of the CAN and is defined to be highest for the smallest binary value.

- All transmission buffers with cleared Transmitter Buffer Empty (TXE) flags in the CAN Transmitter Flag (CTFLG) register participate in the prioritization immediately before the Start of Frame (SOF) is sent.
- The transmission buffer with the lowest local priority field wins prioritization.
- If more than one buffer has the same lowest priority value, the message buffer with the lower index wins. Proper assignment of priority levels should be done at design (referring to the message ID strategy) to prevent this from possibly changing the intended priority of messages transmitted.

8.8 Low Power Options

When the CAN is disabled, that is to say CANE equals 0, the CAN clocks are stopped for power savings.

When the CAN is enabled, or CANE equals 1, the CAN has three modes with reduced power consumption in addition to Normal mode:

1. Sleep
2. Soft Reset

3. Power Down

In Sleep and Soft Reset modes, power consumption is reduced by stopping all clocks except those to access the registers. In the Power Down mode, all clocks are stopped and no power is consumed.

Table 8-14 summarizes the combinations of CAN and CPU modes. A particular combination of modes is entered by the given settings on the CSWAI, SLPAK, and SFTRES bits.

For all modes, an CAN wake-up interrupt can only occur if the CAN is in Sleep mode, that is to say SLPAK equals 1, and the wake-up interrupt is enabled, or WUPIE equals 1.

Table 8-14. CAN vs. CPU Operating Modes

CAN Mode	CPU Mode[1]		
	RUN	WAIT	STOP
Sleep	CSWAI = X SLPAK = 1 SFTRES = 0	CSWAI = 0 SLPAK = 1 SFTRES = 0	
Soft Reset	CSWAI = X SLPAK = 0 SFTRES = 1	CSWAI = 0 SLPAK = 0 SFTRES = 1	
Power Down		CSWAI = 1 SLPAK = X SFTRES = X	CSWAI = X SLPAK = X SFTRES = X
Normal	CSWAI = X SLPAK = 0 SFTRES = 0	CSWAI = 0 SLPAK = 0 SFTRES = 0	

1. 'X' means don't care.

Note: If CAN is in Sleep mode and a Soft Reset is asserted, CAN Wake-Up Interrupt is *not* asserted.

8.8.1 Run Mode

As illustrated in **Table 8-14,** two low power options are available in Run mode: Sleep SLPAK equals 1, and Soft Reset SFTRES equals 1 modes.

8.8.2 Wait Mode

The wait instruction places the MCU in a low power consumption Stand-By mode. If the CSWAI bit is set, additional power can be saved because the CPU clocks have stopped. While the CPU is in Wait mode, the CAN can be operated in Normal mode and still generate interrupts. Registers can be accessed via background Debug mode. The CAN can also operate in any of the low power

modes depending on the values of the SLPAK, SFTRES, and CSWAI bits, detailed in **Table 8-14.**

Please refer to **Section 8.9.3, "Recovery from Stop or Wait"** for recovery from Stop or Wait modes.

Note: **Important:** It is possible for CAN to wake-up the CPU from Wait mode if the following sequence is followed:

1. CAN is put into Sleep mode, discussed in **Section 8.8.4, "Sleep Mode"**
2. CSWAI Control bit is set in CANCTL0 register to set CAN stops in Wait mode
3. CAN is put into Wait mode

In this case, CAN would generate a Wake-Up Interrupt signal to the CPU on sensing any bus event of CAN bus. Glitches of less than 5µs are filtered out. This wake-up design uses the crystal oscillator clock directly and also the whole path for assertion of wake-up interrupt is asynchronous with respect to the system operation.

8.8.3 Stop Mode

The Stop instruction places the MCU in a low power consumption Stand-By mode. In Stop mode, the CAN operates in Power Down mode regardless of the value of the SLPAK, SFTRES, and CSWAI bits. See **Table 8-14.**

Please refer to **Section 8.9.3, "Recovery from Stop or Wait"** for recovery from Stop mode.

Note: **Important:** It is possible for CAN to wake-up the CPU from Stop mode only if the following sequence is followed:

1. CAN is put into Sleep mode, discussed in **Section 8.8.4, "Sleep Mode"**
2. CAN is put into Stop mode

In this case, CAN would generate a Wake-Up Interrupt signal to the CPU on sensing any bus event of CAN bus. Glitches of less than 5µs are filtered out. Our wake-up design uses the crystal oscillator clock directly and also the whole path for assertion of wake-up interrupt is asynchronous with respect to the system operation.

8.8.4 Sleep Mode

The CPU can request the CAN to enter this Low Power mode by asserting the SLPRQ bit in the CANCTL0 register. Please see **Section 8-31.** The time when the CAN enters Sleep mode depends on its activity:

- If it is transmitting, it continues to transmit until the entire message is transmitted and then goes into Sleep mode
- If it is receiving it waits for the end of this message and goes into Sleep mode
- If it is neither transmitting nor receiving, it immediately goes into Sleep mode

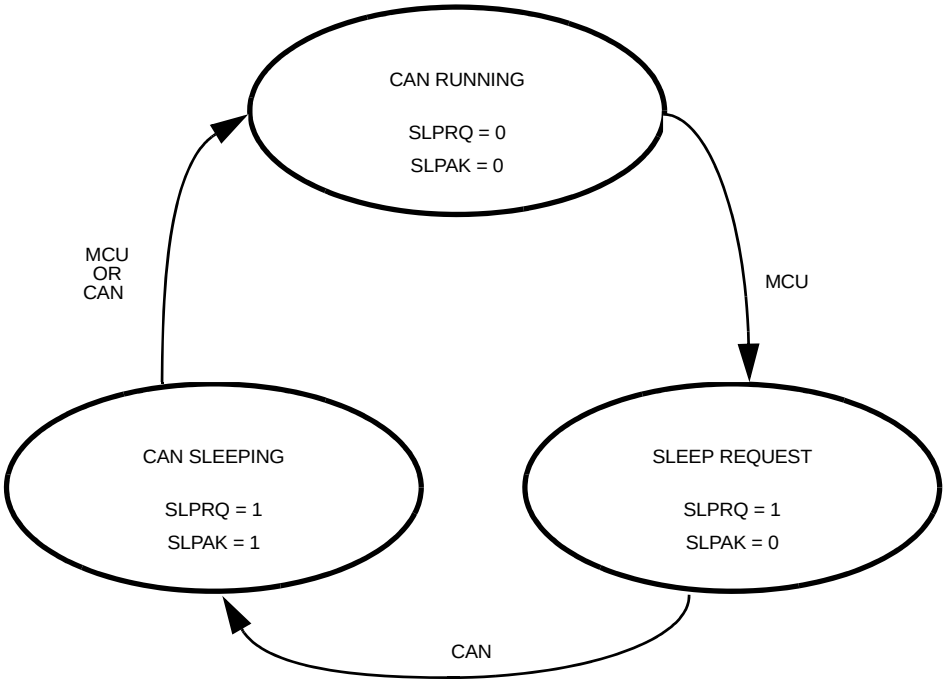


Figure 8-31. Sleep Request/Acknowledge Cycle

Note: The application software must avoid setting up a transmission by clearing 1 or more TXE[2:0] flag(s) then immediately request Sleep mode by setting SLPRQ. It depends on the exact sequence of operations whether the CAN starts transmitting or goes directly into Sleep mode.

The SLPAK flag is set during Sleep mode. It is recommended the application software use SLPAK as a handshake indication for the request to go into Sleep mode (SLPRQ).

When in Sleep mode, the CAN stops its internal clocks, eliminating transmit and receive operation performances. However, clocks allow register accesses to still run. If the CAN is in a Bus Off state, it stops counting the 128 x 11 consecutive recessive bits because of the stopped

clocks. The MSCAN_TX pin remains in a recessive state. If RXF equals 1, the message can be read and RXF can be cleared. Copying RxBG into RxFG does not take place while in Sleep mode. It is possible to access the transmit buffers and to clear the associated TXE[2:0] flags. No message abort will take place while in Sleep mode.

The CAN leaves Sleep mode, when:

- Bus activity occurs, or
- MCU clears the SLPRQ bit, or
- MCU sets the SFTRES bit

Note: The MCU *cannot* clear the SLPRQ bit before the CAN has entered Sleep mode, or SLPAK equals 1.

Note: **Important:** It is possible for CAN to wake-up the CPU from Wait mode, if the following sequence is followed:

1. CAN placed in Sleep mode, discussed in **Section 8.8.4, "Sleep Mode"**
2. CSWAI Control bit is set in CANCTL0 register to set CAN stops in Wait mode
3. CAN is placed into Wait mode

In this case, CAN would generate a Wake-Up Interrupt signal to the CPU on sensing any bus event of CAN bus. Glitches of less than 5µ are filtered out. This wake-up design uses the crystal oscillator clock directly and also the whole path for assertion of wake-up interrupt is asynchronous.

Note: **Important:** After a wake-up from any low power mode, the CAN waits 11 consecutive recessive bits to synchronize to the bus. As a consequence, if the CAN is awakened by a CAN frame, it is not received. The Receive Message Buffers (RxFG and RxBG) contain messages if they were received before Sleep mode was entered. All pending actions will be executed upon wake-up; copying of RxBG into RxFG, message aborts and message transmissions. If the CAN is still in Bus Off state after Sleep mode is left, it continues counting the 128 x 11 consecutive recessive bits.

Note: CAN wakes up from a Sleep mode when SOFTRES bit in CANCTL0 register is asserted. However, this mechanism is not a recommended approach for wake-up because after assertion of SOFTRES bit in CANCTL0 register, all registers come to a known default reset state. Also, in this approach, pending actions before entering into Sleep mode, for example, copying of RxBG into RxFG will not be executed as CAN would already be in SOFTRES state.

8.8.5 Soft Reset Mode

In Soft Reset mode, any ongoing transmission or reception is aborted and synchronization to the bus is potentially lost causing CAN protocol violations. The CAN drives the MSCAN_TX pin to a recessive state to protect the CAN bus.

Note: Ensure the CAN is not active when a Soft Reset mode is entered. The recommended procedure is to bring the CAN into Sleep mode before setting the SFTRES bit in the CANCTL0 register.

The CAN is stopped in the Soft Reset mode. However, registers can still be accessed. This mode is used to initialize the module configuration, bit timing, and the CAN message filter. See **Section 8.7.1** for a description of the Soft Reset mode.

8.8.6 Power Down Mode

The CAN is in Power Down mode when:

- The CPU is in Stop mode, or
- The CPU is in Wait mode and CSWAI bit is set

When entering the Power Down mode, the CAN immediately stops all ongoing transmissions and receptions, potentially causing CAN protocol violations. Please refer to **Table 8-14.**

Note: Ensure the CAN is not active when the Power Down mode is entered. The recommended procedure is to bring the CAN into Sleep mode before the stop or wait instruction is executed, if CSWAI is set.

To protect the CAN bus system from fatal consequences of violations to the above rule, the CAN drives the MSCAN_TX pin into a recessive state.

In the Power Down mode, all clocks are stopped and no registers can be accessed.

8.8.7 Programmable Wake-Up Function

The CAN can be programmed to apply a low-pass filter function to the MSCAN_RX input line while in Sleep mode. See control bit WUPM in **Section 8.7.2, “CAN Control Register 1 (CANCTL1)”.** This feature can be used to protect the CAN from wake-up because of short glitches on the CAN bus lines. These glitches can result from electromagnetic interference within noisy environments. The wake-up filter function, filtering the glitches on CAN bus of duration less than 5µs, is programmable through the bit Wake-Up mode in CANCTL1 register.

8.9 Interrupt Operation

The CAN supports four interrupt vectors mapped onto 11 different interrupt sources. any of the interrupt sources can be individually masked. For details, see **Section 8.7.5, "CAN Receiver Flag Register (CANRFLG)"** to **Section 8.7.8.**

- Transmit Interrupt—At least one of the three Transmit Buffers (TB0, TB1, and TB2) is empty, or is not scheduled. It can be loaded to schedule a message for transmission. The TXE[2:0] flag of the empty message buffer is set. Please see **Table 8-11.**
- Receive Interrupt—A message is successfully received and loaded into the foreground receive buffer. This interrupt is generated immediately after recognizing the EOF symbol. The RXF flag is set
- Wake-up Interrupt—Activity on the CAN bus occurred during CAN internal Sleep mode
- Error Interrupt—An overrun, error, or warning condition occurred. The CAN receiver flag register indicates one of the following conditions:
  - Overrun—An overrun condition occurred. The overrun condition is described in **Section 8.4.1.3, "Receive Structures"**
  - Receiver Warning—The Receive Error counter reached the CPU warning limit of 96
  - Transmitter Warning—The Transmit Error counter reached the CPU warning limit of 96
  - Receiver Error Passive—The Receive Error counter exceeded the error passive limit of 127 and CAN went into the Error Passive state
  - Transmitter Error Passive—The Transmit Error counter exceeded the Error Passive limit of 127 and CAN went into the Error Passive state
  - Bus Off—The Transmit Error counter exceeded 255 and CAN went into the Bus Off state

Note: Error interrupt signal to CPU is asserted when exceeding the threshold condition of interrupt and when it is dropping below the threshold condition for each cause of the error interrupt generation. This ensures CPU will come to know the updated status of the cause of the error interrupt generation. Hence, CPU need not waste cycles in just polling for the event.

For Instance, error interrupt is asserted when transmitter error count crosses the threshold of 96, TX warning error, from <96 to ≥96, and also from ≥96 to <96. Similarly, for receiver warning case, when the Rx error count is crossing from <128 to ≥128 as well as from ≥128 to <128, error interrupt is asserted. Similar rules apply to other error flags.

When CPU writes 1 into any of the error interrupt causing flags the error interrupt signal gets denied, if the interrupt is already asserted, even though the condition to set the flag continues to remain active. That means, a write of 1 by CPU will *shield* only the interrupt signal to CPU and does not update the status of the flag.

8.9.1 Interrupt Acknowledge

Interrupts are directly associated with one or more status flags in either the **CAN Receiver Flag Register (CANRFLG)** or the **CAN Transmitter Flag Register (CANTFLG)**. Interrupts are pending as long as one of the corresponding flags is set. Interrupt flags must be reset within the interrupt handler to re-enable the interrupt. The flags are reset by writing a 1 to the corresponding bit in the flag register. A flag cannot be cleared if the respective condition still prevails, except in the case of an error interrupt.

If an error interrupt is the cause of the interrupt, the CPU has to write 1 in the corresponding flags in the above registers to clear the interrupt signal to the CPU. In this case, even though error interrupt flags status is set. That is, the error counter value is still above the defined threshold value and interrupt still gets denied when the CPU writes 1 into the corresponding bit positions causing error interrupt.

Note: There is a one cycle delay in the clearing of the interrupt request once the interrupt flag has been cleared. It is recommended to clear the interrupt flag early in the interrupt handler routine.

Note: Bit manipulation instructions (BSET) must not be used to clear interrupt flags.

8.9.2 Interrupt Sources

The CAN supports four interrupt functions, as shown in **Table 8-15.**

Note: The vector addresses and the relative interrupt priority are determined at the chip level.

Table 8-15. CAN Interrupt Sources

Interrupt Function	Interrupt Flag	Local Enable
Wake-Up	WUPIF	WUPIE
Error Interrupts	RWRNIF	RWRNIE
	TWRNIF	TWRNIE
	RERRIF	RERRIE
	TERRIF	TERRIE
	BOFFIF	BOFFIE
	OVRIF	OVRIE
Receive	RXF	RXFIE
Transmit	TXE0	TXEIE0
	TXE1	TXEIE1
	TXE2	TXEIE2

8.9.3 Recovery from Stop or Wait

The CAN can recover from Stop or Wait through the Wake-Up Interrupt. This interrupt can only occur if CPU puts CAN into Sleep mode before entering into the Stop or Wait modes, then the Wake-Up Interrupt is enabled, that is WUPIE equals 1. For details about entering into Sleep mode, please refer to **Section 8.8.4.** However, if CAN is in Sleep mode and a Soft Reset is asserted, CAN would not wake up the CPU. The CPU has to put CAN in Sleep mode again, for it to wake up the CPU.

Table 8-16. Document Revision History for Chapter 8

Version History	Description of Change
Rev. 8	Formatting, layout, spelling, and grammar corrections. Added revision history table. In Figure 8-10, corrected the name of bit 6 of the CANCTL0 register (was RSACT, is RXACT).

Chapter 9
Analog-to-Digital Converter (ADC)

9.1 Introduction

Digital signal controllers 56F801, 56F803, and 56F805 each have a dual, four-input, Analog-to-Digital Converter (ADC) named ADCA totaling eight analog inputs.

The 56F802 has two, 12-bit Analog-to-Digital Converters (ADCs) with a total of five analog inputs: ANA2-3 on the first ADC module and ANA4,6,7 on the second ADC channel. Differential pair inputs are only available for pairs ANA2 and ANA3, and ANA6 and ANA7.

The 56F807 has two dual, four-inputs, 12-bit ADCs named ADCA and ADCB, totaling 16 analog inputs. While the registers are named identically, they have different addresses.

9.2 Features

Each Analog-to-Digital Converter (ADC) consists of a digital control module and two analog Sample and Hold (S/H) circuits. ADC features:

- 12-bit resolution
- Sampling rate up to 1.66 million samples per second[1]
- Maximum ADC clock frequency is 5MHz with 200ns period
- Single conversion time of 8.5 ADC clock cycles (8.5 x 200ns = 1.7μs)
- Additional conversion time of 6 ADC clock cycles (6 x 200ns = 1.2μs)
- Eight conversions in 26.5 ADC clock cycles (26.5 x 200ns = 5.3μs) using Simultaneous mode
- ADC can be synchronized to the PWM via the sync signal
- Simultaneous or sequential sampling
- Internal multiplexer to select two of eight inputs
- Ability to sequentially scan and store up to eight measurements

1. Once in Loop mode, the time between each conversion is 6 ADC clock cycles (1.2μs). Using simultaneous conversion, two samples can be obtained in 1.2μs. Samples per second is calculated according to 1.2μs per two samples or 1666666 samples per second.

- Ability to simultaneously sample and hold two inputs per dual ADC block
- Optional interrupts at end of scan, if an out-of-range limit is exceeded (high or low) or at Zero Crossing
- Optional sample correction by subtracting a pre-programmed offset value
- Signed or unsigned result
- Single ended or differential inputs

9.3 Block Diagram

Figure 9-1 illustrates the ADC module.

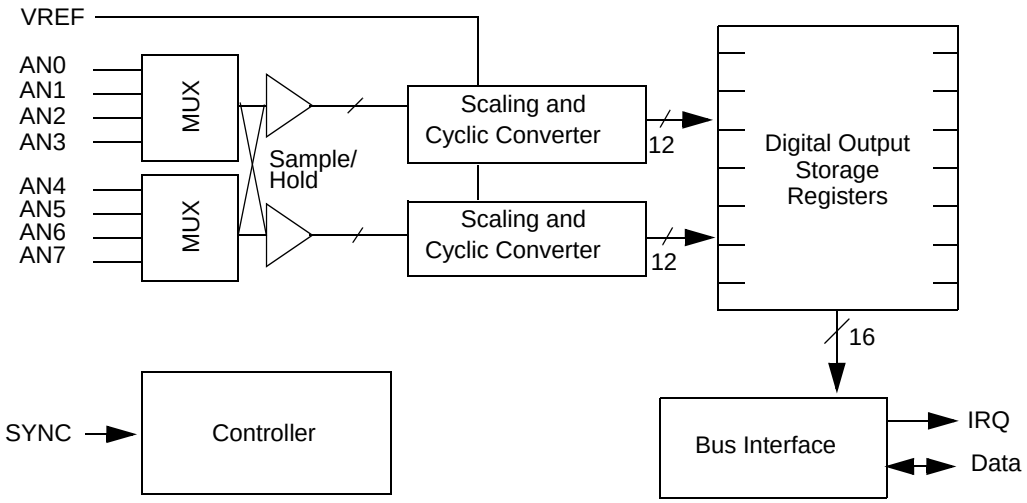


Figure 9-1. ADC Block Diagram

9.4 Functional Description

The ADC function, illustrated in **Figure 9-1,** consists of an eight-channel input select function and two independent Sample and Hold (S/H) circuits feeding separate 12-bit ADCs. The two separate converters store their results in an accessible buffer, awaiting further processing by the internal functions. The conversion process is either initiated by a SYNC signal or a write to the Control Register (ADCR1) START bit. Please see **Section 9.7.1.3, "START Conversion (START)—Bit 13".**

The channel select MUX will accommodate either current injection or current pulling protection at deselected analog inputs. This current sourcing, or pulling at the deselected inputs will not interfere with the sample being acquired at the selected input. The current being injected, or pulled is being supplied from an inductive source and the deselected analog inputs must have provisions to supply or sink this current. When the inductive force is forcing or injecting a current

at a deselected input, the input must provide a path to VSSA. Likewise, when the source is pulling a current at the deselected input, the select mux must provide a path to V_{DDA}.

The ADC consists of a cyclic, algorithmic architecture using two recursive sub-ranging sections. Each sub-ranging section resolves a single bit for each conversion clock, resulting in an overall conversion rate of two bits per clock cycle. Each sub-ranging section is designed to run at a maximum clock speed of 5MHz allowing a complete 12-bit conversion accommodation in 1.2µs, not including sample or post processing time.

The ADC module performs a ratiometric conversion. For single ended measurements, the digital result is proportional to the ratio of the analog input to the reference voltage in the following equation.

Note: The ADC is a 12-bit function with 4096 possible states. However, the 12 bits have been left shifted three bits on the 16-bit data bus so its magnitude, as read from the data bus, is now 32768. In the following example, use 32768.

$$\frac{VIN}{32768} \times V_{REF} = SingleEndedVoltage$$

Note: The result is rounded to the nearest LSB.

V_{IN} = Applied voltage

V_{REF} = External pin on the device

Starting a single conversion actually begins a sequence of conversions, or a scan. A conversion, or scan, can sample and convert up to eight unique single-ended channels, up to four differential channel pairs, or any combination of these.

The ADC can be configured to perform a single scan and halt, perform a scan whenever triggered, or perform the programmed scan sequence repeatedly until manually stopped.

The dual ADC can be configured for either *sequential or simultaneous* conversion. When configured for sequential configuration, up to eight channels, single-ended connection, can be sampled and stored in any order specified by the channel list register. During a simultaneous conversion, both S/H circuits are used to *capture two different channels at the same time*. This configuration places the restriction a single channel may not be sampled by both S/H circuits simultaneously.

The Channel List (ADLST) register is programmed with the scan order of the desired channels.

Optional interrupts can be generated at the end of the scan sequence. An optional interrupt can be used if a channel is *out of range*. *Range* is determined by the High and Low Limit registers, or by several different Zero Crossing conditions.

The method for initiating a conversion, or a scan consisting of multiple conversions, is programmable. It can be set to be either the sync signal or *a write to* the Control (ADCTL) register START bit. If a scan is initiated while another scan is in process, the start signal is ignored until the conversion is complete, or when the Conversion in Progress (CIP) bit breaks to 0, ADC is in its idle state when the CIP is low.

9.4.1 Differential Inputs

The number of conversions in a scan is programmable from one-to-eight separate channels when all inputs are configured as single-ended, or programmable from one-to-four, if all inputs are configured as differential.

Note: The ADC is a 12-bit function with 4096 possible states. However, the 12 bits have been left shifted three bits on the 16-bit data bus so its magnitude, as read from the data bus, is now 32768. In the following example use 32768.

When converting differential measurements, the following formula is useful:

$$6.6\text{ V} \times \frac{\textit{(Value Read)}}{32768} - 3.3\text{ V} = \text{Differential Voltage}$$

Here are a few examples:

Typical reading:

$$\left(6.6V \times \frac{22112}{32768}\right) - 3.3V = 1.15V$$

Most negative reading possible:

$$\left(6.6V \times \frac{0}{32768}\right) - 3.3V = -3.3V$$

Most positive reading possible:

$$\left(6.6V \times \frac{32768}{32768}\right) - 3.3V = 3.3V$$

A 0 reading (hex values shown):

$$\left(6.6V \times \frac{3FFC}{7FF8}\right) - 3.3V = 0V$$

To get dependable data from differential measurements, the voltage applied to the +/- inputs must be symmetric about $V_{DDA}/2$. Also, the V_{REF} input must be tired to the same potential as V_{DDA}.

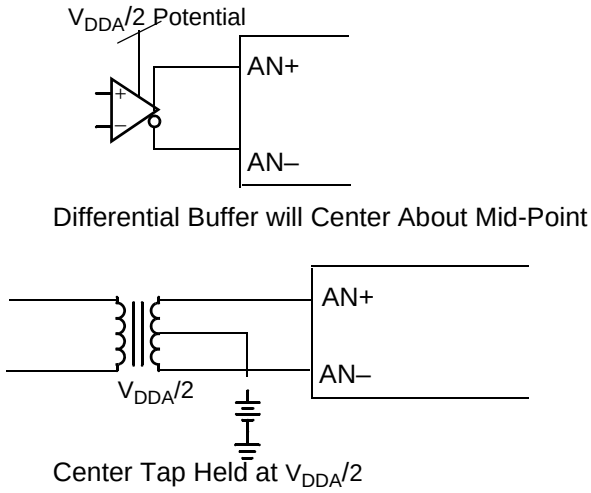


Figure 9-2. Typical connections for Differential Measurements

A mix and match combination of single-ended and differential configurations may exist. For example:

- AN0 and AN1 differential, AN2 and AN3 single-ended
- AN4 and AN5 differential, and AN6 and AN7 single-ended

If the scan parameter indicates more than one conversion, a second conversion is immediately initiated following the completion of the first conversion.

The ADC can be *configured for either sequential or simultaneous conversion*. When configured for *sequential configuration*, up to eight-channels, single-ended connection, can be sampled and stored in any order specified by the Channel List register. During a *simultaneous conversion*, both S/H circuits are used to capture two different channels simultaneously. A simultaneous configuration restricts concurrent channels from having same channel number. Channels must be unique.

The Channel List register is programmed with the order of the desired channels. Additionally, scan sequences can be repeated infinitely by configuring the ADC for Loop mode. Optional interrupts can be generated at the end of the scan sequence if a channel is out of range as determined by the High and Low Limit registers, or at several different Zero Crossing conditions.

The method for initiating a conversion, or a scan, consisting of multiple conversions, is programmable. It can be set to be either the sync signal or a write to the ADCR1 START bit. If a

scan is initiated while another scan is in process, the start signal is ignored or delayed until the analog core is able to start another conversion cycle.

9.5 Timing

Figure 9-3 illustrates a timing diagram for the ADC module.

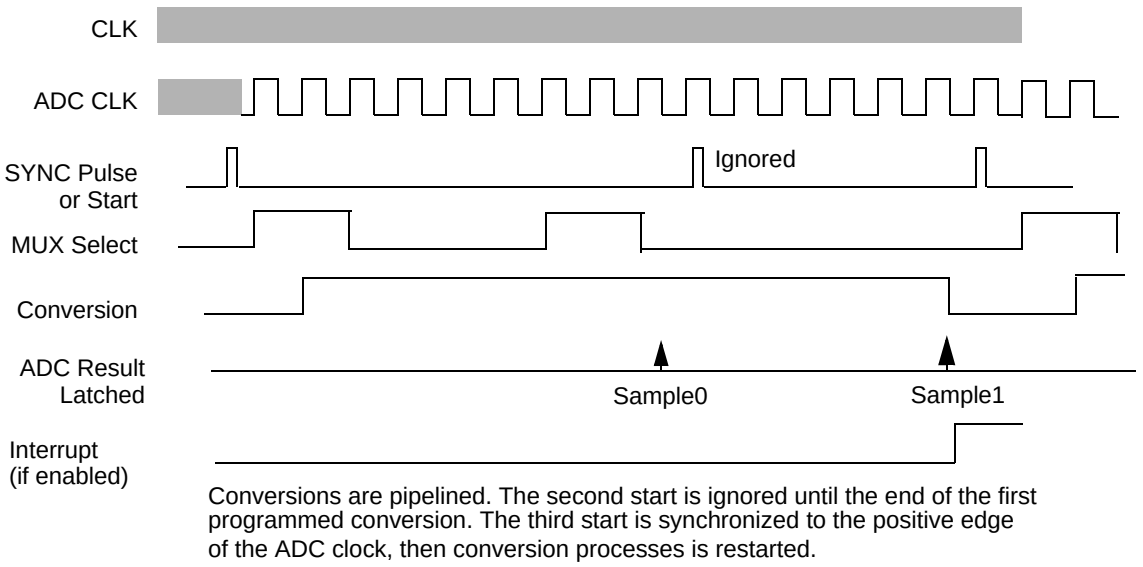


Figure 9-3. ADC Timing

ADC_CLK is derived from IPBus clock. The frequency relationship is programmable.

A conversion is initiated by a sync pulse originating from the Timer module, and can be optionally triggered by PWM, illustrated in **Figure 9-3,** or by a write to the ADCR1 START bit. In the clock cycle following this pulse the conversion is initiated. The ADC clock, ADC_CLK, period is determined by the DIV[3:0] value in the ADCR2. The positive edge of the ADC clock aligns to the positive edge of the IPBus clock (CLK).

The first conversion takes 8.5 ADC clocks to be valid. Then, each additional sample only takes six ADC clocks. The Start Conversion command is latched and the real conversion process is synchronized to the positive edge of the ADC clock. Because the conversion is a pipeline process, once the last sample has been acquired in the S/H, the ADC cannot be restored until the pipeline is emptied. However, the conversion cycle can be aborted by issuing a stop command.

9.6 Pin Descriptions

For *each* ADC, the letter of the module also becomes part of the input pin name.

ADCA has the following pins: ANA0, ANA1, ANA2, ANA3, ANA4, ANA5, ANA6, ANA7. ADCB has the following pins: ANB0, ANB1, ANB2, ANB3, ANB4, ANB5, ANB6, ANB7 The 807 has two reference pins: V_{REF} and $V_{REF\backslash 2}$.

9.6.1 Analog Input Pins (AN0–AN7)

Each ADC module has eight analog input pins. Each of the pins is connected to an internal multiplexer. The multiplexer selects the analog voltage to be converted. The eight analog input pins are subdivided into two sets of four. Each *set of four* inputs has its own S/H circuit. This configuration allows for simultaneous sampling of two selected channels. Two channels from the same subgroup can be sampled simultaneously when the channels are exchanged, so the two channels are on different subgroups. An equivalent circuit for an analog input is illustrated below.

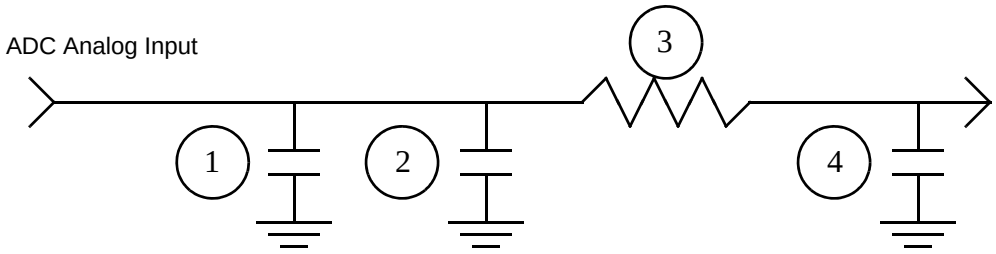


Figure 9-4. Equivalent Analog Input Circuit

1. Parasitic capacitance due to package, pin-to-pin, and pin-to-package base coupling. 1.8pf
2. Parasitic capacitance due to the chip bond pad, ESD protection devices and signal routing. 2.04pf
3. Equivalent resistance for the ESD isolation resistor and the Channel Select mux. 500ohms
4. Sampling capacitor at the Sample and Hold circuit. 1pf

9.6.2 ADC Channel Control

Because the two ADCs and their associated Sample and Hold circuits may calibrate at slightly different readings, it is useful to predict the assignment of the ADC, ADC1 or ADC2 to a given pin or channel. If this factor is taken into consideration, accuracy of readings may be enhanced.

The assignment is automatic, yet predictable. In the case of standard operation, it is very easy to predict. The assignment depends on the conditions outlined below.

- Standard Operation
  - — Pins AN0-AN3 are dedicated to the Sample and Hold circuit feeding ADC1
  - — Pins AN4-AN7 are dedicated to the Sample and Hold circuit feeding ADC2

— Each Sample and Hold circuit is dedicated to a specific ADC. They are not shared. The sample and hold servicing AN0-AN3 is dedicated to ADC1. The sample and hold servicing AN4-AN7 is dedicated to ADC2.

- Swap Operation
  — A channel swap between the two ADCs only occurs when two channels with the *same subgroup of four*, such as AN0-AN3 or AN4-AN7, are selected for a *simultaneous* conversion. Therefore, if channels AN0 and AN1 are selected for a simultaneous conversion, channel AN1 will be swapped with channel AN5 and sent to the Sample and Hold circuit dedicated to ADC2. Conversely, if channels AN4 and AN5 are selected for a simultaneous conversion, channel AN5 will be swapped with channel AN2 and sent to the Sample and Hold circuit dedicated to ADC1.
  — A swap is initiated when the internal control logic senses two channels within the same group are selected while the chip is being asked to perform a simultaneous conversion.

9.6.3 Voltage Reference Pin (V_{REF})

The reference voltage all analog inputs are measured against is provided by the difference between V_{REF} and V_{SSA}, analog ground. The V_{REF} signal is nominally set to the highest value achieved by any of the analog input signals. This reference voltage should be provided from a low noise filtered source. Bypass capacitors must be connected between V_{REF} and V_{SSA}.

The V_{REF} signal is expected to be as noise-free as possible for the ADC to maintain good accuracy. Any noise residing on the V_{REF} voltage is directly transferred to the digital result.

9.6.4 Supply Pins (V_{DDA}, V_{SSA})

Dedicated power supply pins are provided for the purposes of reducing noise coupling and to improve accuracy. The power provided to these pins is suggested to come from a low noise filtered source. Uncoupling capacitors should be connected between V_{DDA} and V_{SSA}.

9.7 Register Definitions

A prefix is added to each register reflecting which ADC module is being accessed: ADCA or ADCB. For example, the ADCR1 for the ADCA module is called ADCA_ADCR1.

Each 56F801/803/805/807 ADC module has the following registers:

- ADC Control Register 1(ADCR1)
- ADC Control Register 2 (ADCR2)
- ADC 0 Crossing Control (ADZCC) register
- ADC Channel List 1(ADLST1) register
- ADC Channel List 2 (ADLST2) register
- ADC Sample Disable (ADSDIS) register
- ADC Status (ADSTAT) register
- ADC Limit Status (ADLSTAT) register
- ADC 0 Crossing Status (ADZCSTAT) register
- ADC Result (ADRSLT0–7) registers
- ADC Low Limit (ADLLMT0–7) registers
- ADC High Limit (ADHLMT0–7) registers
- ADC Offset (ADOFS0–7) registers

For information about ADCA register addresses, please refer to **Table 3-24.** Information about the ADCB register addresses is found in **Table 3-25.**

Table 9-1. ADC Memory Map

Device	Register Map	
801/802/ 803/805	ADCA_BASE	$0E80
	ADCB_BASE	$0EC0
807	ADCA_BASE	$1280
	ADCB_BASE	$12C0

The actual memory address of a register is the sum of a base address and the registers' address offset. The base address is defined at the core. The address offset is defined at the module level.

Please reference **Table 3-11.** The base address given for each register will be either ADCA_BASE or ADCB_BASE depending on which ADC is being used.

Table 9-2. ADC Register Summary

Address Offset	Register Acronym	Register Name	Access Type	Register Location
Base + $0	ADCR1	Control Register 1	Read/Write	Section 9.7.1
Base + $1	ADCR2	Control Register 2	Read/Write	Section 9.7.2
Base + $2	ADZCC	Zero Crossing Control Register	Read/Write	Section 9.7.3
Base + $3	ADLST1	Channel List Register 1	Read/Write	Section 9.7.4
Base + $4	ADLST2	Channel List Register 2	Read/Write	
Base + $5	ADSDIS	Sample Disable Register	Read/Write	Section 9.7.5
Base + $6	ADSTAT	Status Register	Read/Write	Section 9.7.6
Base + $7	ADLSTAT	Limit Status Register	Read/Write	Section 9.7.10
Base + $8	ADZCSTAT	Zero Crossing Status Register	Read/Write	Section 9.7.8
Base + $9-10	ADRSLT0-7	Result Registers 0-7	Read/Write	Section 9.7.9
Base + $11-18	ADLLMT0-7	Low Limit Registers 0-7	Read/Write	Section 9.7.10
Base + $19-20	ADHLMT0-7	High Limit Registers 0-7	Read/Write	
Base + $21-28	ADOFS0-7	Offset Registers 0-7	Read/Write	Section 9.7.11

Bit fields of each of the 40 registers are summarized in **Figure 9-5.** Details of each follow.

Add. Offset	Register Name		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$0	ADCR1	R	0	STOP	0	SYNC	EOSIE	ZCIE	LLMTIE	HLMTIE	CHNCFG				0	SMODE		
		W			START													
$1	ADCR2	R	0	0	0	0	0	0	0	0	0	0	0	0	DIV			
		W																
$2	ADZCC	R	ZCE7		ZCE6		ZCE5		ZCE4		ZCE3		ZCE2		ZCE1		ZCE0	
		W																
$3	ADLST1	R		SAMPLE3				SAMPLE2				SAMPLE1				SAMPLE0		
		W																
$4	ADLST2	R		SAMPLE7				SAMPLE6				SAMPLE5				SAMPLE4		
		W																
$5	ADSDIS	R	TEST		0	0	0	0	0	0	DS7	DS6	DS5	DS4	DS3	DS2	DS1	DS0
		W																
$6	ADSTAT	R	CIP	0	0	0	EOSI	ZCI	LLMTI	HLMTI	RDY							
		W																
$7	ADLSTAT	R	HLS								LLS							
		W																
$8	ADZCSTAT	R	0	0	0	0	0	0	0	0	ZCS							
		W																
$9	ADRSLT0	R	SEXT	RSLT												0	0	0
		W																
$A	ADRSLT1	R	SEXT	RSLT												0	0	0
		W																
$B	ADRSLT2	R	SEXT	RSLT												0	0	0
		W																
$C	ADRSLT3	R	SEXT	RSLT												0	0	0
		W																
$D	ADRSLT4	R	SEXT	RSLT												0	0	0
		W																
$E	ADRSLT5	R	SEXT	RSLT												0	0	0
		W																
$F	ADRSLT6	R	SEXT	RSLT												0	0	0
		W																
$10	ADRSLT7	R	SEXT	RSLT												0	0	0
		W																
$11	ADRLLMT0	R	0	LLMT												0	0	0
		W																
$12	ADRLLMT1	R	0	LLMT												0	0	0
		W																
$13	ADRLLMT2	R	0	LLMT												0	0	0
		W																
$14	ADRLLMT3	R	0	LLMT												0	0	0
		W																
$15	ADRLLMT4	R	0	LLMT												0	0	0
		W																
$16	ADRLLMT5	R	0	LLMT												0	0	0
		W	0															
$17	ADRLLMT6	R	0	LLMT												0	0	0
		W																
$18	ADRLLMT7	R	0	LLMT												0	0	0
		W																
$19	ADHLMT0	R	0	HLMT												0	0	0
		W																

Figure 9-5. ADC Register Map Summary

Add. Offset	Register Name		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$1A	ADHLMT1	R	0	HLMT												0	0	0
		W																
$1B	ADHLMT2	R	0	HLMT												0	0	0
		W																
$1C	ADHLMT3	R	0	HLMT												0	0	0
		W																
$1D	ADHLMT4	R	0	HLMT												0	0	0
		W																
$1E	ADHLMT5	R	0	HLMT												0	0	0
		W																
$1F	ADHLMT6	R	0	HLMT												0	0	0
		W																
$20	ADHLMT7	R	0	HLMT												0	0	0
		W																
$21	ADOFS0	R	0	OFFSET												0	0	0
		W																
$22	ADOFS1	R	0	OFFSET												0	0	0
		W																
$23	ADOFS2	R	0	OFFSET												0	0	0
		W																
$24	ADOFS3	R	0	OFFSET												0	0	0
		W																
$25	ADOFS4	R	0	OFFSET												0	0	0
		W																
$26	ADOFS5	R	0	OFFSET												0	0	0
		W																
$27	ADOFS6	R	0	OFFSET												0	0	0
		W																
$28	ADOFS7	R	0	OFFSET												0	0	0
		W																

R	0	Read a 0
W		Reserved

Figure 9-5. ADC Register Map Summary (Continued)

9.7.1 ADC Control Register 1 (ADCR1)

ADC_BASE+$0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	STOP	0	SYNC	EOSIE	ZCIE	LLMTIE	HLMTIE	CHNCFG				0	SMODE		
Write			START													
Reset	0	1	0	1	0	0	0	0	0	0	0	0	0	1	0	1

Figure 9-6. ADC Control Register 1 (ADCR1)

See Table A-8, List of Programmer's Sheets

9.7.1.1 Reserved—Bit 15

This bit is reserved or not implemented. It cannot be modified and is read as 0.

9.7.1.2 Stop (STOP)—Bit 14

When STOP is selected, the current conversion process is stopped. Any further sync pulses or modifications to the START bit are ignored until the STOP bit has been cleared. Once the ADC is in Stop mode, the results registers can be modified by the processor. Any change to the register in Stop mode is treated as if the analog core supplied the data. Therefore, Limit Checking, Zero Crossing, and associated interrupts can occur when authorized.

- 0 = Normal operation
- 1 = Stop command issued

9.7.1.3 START Conversion (START)—Bit 13

The conversion process is started by a modification to the START bit. This is a *write-only* bit. Once a Start command is issued, the conversion process is synchronized and begun on the positive edge of the next ADC clock cycle. A START bit re-started while the ADC is in a conversion cycle, is ignored. The ADC must be idle in order for it to recognize a new start-up.

- 0 = No action
- 1 = Start command is issued

9.7.1.4 SYNC Select (SYNC)—Bit 12

Conversions start by the sync input or by a write to the START bit. A re-started sync pulse is ignored while the ADC is in a conversion cycle. The ADC must be idle in order for it to recognize a new start-up.

- 0 = Conversion initiated by a write to START bit only
- 1 = Use sync input or START bit to initiate a conversion

9.7.1.5 End Of Scan Interrupt Enable (EOSIE)—Bit 11

This bit begins an interrupt to be generated upon completion of any scan and convert sequence, except for Loop Sequential or Loop Simultaneous modes. This bit is ignored when the ADC is configured for either loop modes.

- 0 = Interrupt disabled
- 1 = Interrupt enabled

9.7.1.6 Zero Crossing Interrupt Enable (ZCIE)—Bit 10

This bit starts the optional interrupt if the current result value has a sign change from the previous result as configured by the ADZCC register.

- 0 = Interrupt disabled
- 1 = Interrupt enabled

9.7.1.7 Low Limit Interrupt Enable (LLMTIE)—Bit 9

This bit enables the optional interrupt when the current result value is less than the Low Limit register value. The raw result value is compared to ADLLMT, the Low Limit register, bits LLMT[11:0], before the Offset register value is subtracted.

- 0 = Interrupt disabled
- 1 = Interrupt enabled

9.7.1.8 High Limit Interrupt Enable (HLMTIE)—Bit 8

This bit confirms the optional interrupt if the current result value is greater than the High Limit register value. The raw result value is compared to ADHLMT, the High Limit register, bits HLMT[11:0], before the Offset register value is subtracted.

- 0 = Interrupt disabled
- 1 = Interrupt enabled

9.7.1.9 Channel Configure (CHNCFG)—Bits 7–4

The inputs can be configured for either single ended or differential.

- xxx1 = AN0–AN1 configured as differential input (AN0 is + and AN1 is –)
- xxx0 = AN0–AN1 configured as single ended inputs
- xx1x = AN2–AN3 configured as differential input (AN2 is + and AN3 is –)
- xx0x = AN2–AN3 configured as single input
- x1xx = AN4–AN5 configured as differential inputs (AN4 is + and AN5 is –)
- x0xx = AN4–AN5 configured as single ended inputs
- 1xxx = AN6–AN7 configured as differential inputs (AN6 is + and AN7 is –)
- 0xxx = AN6–AN7 configured as single ended inputs

9.7.1.10 Reserved—Bit 3

This bit is reserved or not implemented. It is read as 0 and cannot be modified by writing.

9.7.1.11 Scan Mode (SMODE)—Bits 2–0

A conversion sequence can be begun by asserting the ADCR1 START bit, or by a sync pulse. A conversion sequence can take up to eight unique samples. The ADSDIS register describes which

slots are defined. A sample sequence is terminated when the first disabled sample is encountered. Analog input channels are assigned to each of the eight sample slots by the ADLST1 and ADLST2 registers. A conversion sequence can be run through just once every time a trigger occurs, or it can loop continuously. Samples may be taken one at a time, sequentially, or two at a time simultaneously.

- 000 = Once (single) sequential
  Upon start or at the first sync signal, samples are taken one at a time starting with SAMPLE0, until a first disabled sample is encountered. If no disabled sample is encountered, conversion concludes after the SAMPLE7
- 001 = Once (single) simultaneous
  Upon start or at the first sync signal, samples are taken two at a time, in the order: SAMPLE0/4, SAMPLE1/5, SAMPLE2/6, and SAMPLE3/7. The sample sequence will stop at SAMPLE3/7, or as soon as any disabled sample slot is encountered. For example, if SAMPLE0 or SAMPLE4 were disabled, no samples would be taken at all.

Note: In the once (single) sequential 000 and once (single) Simultaneous 001 modes, provided a sampling sequence has completed, a start pulse will always initiate another once (single) sequence, either sequential or simultaneous. A sync pulse must be re-armed via another write to the ADCR1 register in order for a second sampling sequence to occur.

- 010 = Loop sequential
  Upon Start or at the first sync pulse, samples are taken sequentially until a disabled sample is encountered and then the sequence is restarted. The process repeats perpetually until the STOP bit is set in ADCR1. For example, if samples zero, one, and five were enabled, the loop would look like SAMPLE0, SAMPLE1, SAMPLE0, SAMPLE1,and so on because SAMPLE 2 is disabled. While a Loop mode is running, any additional Start commands or sync pulses are ignored
- 011= Loop simultaneous
  Like once (single) simultaneous, samples are taken two at a time. The loop restarts as soon as a sample slot is encountered in either, or both, disabled samples. For example, of enabled samples zero, four, one, five, and two, the loop would look like SAMPLE0/4, SAMPLE1/5, SAMPLE0/4, and SAMPLE1/5, and so on. SAMPLE6 was disabled, therefore no data was taken in the SAMPLE2/6 slot
- 100 = Triggered sequential
  A single sequential sampling begins with every recognized start or sync pulse. Samples are taken sequentially as described above
- 101 = Triggered simultaneous
  A single simultaneous sampling begins with every recognized start or sync pulse. Samples

are taken simultaneously as previously described. Reaffirmed START bits and sync pulse presented during an active sample sequence are not recognized until the ADC is in its idle state, that is CIP low

- 110 = Reserved use
- 111 = Reserved use

9.7.2 ADC Control Register 2 (ADCR2)

ADC_BASE+$1	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	0	0	0	0	0	0	0	DIV			
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Figure 9-7. ADC Control Register 2 (ADCR2)

See Table A-8, List of Programmer's Sheets

9.7.2.1 Reserved—Bits 15–4

This bit field is reserved or not implemented. It is read as 0 and cannot be modified by writing.

9.7.2.2 Clock Divisor Select (DIV)—Bits 3–0

ADC_CLK can be run at a slower rate than the system clock. The divider circuit provides a clock of period 2N of system clock where N=DIV[3:0]+1. A divisor must be chosen so the ADC clock is within specified limits. For a IPBus clock frequency of 40MHz and a desired ADC_CLK frequency of 5MHz, a DIV[3:0] value of 3d is required. This is the equation for the Clock Divisor Select Value:

Clock Divisor Select Value =

N = DIV + 1

$F_{ADC} = (F_{IPR}) / 2N$

F_{ADC} = Analog-to-Digital Converter Frequency

F_{IPR} = Interface Peripheral Bus Clock Frequency

9.7.3 ADC Zero Crossing Control Register (ADZCC)

The Zero Crossing Control (ADZCC) register provides the ability to monitor the selected channels and determine the direction of Zero Crossing triggering the optional interrupt. Zero Crossing logic monitors only the sign change between current and previous sample. ZCE0 monitors the sample stored in ADRSLT0 and ZCE7 monitors ADRSLT7. When Zero Crossing is

disabled for a selected result register, sign changes are not monitored and updated in the ADZCSTAT, but they are properly stored in the respective result register.

ADC_BASE+$2	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read Write	ZCE7		ZCE6		ZCE5		ZCE4		ZCE3		ZCE2		ZCE1		ZCE0	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 9-8. ADC Zero Crossing Control Register (ADZCC)

See Table A-8, List of Programmer's Sheets

ZCEx—Zero Crossing enable *x*

x represents the channel number used to enable or disable Zero Crossing.

- 00 = Zero Crossing disabled
- 01 = Zero Crossing enabled for positive to negative sign change
- 10 = Zero Crossing enabled for negative to positive sign change
- 11 = Zero Crossing enabled for any sign change

9.7.4 ADC Channel List Registers (ADLST1 & ADLST2)

The channel list register contains an ordered list of the channels to be converted when the next scan is initiated. If all samples are enabled, ADSDIS register, a sequential scan of inputs proceeds in order of: SAMPLE0 through SAMPLE7. If one of the simultaneous sampling modes is selected instead, the sampling order is SAMPLE0 and SAMPLE4, SAMPLE1 and SAMPLE5, SAMPLE2 and SAMPLE6, then SAMPLE3 and SAMPLE7.

ADC_BASE+$3	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read Write		SAMPLE3				SAMPLE2				SAMPLE1				SAMPLE0		
Reset	0	0	1	1	0	0	1	0	0	0	0	1	0	0	0	0

Figure 9-9. ADC Channel List Register 1 (ADLST1)

See Table A-8, List of Programmer's Sheets

SAMPLEx[3]—Reserved, where x is the sample number, 0–7.

SAMPLEx[2:0]—Channel select, where x is the sample number, 0–7. This is a binary representation of the analog input to be converted.

.

ADC_BASE+$4	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read		SAMPLE7				SAMPLE6				SAMPLE5				SAMPLE4		
Write																
Reset	0	1	1	1	0	1	1	0	0	1	0	1	0	1	0	0

Figure 9-10. ADC Channel List Register 2 (ADLST2)

See Table A-8, List of Programmer's Sheets

Table 9-3. ADC Input Conversion for Sample Bits

SAMPLEx[2:0]	Single Ended	Differential
000	AN0	AN0+, AN1–
001	AN1	AN0+, AN1–
010	AN2	AN2+, AN3–
011	AN3	AN2+, AN3–
100	AN4	AN4+, AN5–
101	AN5	AN4+, AN5–
110	AN6	AN6+, AN7–
111	AN7	AN6+, AN7–

In any of the sequential sampling modes there are no restrictions on assigning an analog channel, AN0–AN7, to a sample. Any sample slot may contain any channel assignment.

In simultaneous modes, *any sample slot may contain any channel with one exception.* That exception is: for any sample pair, each sample slot of the pair must contain a unique analog channel assignment. See OnCE simultaneous for a description of sample pairs. For example, if program SAMPLE0 with 000, AN0 is elected, then SAMPLE4 may contain any code except 000.

When channels are configured as differential inputs, there are further restrictions. First, like the simultaneous restriction, each member of a sampling pair must be unique. A second SAMPLE0 through SAMPLE3 may only contain analog channel assignments in the range 0b000–0b011, and SAMPLE4 through SAMPLE7 may only contain analog channel assignments in the range 0b100–0b111.

No damage will occur to the ADC if these restrictions are violated; however, measurement results are undefined.

9.7.5 ADC Sample Disable Register (ADSDIS)

This register is an extension to the ADLST1and 2, providing the ability to enable only the desired samples programmed in the SAMPLE0–SAMPLE7. When power is on default all samples are enabled. For example, if in Sequential mode and DS5 is set to 1, SAMPLE0 through SAMPLE4 are sampled. However, if Simultaneous mode is selected and DS5 or DS1 is set to 1, only SAMPLE0 and SAMPLE4 are sampled.

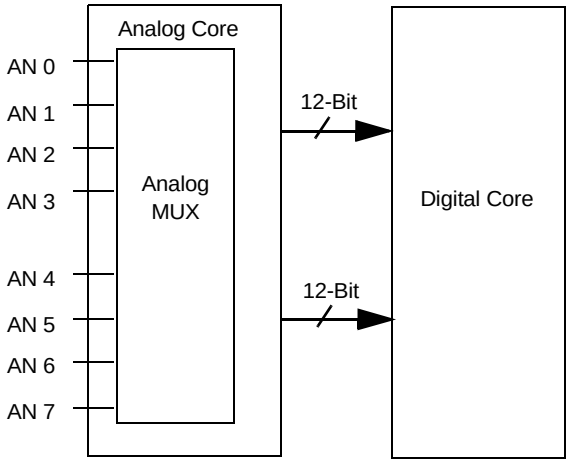


Figure 9-11. ADC Core

ADC_BASE+$5	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	TEST		0	0	0	0	0	0	DS7	DS6	DS5	DS4	DS3	DS2	DS1	DS0
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 9-12. ADC Sample Disable Register (ADSDIS)

See Table A-8, List of Programmer's Sheets

9.7.5.1 Test (TEST)—Bits 15–14

The ADC core has a built-in Test mode allowing the V_{REF} to be applied to AN0.

- 00 = Normal mode
- 01 = Test mode, $V_{REF}/2$ applied to AN0 and AN4
- 10 = Reserved
- 11 = Reserved

9.7.5.2 Reserved—Bits 13–8

This bit field is reserved or not implemented. It is read as 0 and cannot be modified by writing.

9.7.5.3 Disable Sample (DS)—Bits 7–0

The respective SAMPLEx can be enabled or disabled where x = 0–7.

- 0 = Enable SAMPLEx
- 1 = Disable SAMPLEx

Note: When TEST[1:0] is configured for Test mode, VREF is applied to AN0 and AN4 between the analog muxing and the ADC core. Only AN0 and AN4 will have valid results, so only AN0 and AN4 should be sampled.

9.7.6 ADC Status Register (ADSTAT)

This register provides the current status of the ADC module. RDYx bits are cleared by reading their corresponding result registers (ADRSLTx). HLMTI and LLMTI bits are cleared by writing a value of 1 to them. The ZCI, LLMTI, and HLMTI bits can only be cleared by clearing all asserted bits in the Zero Crossing Status (ADZCSTST) register. The EOSI bit is cleared by writing a 1 to it, that is, a write of $0800 to ADSTAT will clear the EOSI bit. ADSTAT bits are sticky. Once set to a 1 state, they require some specific action to clear them. They are not cleared automatically on the next scan sequence. These bits cannot be set by software.

ADC_BASE+$6	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	CIP	0	0	0	EOSI	ZCI	LLMTI	HLMTI	RDY [7:0]							
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 9-13. ADC Status Register (ADSTAT)

See Table A-8, List of Programmer's Sheets

9.7.6.1 Conversion in Progress (CIP)—Bit 15

This bit indicates whether a scan is in progress.

- 0 = Idle state
- 1 = A scan cycle is in progress. The ADC will ignore all sync pulses or start commands

9.7.6.2 Reserved—Bits 14–12

This bit field is reserved or not implemented. It is read as 0 and cannot be modified by writing.

9.7.6.3 End of Scan Interrupt (EOSI)—Bit 11

This bit indicates whether a scan of analog inputs have been completed since the last read of the Status register or since a reset. The EOSI bit is cleared by writing a 1 to it. This bit cannot be set by software.

- 0 = A scan cycle has not been completed, no end of scan IRQ pending
- 1 = A scan cycle has been completed, end of scan IRQ pending

9.7.6.4 Zero Crossing Interrupt (ZCI)—Bit 10

If the respective Offset register is begun by having a value greater than 0000h, Zero Crossing checking is enabled. If the Offset register is programmed with 7FF8h, the result will always be less than, or equal to zero. On the other hand, if 0000h is programmed into the Offset register, the result will always be greater than, or equal to zero and no Zero Crossing can occur because the sign of the result will not change.

- 0 = No ZCI IRQ
- 1 = Zero Crossing encountered, IRQ pending if ZCIE is set

9.7.6.5 Low Limit Interrupt (LLMTI)—Bit 9

If the respective Low Limit register is enabled by having a value other than 0000h, low limit checking is enabled.

- 0 = No low limit IRQ
- 1 = Low limit exceeded, IRQ pending if LLMTIE is set

9.7.6.6 High Limit Interrupt (HLMTI)—Bit 8

If the respective High Limit register is enabled by having a value other than 7FF8h, high limit checking is enabled

- 0 = No high limit IRQ
- 1 = High Limit exceeded, IRQ pending if HLMTIE is set

9.7.6.7 Ready Channel 7–0 (RDY)—Bits 7–0

These bits indicate channels 7 through 0 are ready to be read. These bits are cleared after a read from the respective results register.

- 0 = Channel not ready or has been read
- 1 = Channel ready to be read

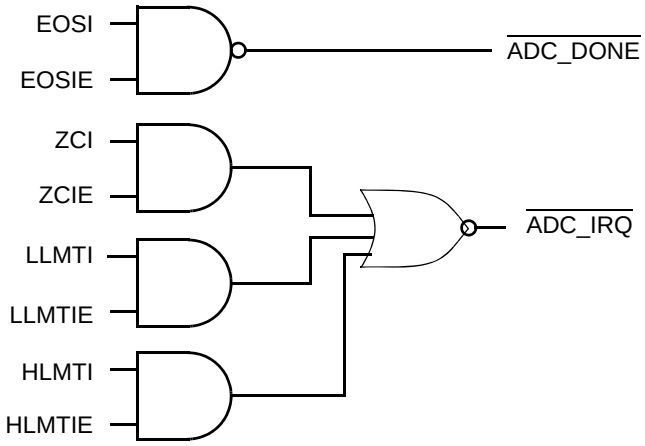


Figure 9-14. ADC Interrupt

9.7.7 ADC Limit Status Register (ADLSTAT)

The Limit Status register latches in the result of the comparison between the result of the sample and the respective Limit register (ADHLMT0-7 and ADLLMT0-7). Here is an example: If the result for the channel programmed in SAMPLE0 is greater than the value programmed into the High Limit Register 0, the HLS0 bit is set to 1. An interrupt is generated if the HLMTIE bit is set in ADCR1. A bit may only be cleared by writing a value of 1 to that specific bit. These bits are sticky. Once set, the bits require a specific modification to clear them. They are not cleared automatically by subsequent conversions.

ADC_BASE+$7	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	HLS [7:0]								LLS [7:0]							
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 9-15. ADC Limit Status Register (ADLSTAT)

See Table A-8, List of Programmer's Sheets

9.7.8 ADC Zero Crossing Status Register (ADZCSTAT)

The Zero Crossing Status (ADZCSTAT) register latches in the result of the comparison between the current result of the sample and the previous result of the same sample register. For example, if the result for the channel programmed in SAMPLE0 changes sign from the previous conversion and the respective ZCE bit in register ADZCC is set to 11b, any edge change, then the ZCS0 bit is set to 1. An interrupt is generated if the ZCIE bit is set in ADCR1. A bit can only be cleared by writing a value of 1 to that specific bit. These bits are sticky. Once set, they require a write to clear them. They are not cleared automatically by subsequent conversions.

ADC_BASE+$8	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	0	0	0	ZCS [7:0]							
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 9-16. ADC Zero Crossing Status Register (ADZCSTAT)

See Table A-8, List of Programmer's Sheets

9.7.8.1 Reserved—Bits 15–8

This bit field is reserved or not implemented. It is read as 0 and cannot be modified by writing.

9.7.8.2 Zero Crossing Status (ZCS)—Bits 7–0

- 0 = a. A sign change did not occur in a comparison between the current channelx result and the previous channelx result, or

  b. Zero crossing control is disabled for channelx in the Zero Crossing Control (ADZCC) register
- 1 = In a comparison between the current channelx result and the previous channelx result, a sign change condition occurred as defined in the Zero Crossing Control (ADZCC) register

Note: To clear a specific ZCS[7:0] bit, write a value of 1 to that specific bit.

9.7.9 ADC Result Registers (ADRSLT0–7)

The eight Result Registers contain the converted results from a scan. The SAMPLE0 result is loaded into ADRSLT0, SAMPLE1 result in ADRSLT1, and so on. In a Simultaneous Scan mode, the first channel pair designated by SAMPLE0 and SAMPLE4 in register ADLST1 are stored in ADRSLT0 and ADRSLT4, respectively.

ADC Result Register 0—Address: ADC_BASE + $9
ADC Result Register 1—Address: ADC_BASE + $A
ADC Result Register 2—Address: ADC_BASE + $B
ADC Result Register 3—Address: ADC_BASE + $C
ADC Result Register 4—Address: ADC_BASE + $D
ADC Result Register 5—Address: ADC_BASE + $E
ADC Result Register 6—Address: ADC_BASE + $F
ADC Result Register 7—Address: ADC_BASE + $10

ADC_BASE+$9-10	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	SEXT	RSLT												0	0	0
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 9-17. ADC Result Registers (ADRSLT0–7)

See Table A-8, List of Programmer's Sheets

9.7.9.1 Sign Extend (SEXT)—Bit 15

SEXT is the Sign-Extend bit of the result. SEXT set to one implies a negative result zero, a positive is one. If positive results are required, then the respective Offset register must be set to a value of zero.

9.7.9.2 Digital Result of the Conversion (RSLT)—Bits 14–3

ADRSLT can be interpreted as either a signed integer or a signed fractional number. As a signed fractional number, the ADRSLT can be used directly. As a signed integer, it is an option to right shift with sign extend (ASR) three places and interpret the number, or accept the number as presented, knowing there are missing codes. The lower three bits are always going to be zero.

Negative results (SEXT = 1) are always presented in two’s complement format. If it is a requirement of your application the result registers will always be positive, the Offset registers must always be set to zero.

The interpretation of the numbers programmed into the Limit and Offset registers (ADLLMT, ADHLMT, and ADOFS) should match your interpretation of the result register.

9.7.9.3 Reserved—Bits 2–0

This bit field is reserved or not implemented. It is read as 0 and cannot be modified by writing.

Each resulting register may only be modified when the ADC is in Stop mode, that is the STOP bit is set to one. This write operation is treated as if it came from the ADC analog core; therefore the Limit Checking, Zero Crossing, and Offset registers function as if in Normal mode. For example, if the STOP bit is set to one and the processor writes to ADRSLT5, the data written to the ADRSLT5 is muxed to the digital core, processed and stored into ADRSLT5 as if the analog core provided the data. This test data must be justified as illustrated by the ADRSLT register definition.

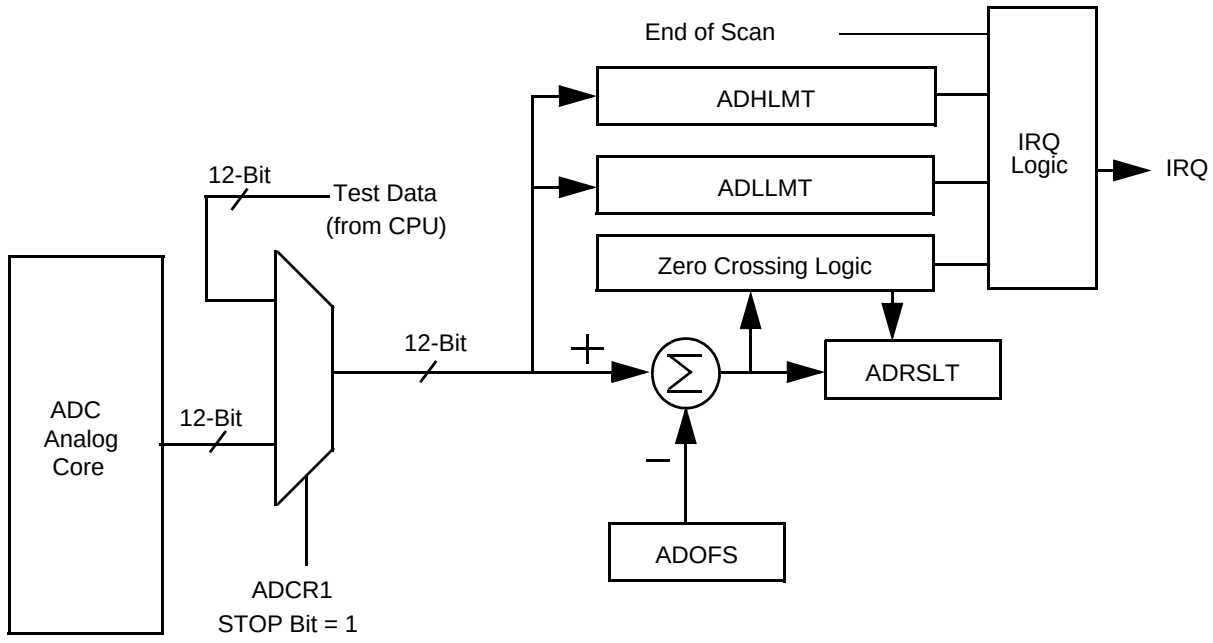


Figure 9-18. Result Register Data Manipulation

The result value sign is determined from the ADC unsigned result minus the respective Offset register value. If the Offset register is programmed with a value of zero, the Result Register value is unsigned and equals the cyclic converter unsigned result. The range of the result (ADRSLT) is $F001–$7FF8. If the Offset (ADOFS) register is set to all zeros, the range is $F001–$7FF8. This is equal to the raw value of the ADC core.

9.7.10 ADC Low and High Limit Registers (ADLLMT0–7) and (ADHLMT0–7)

The Limit registers are programmed with the value the result is compared against. The High Limit register is used for comparison with Result>High Limit. The Low Limit register is used for the comparison with Result<Low Limit. The limit checking can be disabled by programming the respective Limit register with $FFFF for High Limit and $0000 for the Low Limit register. The power-up default is limit checking disabled.

ADC Low Limit Register 0—Address:ADC_BASE + $11
ADC Low Limit Register 1—Address:ADC_BASE + $12
ADC Low Limit Register 2—Address:ADC_BASE + $13
ADC Low Limit Register 3—Address:ADC_BASE + $14
ADC Low Limit Register 4—Address:ADC_BASE + $15
ADC Low Limit Register 5—Address: ADC_BASE + $16
ADC Low Limit Register 6—Address:ADC_BASE + $17
ADC Low Limit Register 7—Address:ADC_BASE + $18

BASE + $11-18	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	LLMT												0	0	0
Write																
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 9-19. ADC Low Limit Registers (ADLLMT0-7)

See Table A-8, List of Programmer's Sheets

ADC High Limit Register 0—Address:ADC_BASE + $19
ADC High Limit Register 1—Address:ADC_BASE + $1A
ADC High Limit Register 2—Address:ADC_BASE + $1B
ADC High Limit Register 3—Address:ADC_BASE + $1C
ADC High Limit Register 4—Address:ADC_BASE + $1D
ADC High Limit Register 5—Address:ADC_BASE + $1E
ADC High Limit Register 6—Address:ADC_BASE + $1F
ADC High Limit Register 7—Address:ADC_BASE + $20

BASE + $19-20	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	HLMT												0	0	0
Write																
RESET	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0

Figure 9-20. ADC High Limit Registers (ADHLMT0-7)

See Table A-8, List of Programmer's Sheets

9.7.11 ADC Offset Registers (ADOFS0–7)

Value of the Offset register is used to correct the ADC result before it is stored in the ADRSLT registers.

ADC Offset Register 0—Address: ADC_BASE + $21
ADC Offset Register 1—Address: ADC_BASE + $22
ADC Offset Register 2—Address: ADC_BASE + $23
ADC Offset Register 3—Address: ADC_BASE + $24
ADC Offset Register 4—Address: ADC_BASE + $25
ADC Offset Register 5—Address: ADC_BASE + $26
ADC Offset Register 6—Address: ADC_BASE + $27

ADC Offset Register 7—Address: ADC_BASE + $28

ADC_BASE+$21-28	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	OFFSET												0	0	0
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 9-21. ADC Offset Registers (ADOFS0–7)

See Table A-8, List of Programmer's Sheets

The offset value is subtracted from the ADC result. In order to obtain unsigned results, the respective Offset register should be programmed with a value of $0000, thus giving a result range of $0000 to $7FF8.

9.8 Starting a Conversion if Status of ADC is Unknown

The ADC module is unique because it can be considered to run asynchronously with the rest of the chip. Further, its function is pipelined, greatly improving throughput but complicating its control slightly.

In normal operating scenarios, the controlling software will be fully aware of the status of the ADC. However, if software design is a conversion and may be started when the ADC state is not known, then consider the following guideline.

Setting the START bit in the ADCR1 does not really set a bit. Instead, it generates an internal start pulse beginning the conversion process on the first available rising edge of the ADC clock. Setting the START bit a second time immediately after the first start may not generate a second start pulse. If there is any chance software routines may be asserting start without waiting or knowing a previous conversion has completed, then write to the STOP bit of ADCR1. Secondly, write to the START bit. Writing to Stop ensures an active conversion sequence is halted and the ADC is brought back to its idle condition. Once in its idle condition, writing to the START bit is assured to be recognized.

Table 9-4. Document Revision History for Chapter 9

Version History	Description of Change
Rev. 8	Formatting, layout, spelling, and grammar corrections. Added revision history table. In **Figure 9-5**, changed the name of bits 7:4 in ADCR1 (was CHNCRG, is CHNCFG). Changed the name of bit 8 in ADSTAT (was HLMT, is HLMTI) in **Figure 9-5** and **Figure 9-13**

Chapter 10
Quadrature Decoder

10.1 Introduction

Each Quad Decoder module has four input signals and circuitry called the switch matrix. Each has three modes. It provides a means to share input pins with an associated time module.

Quad Decoder refers to the module on the 56F803/805/807 and an external device mounted on a motor shaft.

The 56F801 and 56F802 have no Quad Decoder

The 56F803 has one Quad Decoder

The 56F805 and 56F807 have two Quad Decoders

On the 56F803/805/807, the Quad Timer modules A and B share pins with Quad Decoder modules numbers zero and one. If the shared pins are not configured as timer outputs, the pins are available for use as inputs to the Quad Decoder modules. Each Quad Decoder module has four input signals:

1. PHASEA
2. PHASEB
3. INDEX
4. HOME

10.2 Features

- Includes logic to decode quad signals
- Configurable digital filter for inputs
- 32-bit position counter
- 16-bit position difference register
- Maximum count frequency equals the peripheral clock rate
- Position counter can be initialized by SW or external events
- Preloadable 16-bit revolution counter

- Inputs can be connected to a general purpose timer to aid low speed velocity measurements
- Quad Decoder filter can be bypassed
- A Watchdog Timer to detect a non-rotating shaft condition

10.3 Pin Descriptions

With the 56F803, input pins are labeled PHASEA0, PHASEB0, INDEX0 and HOME0. On the 56F805/807 they are PHASEA0, PHASEB0, INDEX0, HOME0, PHASEA1, PHASEB1, INDEX1 and HOME1.

10.3.1 Phase A Input (PHASEA)

The PHASEA input can be connected to one of the phases from a two-phase shaft quad encoder output. It is used by the Quad Decoder module in conjunction with the PHASEB input to indicate a decoder increment has passed, and to calculate its direction. PHASEA is the leading phase for a shaft rotating in the positive direction. PHASEA is the trailing phase for a shaft rotating in the negative direction. It can also be used as the single input when the Quad Decoder module is used as a single phase pulse accumulator.

The PHASEA input is also an input capture channel for one of the timer modules. The exact connection to the Timer module is specified in **Table 10-1.**

10.3.2 Phase B Input (PHASEB)

The PHASEB input is used as one of the phases from a two-phase Shaft Quad Encoder output. It is used by the Quad Decoder module in conjunction with the PHASEA input indicating a decoder increment has passed, and to calculate its direction. PHASEB is the trailing phase for a shaft rotating in the positive direction. PHASEB is the leading phase for a shaft rotating in the negative direction.

The PHASEB input is also an input capture channel for one of the timer modules. The exact connection to the Timer module is specified in **Table 10-1.**

10.3.3 Index Input (INDEX)

Normally connected to the Index Pulse Output of a Quad Encoder, this pulse can optionally reset the position counter and the pulse accumulator of the Quad Decoder module. It also causes a change of state on the revolution counter. The direction of this change, increment or decrement, is calculated from the PHASEA and PHASEB inputs.

The INDEX input is also an input capture channel for one of the timer modules. The exact connection to the Timer module is specified in **Table 10-1.**

10.3.4 Home Switch Input (HOME)

The HOME input can be used by the Quad Decoder and the Timer module. This input can be used to trigger the initialization of the position counters (UPOS and LPOS). Often this signal is connected to a sensor signaling the motor or machine notifying it has reached a defined home position. This general-purpose signal is also connected to the Timer module as specified in **Table 10-1.**

10.4 Register Summary

Each 56F803/805/807 Quad Decoder module has the following registers:

- Decoder Control Register (DECCR)
- Filter Interval Register (FIR)
- Watchdog Timeout Register (WTR)
- Position Difference Counter (POSD) register
- Position Difference Hold (POSDH) register
- Revolution Counter (REV) register
- Revolution Hold (REVH) register
- Upper Position Counter (UPOS) register
- Lower Position Counter (LPOS) register
- Upper Position Hold (UPOSH) register
- Lower Position Hold (LPOSH) register
- Upper Initialization Register (UIR)
- Lower Initialization Register (LIR)
- Input Monitor Register (IMR)
- Test Register (TSTREG)

10.5 Functional Description

A timing diagram illustrating the basic operation of a quad incremental position Quad Decoder is illustrated in **Figure 10-1.**

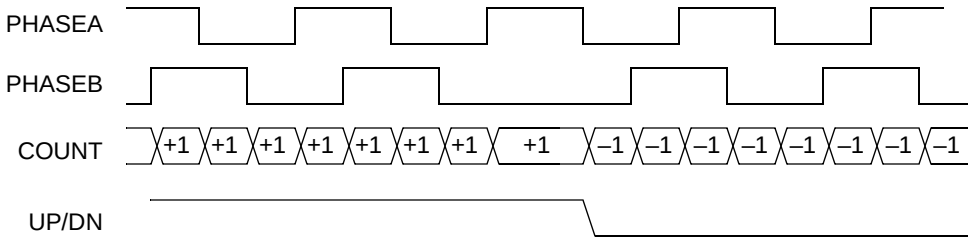


Figure 10-1. Quad Decoder Signals

10.5.1 Positive versus Negative Direction

A typical Quad Encoder has three outputs:

1. PHASEA signal
2. PHASEB signal
3. INDEX pulse (not shown)

If PHASEA leads PHASEB then motion is in the positive direction. If PHASEA lags PHASEB, the motion is in the negative direction. Transitions on these phases can be integrated to yield position, or differentiated to yield velocity. The 56F803/805/807's Quad Decoder is designed to perform these functions in hardware.

10.5.2 Block Diagram

A block diagram of the Quad Decoder module is shown in **Figure 10-2.**

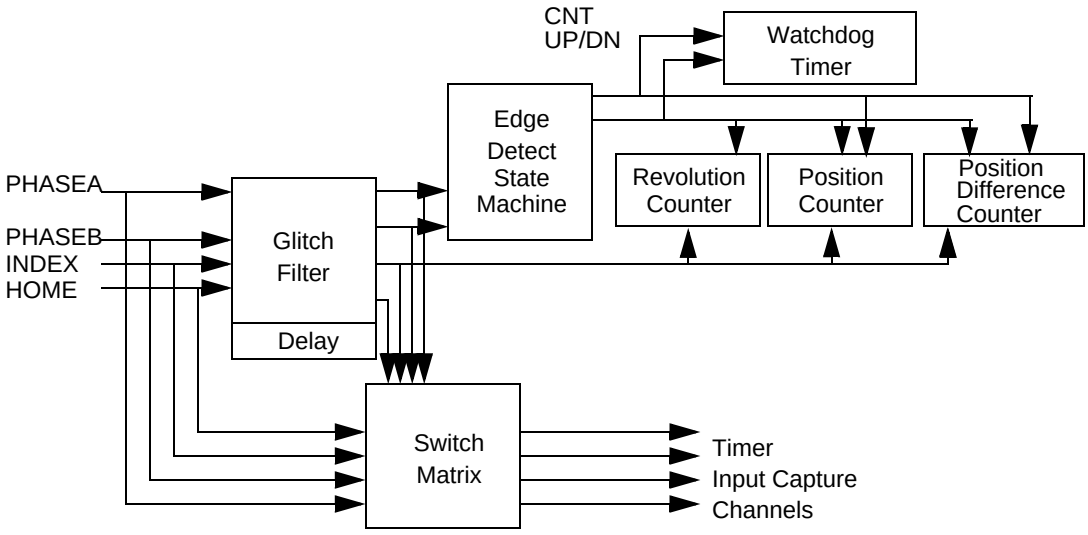


Figure 10-2. Quad Decoder Block Diagram

10.5.2.1 Glitch Filter

Because the logic of the Quad Decoder must sense transitions, the inputs are first run through a glitch filter. This filter has a digital delay line sampling four time points on the signal and verifying a majority of the samples are at a new state before outputting this new state to the internal logic. The sample rate of this delay line is programmable to adapt to a variety of signal bandwidths.

10.5.2.2 Edge Detect State Machine

The Edge Detect State Machine looks for changes in the four possible states of the filtered PHASEA and PHASEB inputs, calculating the direction of motion. This information is formatted as Count_Up and Count_Down signals. These signals are routed into up to three up/down counters:

1. Position Counter
2. Revolution Counter
3. Position Difference Counter

10.5.2.3 Position Counter

The 32-bit Position Counter calculates up or down on every count pulse, generated by the difference of PHASEA and PHASEB. This counter acts as an integrator, whose count value is proportional to position. The direction of the count is determined by the count-up and count-down signals. Position counters may be initialized to a predetermined value by one of three different methods:

1. Software-triggered event
2. INDEX signal transition
3. HOME signal transition

The INDEX and HOME signals can be programmed to interrupt the processor. Whenever the Position Counter is read, either UPOS or LPOS, a snapshot of the Position Counter, the Position Difference Counter, and the Revolution Counters are each placed into their respective Hold registers. The direction of the count is determined by Count_Up and Count_Down signals.

10.5.2.4 Position Difference Counter

The 16-bit Position Difference Counter contains the position difference value occurring between each read of the Position register. This register counts up or down on every count pulse. The counter acts as a differentiator whose count value is proportional to the change in position since the last time the Position Counter was read. When the Position register is read, the position difference of the counter's contents are copied into the Position Difference Hold register (POSDH) and Position Difference Counter is cleared.

10.5.2.5 Position Difference Counter Hold

This register stores a copy of the Position Difference Counter at the time the Position register was read. When the Position register is read, the position difference of the counter's contents are copied into the Position Difference Hold (POSDH) register and Position Difference Counter is cleared.

10.5.2.6 Revolution Counter

The 16-bit up/down Revolution Counter is intended to count, or integrate revolutions. This is achieved by counting index pulses. The direction of the count is determined by the Count_Up and Count_Down signals, determined by Phase A & B inputs. If the direction of the count is different

on the rising and falling edges of the index pulse, it indicates the Quad Encoder changed direction on the index pulse.

10.5.2.7 Pulse Accumulator Functionality

The logic can be programmed to integrate only selected transitions of the PHASEA signal. In this way the Position Counter is used as a Pulse Accumulator. The count direction is up. The Pulse Accumulator can also optionally be initialized by the INDEX input.

10.5.2.8 Watchdog Timer

A Watchdog Timer is included. It ensures the algorithm is indicating motion in the shaft. Two successive counts indicate proper operation and will reset the timer. Timeout value is programmable. When a timeout occurs, an interrupt to the processor can be generated.

10.5.3 Prescaler for Slow or Fast Speed Measurement

For a fast moving shaft encoder, the speed can be computed by calculating the change in the Position Counter per unit time, or by reading the Position Difference Counter (POSD) register to calculate speed. For applications with slow motor speeds and low line count Quad Encoders, high resolution velocity measurements can be made with the Timer module by measuring the time period between quad phases. The Timer module uses a 16-bit free running counter operate from a prescaled version of the IPBus clock. The prescaler divides the IPBus clock by values ranging from one to 64. A 40MHz IPBus clock frequency would yield a resolution of from 25ns to 1.6µs, and a maximum count period of from 1.62ms to 102ms. For example, with a 1000 tooth decoder, speeds could be calculated down to 0.15 RPM using a prescaler.

10.5.4 Modes

Table 10-1. Switch Matrix for Inputs to the Timer

	PHASEA	PHASEA filtered	PHASEB	PHASEB filtered	INDEX	INDEX filtered	HOME	HOME filtered
Mode0	Timer0	—	Timer1	—	Timer2	—	Timer3	—
Mode1	—	Timer0	—	Timer1	—	Timer2	—	Timer3
Mode2	—	Timer0,1	—	Timer2,3	—	—	—	—
Mode3	Reserved							

The PHASEA and PHASEB inputs are routed through a switch matrix to a general- purpose Timer module. The possible connections of the switch matrix are shown in **Table 10-1.** In Mode Zero, the Timer module can use all four available inputs as normal timer input capture channels. This does not preclude use of the Quad Decoder, but normally it would not be used in this mode. Modes One and Zero are similar, but Mode One takes advantage of the digital filter incorporated

in the Quad Decoder. Mode Two is the mode most likely to be used in conjunction with operation of the Quad Decoder. Both the positive and negative edges of PHASEA and PHASEB can be captured in Mode Two. The full speed range measurement is able to be reached in this mode.

10.6 Holding Registers and Initializing Registers

Hold registers are associated with three counters:

1. Position
2. Position difference
3. Revolution

When counter registers are read, a contents copy of the Counter register is written to the corresponding Hold register. Taking a snapshot of the counters' values provides a consistent view of a system position and a velocity to be attained.

The Counter and the Hold registers are read/write capable. However, *beware of writing values into any Hold register* and then reading any counter. If taking this action it will result in the Hold register contents being overwritten with the values in the counters.

The Position Counter is 32 bits wide. Assuring it can be reliably initialized with two 16-bit accesses, two registers, an upper and a lower initialization register, are provided. The Upper Initialization Register (UIR) and Lower Initialization Register (LIR) should be modified with the desired value. Next, the Position Counter can be loaded by writing 1 to the SWIP bit in the Decoder Control Register (DECCR). Alternatively, either the XIP or the HIP bits in the DECCR can enable the Position Counter to be initialized in response to a HOME or INDEX signal transition.

10.7 Register Definitions

Table 10-2. DEC Memory Map

Device	Peripheral	Address
803/805	DEC0_BASE	$0E40
	DEC1_BASE	$0E50
807	DEC0_BASE	$1240
	DEC1_BASE	$1250

The address of a register is the sum of a base address and an address offset. The base address is defined at the MCU level, while the address offset is defined at the module level Please refer to **Table 3-11.** All memory locations base and offsets are given in hex.

Each of the following set of registers are used for each Quad Decoder module. For example, if there are two Quad Decoders on the chip, there are two decoder control registers: DEC0_DECCR at data memory location DEC0_BASE+$0 and DEC1_DECCR at data memory location DEC1_BASE+$0.

Table 10-3. DEC Register Summary

Address Offset	Register Acronym	Register Name	Access Type	Register Location
Base + $0	DECCR	Control Register	Read/Write	Section 10.7.1
Base + $1	FIR	Filter Interval Register	Read/Write	Section 10.7.2
Base + $2	WTR	Watchdog/Timeout Register	Read/Write	Section 10.7.3
Base + $3	POSD	Position Difference Count Register	Read/Write	Section 10.7.4
Base + $4	POSDH	Position Difference Hold Register	Read/Write	Section 10.7.5
Base + $5	REV	Revolution Counter Register	Read/Write	Section 10.7.6
Base + $6	REVH	Revolution Hold Register	Read/Write	Section 10.7.7
Base + $7	UPOS	Upper Position Counter Register	Read/Write	Section 10.7.8
Base + $8	LPOS	Lower Position Counter Register	Read/Write	Section 10.7.9
Base + $9	UPOSH	Upper Position Hold Register	Read/Write	Section 10.7.10
Base + $A	LPOSH	Lower Position Hold Register	Read/Write	Section 10.7.11
Base + $B	UIR	Upper Initialization Register	Read/Write	Section 10.7.12
Base + $C	LIR	Lower Initialization Register	Read/Write	Section 10.7.13
Base + $D	IMR	Input Monitor Register	*Read Only*	Section 10.7.14

Bit fields of the 14 registers are summarized in **Figure 10-3.** Details of each follow.

Add. Offset	Register Name		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$0	DECCR	R	HIRQ	HIE	HIP	HNE	SWIP	REV	PH1	XIRQ	XIE	XIP	XNE	DIRQ	DIE	WDE	MODE[1:0]	
		W																
$1	FIR	R	0	0	0	0	0	0	0	0	DELAY[7:0]							
		W																
$2	WTR	R	CNT[15:0]															
		W																
$3	POSD	R	POSD[15:0]															
		W																
$4	POSDH	R	POSDH[15:0]															
		W																
$5	REV	R	REV[15:0]															
		W																
$6	REVH	R	REV[15:0]															
		W																
$7	UPOS	R	POS[31:16]															
		W																
$8	LPOS	R	POS[15:0]															
		W																
$9	UPOSH	R	POS[31:16]															
		W																
$A	LPOSH	R	POS[15:0]															
		W																
$B	UIR	R	INITIALIZATION VALUE[15:0]															
		W																
$C	LIR	R	INITIALIZATION VALUE[15:0]															
		W																
$D	IMR	R	0	0	0	0	0	0	0	0	FPHA	FPHB	FIND	FHOM	PHA	PHB	INDEX	HOME
		W																

R — Read as 0
W — Reserved

Figure 10-3. DEC Register Map Summary

10.7.1 Decoder Control Register (DECCR)

DEC_BASE+$0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	HIRQ	HIE	HIP	HNE	SWIP	REV	PH1	XIRQ	XIE	XIP	XNE	DIRQ	DIE	WDE	MODE	
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-4. Decoder Control Register (DECCR)

See Table A-8, List of Programmer's Sheets

10.7.1.1 HOME Signal Transition Interrupt Request (HIRQ)—Bit 15

The HIRQ bit is set when a transition on the HOME signal occurs according to the HNE bit. When the HIRQ and HIE bits are set, a HOME interrupt occurs. The HIRQ bit will remain set until it is cleared by software. To clear this bit, write 1 to it.

- 0 = No interrupt
- 1 = HOME signal transition interrupt request

10.7.1.2 HOME Interrupt Enable (HIE)—Bit 14

This read/write bit enables HOME signal interrupts.

- 0 = Disable HOME interrupts
- 1 = Enable HOME interrupts

10.7.1.3 Enable HOME to Initialize Position Counters UPOS and LPOS (HIP)—Bit 13

This read/write bit allows the position counter to be initialized by the HOME signal.

- 0 = No action
- 1 = Enable HOME signal

10.7.1.4 Use Negative Edge of HOME Input (HNE)—Bit 12

This read/write bit determines whether to use the positive or negative edge of the HOME input.

- 0 = Use positive going edge-to-trigger initialization of position counters UPOS and LPOS
- 1 = Use negative going edge-to-trigger initialization of position counters UPOS and LPOS

10.7.1.5 Software Triggered Initialization of Position Counters UPOS and LPOS (SWIP)—Bit 11

Writing a 1 to this bit will transfer the UIR and LIR contents to ENPOS. This bit is always read as 0.

- 0 = No action
- 1 = Initialize position counter

10.7.1.6 Enable Reverse Direction Counting (REV)—Bit 10

This read/write bit reverses the interpretation of the quad signal, changing the direction of count.

- 0 = Count normally

- 1 = Count in the reverse direction

10.7.1.7 Enable Signal Phase Count Mode (PH1)—Bit 9

This read/write bit bypasses the Quad Decoder logic.

- 0 = Use standard Quad Decoder where PHASEA and PHASEB represent a two phase quad signal
- 1 = Bypass the Quad Decoder. A positive transition of the PHASEA input generates a count signal. The PHASEB input and the REV bit control the counter direction
  - IF REV = 0, PHASEB = 0, then count up
  - IF REV = 0, PHASEB = 1, then count down
  - IF REV = 1, PHASEB = 0, then count down
  - IF REV = 1, PHASEB = 1, then count up

10.7.1.8 Index Pulse Interrupt Request (XIRQ)—Bit 8

This bit is set when an index interrupt occurs. It will remain set until cleared by software. The clearing procedure is to write 1 to this bit.

- 0 = No interrupt has occurred
- 1 = Index pulse interrupt has occurred

10.7.1.9 Index Pulse Interrupt Enable (XIE)—Bit 7

This read/write bit enables index interrupts.

- 0= Index pulse interrupt is disabled
- 1= Index pulse interrupt is enabled

10.7.1.10 Index Triggered Initialization of Position Counters UPOS and LPOS (XIP)—Bit 6

This read/write bit enables the position counter to be initialized by the INDEX pulse.

- 0 = No action
- 1 = INDEX pulse initializes the position counter

10.7.1.11 Use Negative Edge of Index Pulse (XNE)—Bit 5

This read/write bit determines the edge of the Index Pulse used to initialize the position counter.

- 0 = Use positive transition edge of INDEX pulse

- 1 = Use negative transition edge of INDEX pulse

10.7.1.12 Watchdog Timeout Interrupt Request (DIRQ)—Bit 4

This bit is set when a Watchdog Timeout Interrupt occurs. It will remain set until cleared by software. Write 1 to this bit to clear it.

- 0 = No interrupt has occurred
- 1 = Watchdog timeout interrupt has occurred

10.7.1.13 Watchdog Timeout Interrupt Enable (DIE)—Bit 3

This read/write bit enables watch timeout interrupts.

- 0 = Watchdog Timer Interrupt is disabled
- 1 = Watchdog Timer Interrupt is enabled

10.7.1.14 Watchdog Enable (WDE)—Bit 2

This bit allows operation of the Watchdog Timer monitoring the PHASEA and PHASEB inputs for motor movement.

- 0 = Watchdog Timer is disabled
- 1 = Watchdog Timer is enabled

10.7.1.15 Switch Matrix Mode (MODE[1:0])—Bits 1–0

These read/write bits selects the Switch Matrix mode connecting inputs to the Timer module. The modes are illustrated in **Table 10-1.**

- 00 = Mode0: Input captures connected to the four input pins
- 01 = Mode1: Input captures connected to the filtered versions of the four input pins
- 10 = Mode2: PHASEA input connected to both channels zero and one of the timer to allow capture of both rising and falling edges; PHASEB input connected to both channels 2 and 3 of the timer
- 11 = Mode3: Reserved

10.7.2 Filter Interval Register (FIR)

This register sets the sample rate of the Digital Glitch Filter. A counter increases or decreases to the value in the FIR. When the count reaches the specified value, the counter is reset and the filter takes a new sample of the raw PHASEA, PHASEB, INDEX, and HOME input signals. If the filter interval value is zero, the digital filter for the PHASEA, PHASEB, INDEX, and HOME inputs is bypassed.

Bypassing the Digital Filter enables the position/position difference counters to operate with count rates up to the IPBus frequency (40MHz).

DEC_BASE+$1	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	0	0	0	DELAY							
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-5. Filter Delay Register (FIR)

See Table A-8, List of Programmer's Sheets

The value of DELAY[7:0] +1 represents the filter interval period in increments of the IPBus clock period, 25ns for a 40MHz IPBus clock. Therefore, a value of one represents a filter interval of 50ns.

The Filter Interval Register (FIR) works as follows: Drive the Quad Decoder inputs, PHASEA, PHASEB, INDEX, and HOME monitoring the output in the Input Monitor Register (IMR). Determine how many IPBus clock cycles it takes before the output shows up, by using the following equations, where *f* is the number loaded in the FIR and *s* is the number of samples needed. s = 4 for this example.

(1) DELAY (IPBus clock cycles) = (f +1) × s +1 (to read the filtered output)

(2) DELAY (IPBus clock cycles) = (f +1) × s +2 (to monitor the output in the IMR)

One more additional IPBus clock cycle is needed to read the filtered output and two more IPBus clock cycles to monitor the filtered output in the IMR. A one is added to *f* to account for the interval timer in the filter starting its counting from zero. The sample rate is set when it reaches the number *f*.

For example: 1, when *f* = 0, then the filter is bypassed and *s* is zero because there is no sampling. Therefore, DELAY = 1 or 2 clock cycles according to the equations above.

For example: 2, when *f* = 5, then DELAY = (5+1) × 4 + (1 or 2) = 25 or 26 clock cycles.

10.7.3 Watchdog Timeout Register (WTR)

This register stores the timeout count for the Quad Decoder module Watchdog Timer. This timer is separate from the Watchdog Timer in the clock module**.**

DEC_BASE+$2	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	CNT															
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-6. Watchdog Timeout Register (WTR)

See Table A-8, List of Programmer's Sheets

CNT[15:0] is a binary representation of the number of clock cycles plus one the Watchdog Timer will count before timing out and optionally generating an interrupt. This is a read/write register.

10.7.4 Position Difference Counter Register (POSD)

This register contains the position change in value occurring between each read of the Position register. The value of the Position Difference Counter (POSD) register can be used to calculate velocity. This is a read/write register.

The 16-bit Position Difference Counter computes up or down on every count pulse. This counter acts as a differentiator whose count value is proportional to the change in position since the last time the position counter was read. When the Position register is read, the Position Difference Counter's contents are copied into the Position Difference Hold (POSDH) register and Position Difference Counter is cleared.

DEC_BASE+$3	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	POSD															
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-7. Position Difference Counter Register (POSD)

See Table A-8, List of Programmer's Sheets

10.7.5 Position Difference Hold Register (POSDH)

This read/write register contains a snapshot of the change in position value occurring between each read of the Position register by storing a copy of the Position Difference Counter at the time the Position register is read. When the Position register is read, the Position Difference Counter's contents are copied into the Position Difference Hold (POSDH) register and Position Difference

Counter is cleared. The value of the Position Difference Hold (POSDH) register can be used to calculate velocity.

DEC_BASE+$4	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	POSDH															
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-8. Position Difference Hold Register (POSDH)

See Table A-8, List of Programmer's Sheets

10.7.6 Revolution Counter Register (REV)

This read/write register contains the current value of the Revolution Counter.

DEC_BASE+$5	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	REV															
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-9. Revolution Counter Register (REV)

See Table A-8, List of Programmer's Sheets

10.7.7 Revolution Hold Register (REVH)

This read/write register contains a snapshot of the value of the Revolution Counter.

DEC_BASE+$6	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	REV															
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-10. Revolution Hold Register (REVH)

See Table A-8, List of Programmer's Sheets

10.7.8 Upper Position Counter Register (UPOS)

This read/write register contains the upper and most significant half of the Position Counter. This is the binary count from the Position Counter.

DEC_BASE+$7	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	POS[31:16]															
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-11. Upper Position Counter Register (UPOS)

See Table A-8, List of Programmer's Sheets

10.7.9 Lower Position Counter Register (LPOS)

This read/write register contains the lower and least significant half of the current value of the Position Counter. It is the binary count from the Position Counter.

DEC_BASE+$8	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	POS[15:0]															
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-12. Lower Position Counter Register (LPOS)

See Table A-8, List of Programmer's Sheets

10.7.10 Upper Position Hold Register (UPOSH)

This read/write register contains a snapshot of the upper and most significant half of the Position Counter. It is the binary count from the Position Counter.

DEC_BASE+$9	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	POS[31:16]															
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-13. Upper Position Counter Register (UPOSH)

See Table A-8, List of Programmer's Sheets

10.7.11 Lower Position Hold Register (LPOSH)

This read/write register contains a snapshot of the lower and least significant half of the current value of the Position Counter. It is the binary count from the Position Counter.

DEC_BASE+$A	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	POS[15:0]															
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-14. Lower Position Hold Register (LPOSH)

See Table A-8, List of Programmer's Sheets

10.7.12 Upper Initialization Register (UIR)

This read/write register contains the value to be used to initialize the Upper Half of the Position Counter (UPOS).

DEC_BASE+$B	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	INITIALIZATION VALUE															
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-15. Upper Initialization Register (UIR)

See Table A-8, List of Programmer's Sheets

10.7.13 Lower Initialization Register (LIR)

This read/write register contains the value to be used to initialize the Lower Half of the Position Counter (LPOS).

DEC_BASE+$C	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	INITIALIZATION VALUE															
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-16. Lower Initialization Register (LIR)

See Table A-8, List of Programmer's Sheets

10.7.14 Input Monitor Register (IMR)

This *read-only* register contains the values of the raw and filtered PHASEA, PHASEB, INDEX, and HOME input signals. The reset value depends on the values of the raw and filtered values of the PHASEA, PHASEB, INDEX and HOME. If these input pins are connected to a pull-up, bits 0–7 of the IMR will all be ones. However, if these input pins are connected to a pull-down device, bits 0–7 will all be zeros. If no pull-up or pull-down is connected to these input pins, the reset value of the eight lower bits of the IMR will all be unknown.

DEC_BASE+$D	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	0	0	0	FPHA	FPHB	FIND	FHOM	PHA	PHB	INDEX	HOME
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-17. Input Monitor Register (IMR)

See Table A-8, List of Programmer's Sheets

10.7.14.1 Reserved Bits—15–8

This bit field is reserved or not implemented. It is read as 0 and cannot be modified by writing.

10.7.14.2 FPHA—Bit 7

This is the filtered version of PHASEA input.

10.7.14.3 FPHB—Bit 6

This is the filtered version of PHASEB input.

10.7.14.4 FIND—Bit 5

This is the filtered version of INDEX input.

10.7.14.5 FHOM—Bit 4

This is the filtered version of HOME input.

10.7.14.6 PHA—Bit 3

This is the raw PHASEA input.

10.7.14.7 PHB—Bit 2

This is the raw PHASEB input.

10.7.14.8 INDEX—Bit 1

This is the raw INDEX input.

10.7.14.9 HOME—Bit 0

This is the raw HOME input.

Table 10-4. Document Revision History for Chapter 10

Version History	Description of Change
Rev. 8	Formatting, layout, spelling, and grammar corrections. Added revision history table. Corrected the field names in the UPOS and UPOSH registers in **Figure 10-3**.

Chapter 11
Pulse Width Modulator Module (PWM)

11.1 Introduction

This chapter describes the Pulse Width Modulator (PWM) module. The PWM can be configured as three complementary pairs, six independent PWM signals, or their combinations, such as one complementary or four independent. Both Edge- and Center-Aligned synchronous pulse width control, from zero to 100 percent modulation, are supported.

A 15-bit common PWM counter is applied to all six channels. PWM resolution is one clock period for Edge-Aligned operation and two clock periods for Center-Aligned operation. The clock period is dependent on the IPBus frequency and a programmable prescaler.

When generating complementary PWM signals, the module features automatic deadtime insertion to PWM output pairs. Each PWM output can be controlled by PWM generator or software manually.

11.2 Block Diagram

The PWM block diagram is illustrated in **Figure 11-1.**

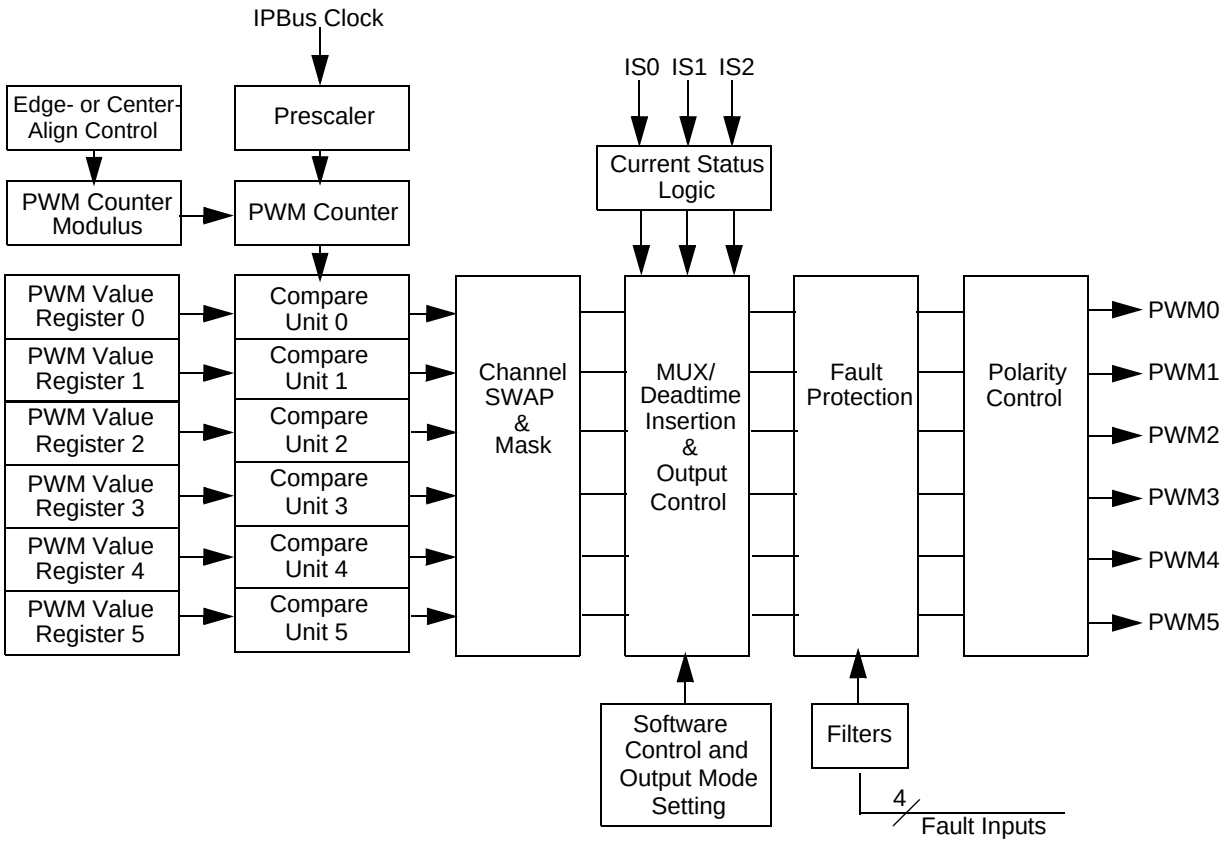


Figure 11-1. PWM Block Diagram

11.3 Features

- Six PWM signals
  - — All independent
  - — Complementary pairs
  - — Mix independent and complementary
- Features of complementary channel operation
  - — Deadtime insertion
  - — Separate top and bottom pulse width correction via current status inputs or software
  - — Separate top and bottom polarity control
- Edge- or Center-Aligned PWM signals
- 15-bits of resolution

- Half-cycle reload capability
- Integral reload rates from 1 to 16
- Individual software controlled PWM output
- Programmable fault protection
- Polarity control
- 10/12mA current source/sink capability on PWM pins
- Write protected registers

11.4 Functional Description

11.4.1 Prescaler

To permit lower PWM frequencies, the prescaler produces the PWM clock frequency by dividing the IPBus clock frequency by one, two, four, or eight. The prescaler bits, PRSC0 and PRSC1 in the PWM Control (PMCTL) register, select the prescaler divisor. This prescaler is buffered and will not be used by the PWM generator until the LDOK bit is set and a new PWM reload cycle begins.

11.4.2 PWM Generator

The PWM generator contains a 15-bit up/down PWM counter producing output signals with software-selectables.

11.4.2.1 Alignment

The Edge-Align (EDGE) bit in the PWM Configure (PMCFG) register selects either Center-Aligned or Edge-Aligned PWM generator outputs.

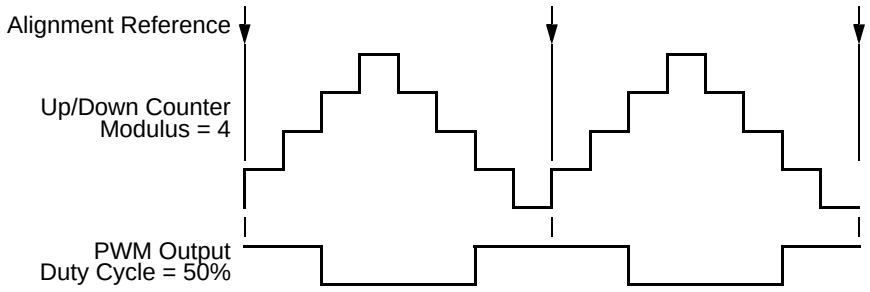


Figure 11-2. Center-Aligned PWM Output

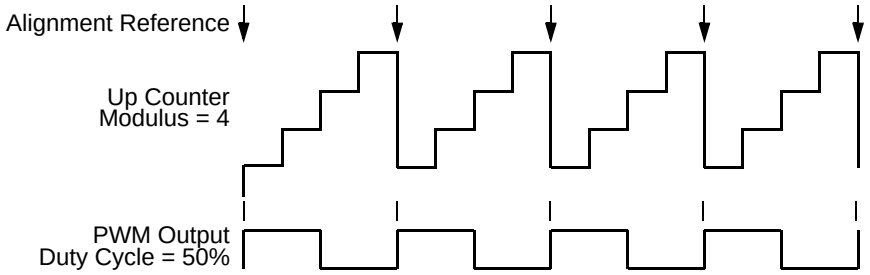


Figure 11-3. Edge-Aligned PWM Output

Note: Because of the equals-comparator architecture of this PWM, the modulus equals zero case is considered illegal. However, the deadtime constraints and fault conditions will still be guaranteed.

11.4.2.2 Period

The PWM period is determined by the value written to the PWMCM register. The PWM counter is an up/down counter in a Center-Aligned operation. In this mode the PWM highest output resolution is two IPBus clock cycles. The modulus is one-half of the PWM output period in PWM clock cycles.

$$\text{PWM period} = (\text{PWM modulus}) \times (\text{PWM clock period}) \times 2$$

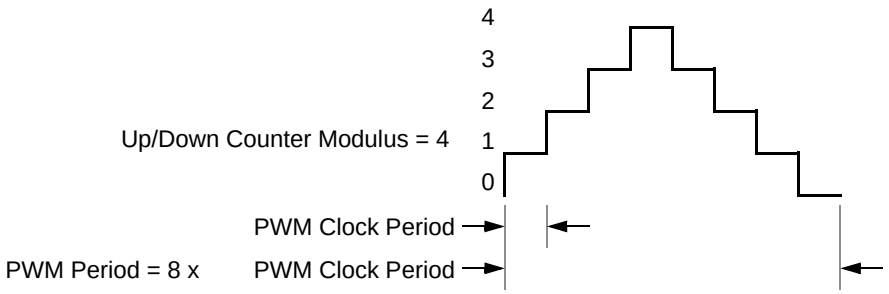


Figure 11-4. Center-Aligned PWM Period

The PWM counter is an up-counter during an Edge-Aligned operation. In this mode, the PWM highest output resolution is one IPBus clock cycle. The modulus is the period of the PWM output in PWM clock cycles.

$$\text{PWM period} = \text{PWM modulus} \times \text{PWM clock period}$$

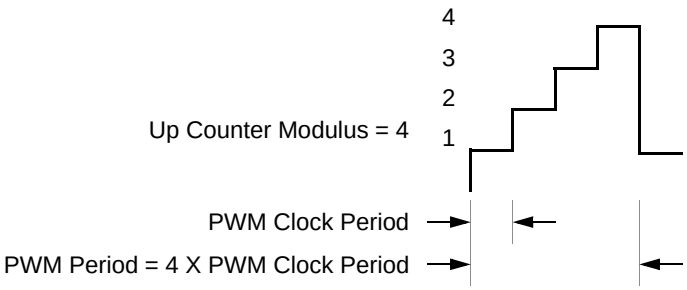


Figure 11-5. Edge-Aligned PWM Period

11.4.2.3 Pulse Width Duty Cycle

The signed 16-bit number written to the PWM value registers is the pulse width in PWM clock periods of the PWM prescaler output.

$$\text{Duty Cycle} = \frac{\text{PWM value}}{\text{Modulus}} \times 100$$

Note: A PWM value less than, or equal to zero, deactivates the PWM output for the entire PWM period. A PWM value greater than or equal to the modulus activates the PWM output for the entire PWM period.

Table 11-1. PWM Value and Underflow Conditions

PWMVALx	Condition	PWM Value Used
$0000–$7FFF	Normal	Value in Registers
$8000–$FFFF	Underflow	$0000

A Center-Aligned operation is illustrated in **Figure 11-6.** The pulse width is twice the value written to the PWM Value register with Center-Aligned output in PWM clock cycles.

$$\text{Pulse width} = (\text{PWM value}) \times (\text{PWM clock period}) \times 2$$

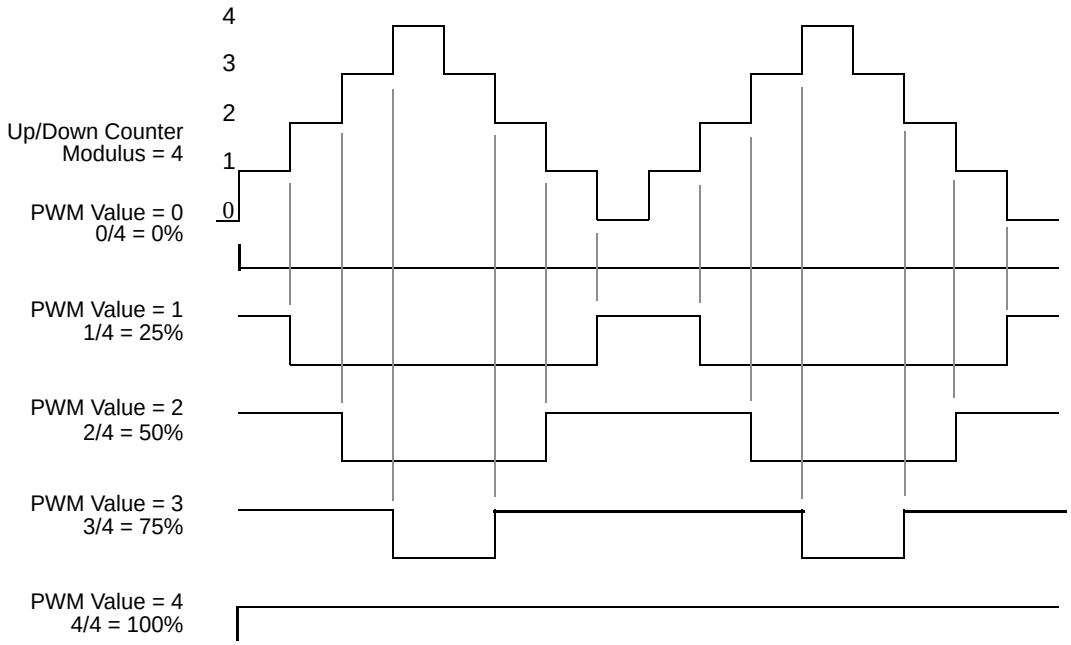


Figure 11-6. Center-Aligned PWM Pulse Width

An Edge-Aligned operation is illustrated in **Figure 11-7.** The pule width is the value written to the PWM Value register with Edge-Aligned output in PWM clock cycles.

$$\text{Pulse width} = (\text{PWM value}) \times (\text{PWM clock period})$$

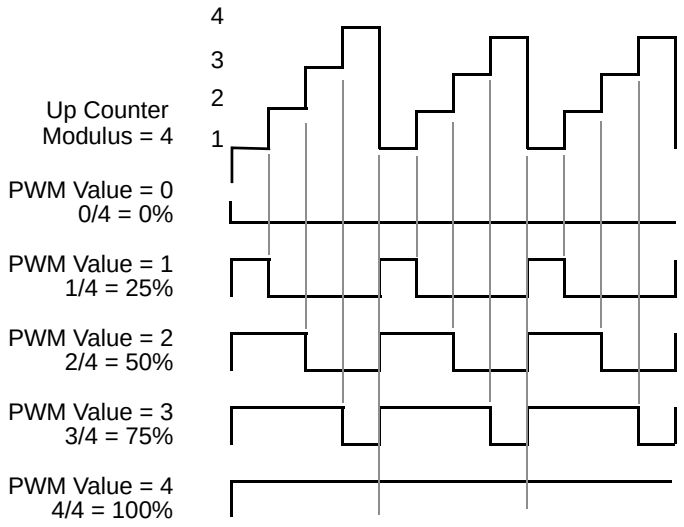


Figure 11-7. Edge-Aligned PWM Pulse Width

11.4.3 Independent or Complementary Channel Operation

In the PWM Configure register, writing Logic 1 of the Independent or complement pair operation (INDEPxx) bit configures a pair of the PWM outputs as two independent PWM channels. Each PWM output has its own PWM value register operating independently of the other channels in independent channel operation.

Writing Logic 0 to the INDEPxx bit configures the PWM output as a pair of complementary channels. The PWM pins are paired in complementary channel operation, illustrated in **Figure 11-8.**

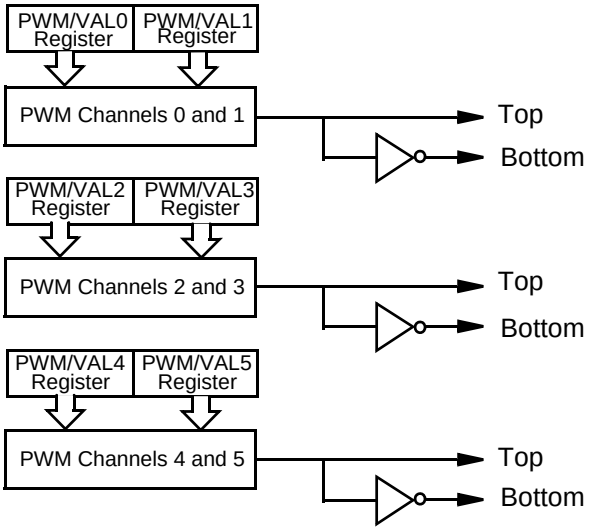


Figure 11-8. Complementary Channel Pairs

The complementary channel operation drives top and bottom transistors in an inverter circuit, such as the one in **Figure 11-9.**

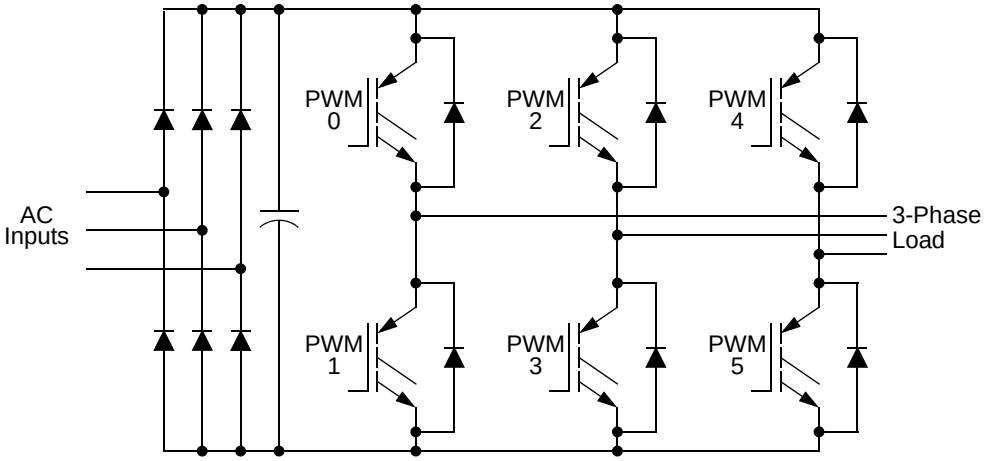


Figure 11-9. Typical 3-Phase Inverter

In complementary channel operation, there are three additional features:

1. Deadtime insertion
2. Separate top and bottom pulse width correction for distortions caused by deadtime inserted and reactive load characteristics
3. Separate top and bottom output polarity control

11.4.4 Deadtime Generators

While in the Complementary mode, each PWM pair can be used to drive top/bottom transistors, illustrated in **Figure 11-10.** Ideally, the PWM pairs are an inversion of each other. When the top PWM channel is active, the bottom PWM channel is inactive and vice versa.

To avoid short circuiting between top and bottom transistor, there must be no overlap of conducting intervals between top and bottom transistor. But the transistor's characteristics make its switching-off time longer than switching-on time. To avoid the conducting overlap of top and bottom transistors, deadtime needs to be inserted in the switching period.

Deadtime generators automatically insert software-selectable activation delays into each pair of PWM outputs. The Pulse Module Deadtime (PMDEADTM) register specifies the number of PWM clock cycles to use for deadtime delay. Every time the PWM generator output changes state, deadtime is inserted. Deadtime forces both PWM outputs in the pair to the inactive state. A method of correcting this inserted deadtime, adding to or subtracting from the PWM value used, is discussed subsequently.

Figure 11-11, Figure 11-12, and **Figure 11-13** illustrate deadtime insertion in different operation conditions.

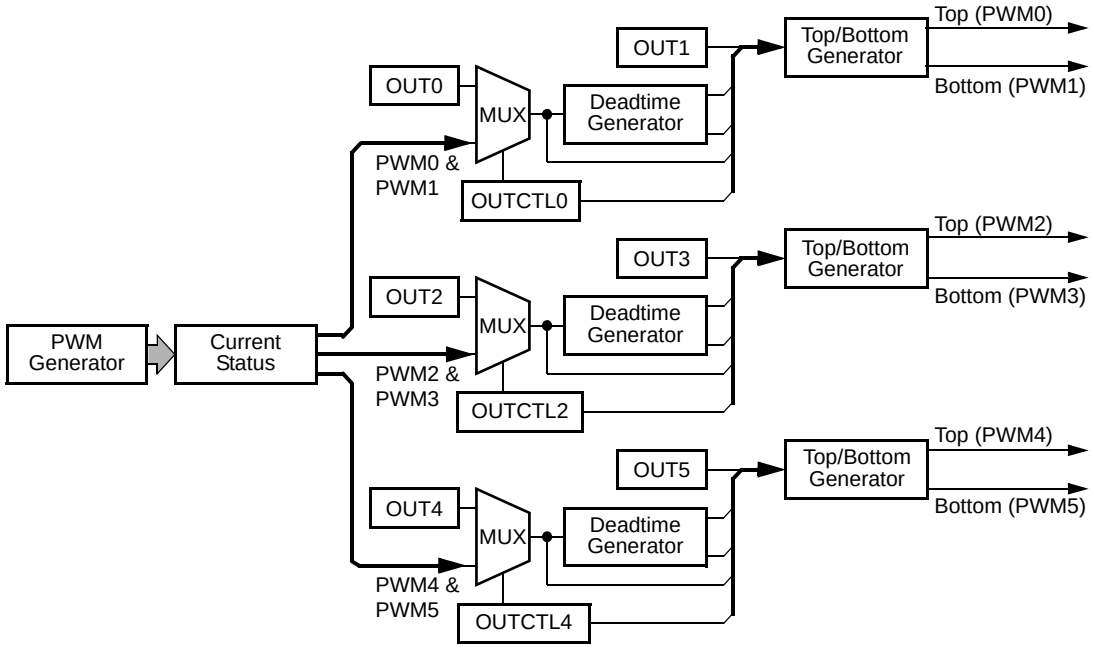


Figure 11-10. Deadtime Generators

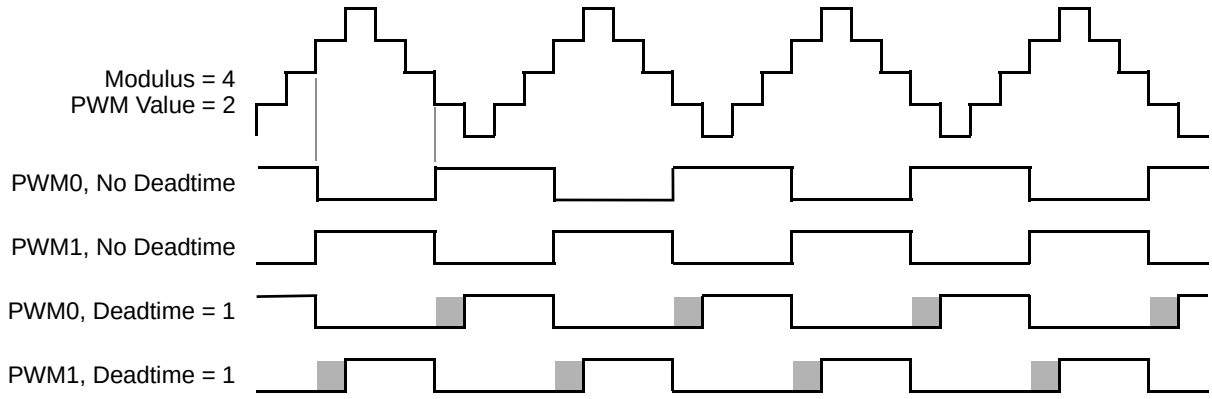


Figure 11-11. Deadtime Insertion, Center Alignment

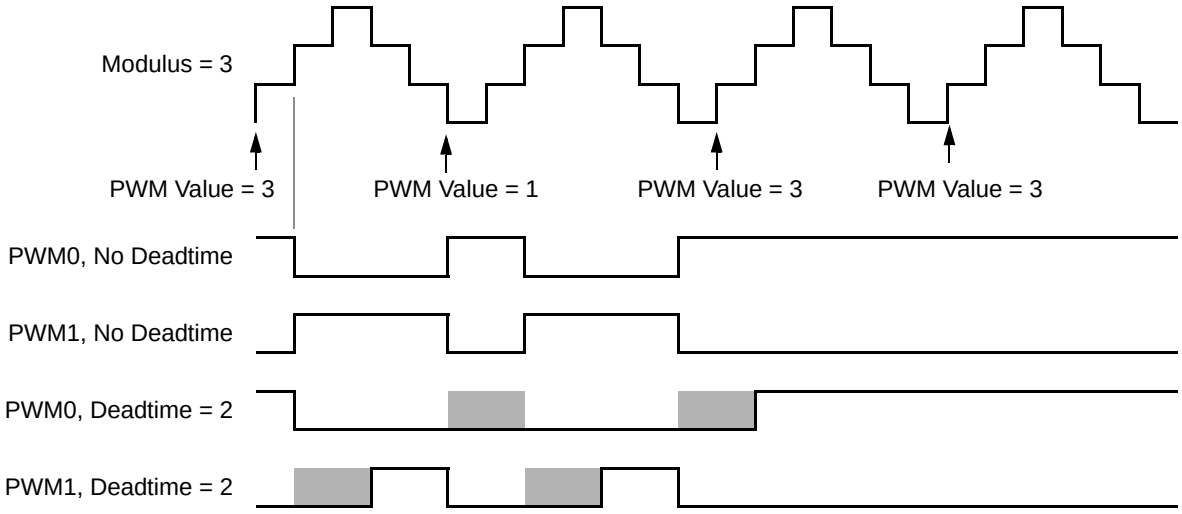


Figure 11-12. Deadtime at Duty Cycle Boundaries

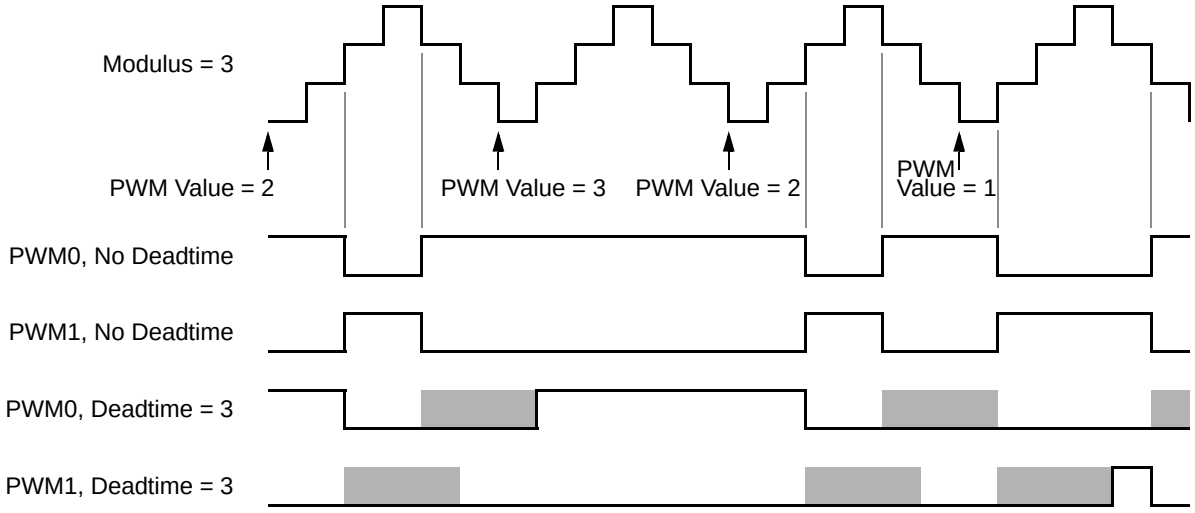


Figure 11-13. Deadtime and Small Pulse Widths

Note: The waveform at the output pin is delayed by two IPBus clock cycles for deadtime insertion.

11.4.4.1 Top/Bottom Deadtime Correction

In the Complementary mode, either the top or the bottom transistor controls the output voltage. However, deadtime has to be inserted to avoid overlap of conducting interval between the top and bottom transitions. Both transistors in Complementary mode are off during deadtime inserted,

allowing the output voltage to be determined by the current status (direction) of load and introduce distortion in the output voltage. Please refer to **Figure 11-14.** The distortion typically manifests itself as poor output waveforms with visible glitches and harmonics.

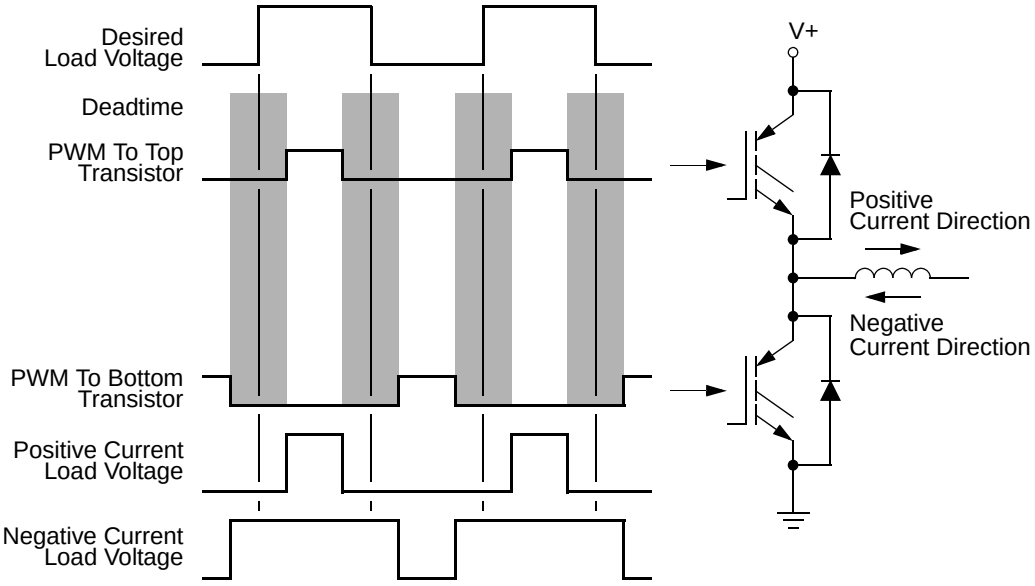


Figure 11-14. Deadtime Distortion

Load inductance distorts output voltage by keeping current flowing through the anti-body diode of transistor during deadtime. This deadtime current flow creates a output voltage varying with current direction. With a positive current flow, the output voltage during deadtime is equal to the bottom supply voltage, putting the top transistor in control. With a negative current flow, the output voltage during deadtime is equal to the top supply voltage, putting the bottom transistor in control. This results in the original pulse widths shortened by deadtime insertion, the averaged output will be less than desired value. However, when deadtime is inserted, it creates a distortion in load current waveform. This distortion is aggravated by dissimilar turn-on and turn-off delays of each of the transistors. By giving the PWM module information regarding which transistor is controlling at a given time, distortion can be corrected.

For a typical circuit in complementary channel operation, only one of the transistors will be effective in controlling the output voltage at any given time. This depends on the direction of the load current for that pair. Please see **Figure 11-14.** To correct distortion one of two different factors must be added to the desired PWM value, depending on whether the top or bottom transistor is controlling the output voltage. Therefore, the software is responsible for calculating both compensated PWM values prior to placing them in an odd/even numbered PWM register pair. Either the odd or the even PWM Value (PWMVALx) registers control the pulse width at any given time. For a given PWM pair, whether the odd or even PWMVAL register is active depends on either:

- The state of the current status pin (ISx) for that driver
- The state of the odd/even correction bit (IPOLx) for that driver

To correct deadtime distortion, software can decrease or increase the value in the appropriate PWMVAL register.

- In Edge-Aligned operation, decreasing or increasing the PWM value by a correction value equal to the deadtime typically compensates for deadtime distortion.
- In Center-Aligned operation, decreasing or increasing the PWM value by a correction value equal to one-half the deadtime typically compensates for deadtime distortion.

In the complementary channel operation, the ISENS[1:0] bits in the PMCTL register select one of three correction methods:

- Manual deadtime correction
- Automatic deadtime correction during deadtime sampling the current status pins (ISx)
- Automatic current status correction when the PWM counter value equals the value in the PWM counter modulus registers samples the current status pins (ISx)

Table 11-2. Correction Method Selection

ISENS[1:0]	Correction Method	Comments
0X	Manual correction or no correction	—
10	Current status sample correction on pins IS0, IS1, and IS2 during deadtime.	The polarity of the ISx pin is latched when both the top and bottom PWMs are off. At the 0% and 100% duty cycle boundaries, there is no deadtime, so no new current value is sensed.
11	Current status sample on pins IS0, IS1, and IS2. At the half cycle in Center-Aligned operation At the end of the cycle in Edge-Aligned operation	Current is sensed even with 0%, or 100% duty cycle.

Note: Assumes the user will provide current status sensing circuitry causing the voltage at the corresponding input pin to be low for positive current and high for negative current. Additionally, it assumes the top PWMs are PWM 0, 2, and 4 while the bottom PWMs are PWM 1, 3, and 5.

11.4.4.2 Manual Deadtime Correction

The IPOL0–IPOL2 bits in PWM Control (PMCTL) register select either the odd or the even PWM value registers to use in the next PWM cycle in Complementary mode when ISENS[1:0 0x.

Table 11-3. Top/Bottom Manual Correction

Bit	Logic State	Output Control
IPOL0	0	PWMVAL0 Controls PWM0/PWM1 Pair
	1	PWMVAL1 Controls PWM0/PWM1 Pair
IPOL1	0	PWMVAL2 Controls PWM2/PWM3 Pair
	1	PWMVAL3 Controls PWM2/PWM3 Pair
IPOL2	0	PWMVAL4 Controls PWM4/PWM5 Pair
	1	PWMVAL5 Controls PWM4/PWM5 Pair

Note: IPOLx bits are buffered allowing only one PWM register to be used per PWM cycle. If an IPOLx bit changes during a PWM period, the new value does not take effect until the next PWM period. IPOLx bits take effect at the end of each PWM cycle regardless of the state of the LOAD OKAY (LDOK) bit.

To detect the current status, the voltage on each ISx pin is sampled at the end of each deadtime. The value is stored in the DTx bits in the fault acknowledge register. The DTx bits are a timing marker especially indicating when to toggle between PWM value registers. Software can then set the IPOLx bit to toggle PWMVAL registers according to DTx values.

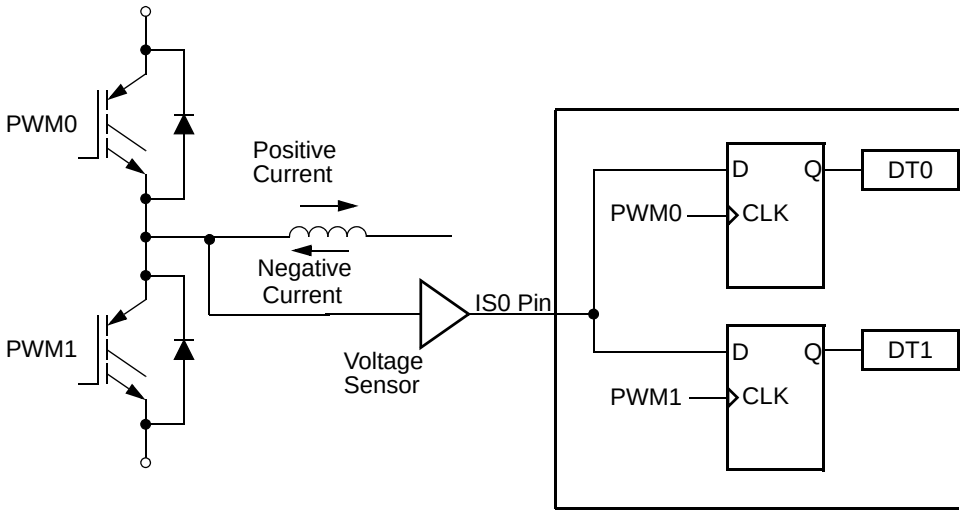


Figure 11-15. Current Status Sense Scheme for Deadtime Correction

Both D flip-flops latch *low* (DT0 = 0 and DT1 = 0) during deadtime periods if the current is large and flowing out of the complementary circuit. Please refer to **Figure 11-15.**

Both D flip-flops latch the *high* (DT0 = 1, DT1 = 1) during deadtime periods if current is large and flowing into the complementary circuit. Please see **Figure 11-15.**

However, under low-current, the output voltage of the complementary circuit during deadtime is somewhere between the high and low levels. The current cannot free-wheel throughout the opposition anti-body diode, regardless of polarity, giving additional distortion when the current crosses zero. Sampled results will be DT0 = 0 and DT1 = 1. Thus, the best time to change one PWM value register to another is just before the current zero crossing. Please refer to **Figure 11-16.**

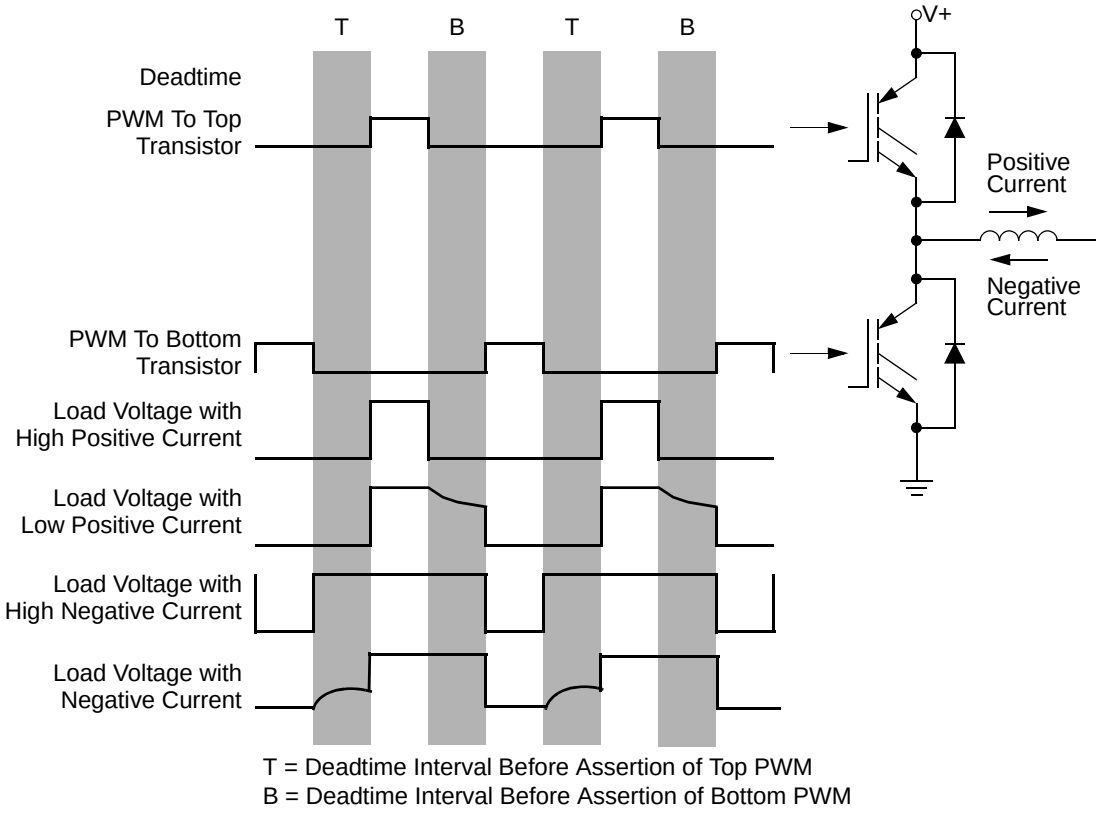


Figure 11-16. Output Voltage Waveforms

11.4.5 Automatic Deadtime Correction

A Current Status pin, ISx, for a PWM pair, selects either the odd or the even PWM value registers to use in the next PWM cycle. The selection is based on user-provided current sense circuitry driving the ISx pin high for negative current and low for positive current.

Table 11-4. Top/Bottom Automatic Correction

Pin	Logic State	Output Control
$\overline{IS0}$	0	PWMVAL0 Controls PWM0/PWM1 Pair
	1	PWMVAL1 Controls PWM0/PWM1 Pair
$\overline{IS1}$	0	PWMVAL2 Controls PWM2/PWM3 Pair
	1	PWMVAL3 Controls PWM2/PWM3 Pair
$\overline{IS2}$	0	PWMVAL4 Controls PWM4/PWM5 Pair
	1	PWMVAL5 Controls PWM4/PWM5 Pair

The first automatic deadtime correction mode, where ISENS[1:0] = 10, is accomplished by sampling the current status pin (ISx) during deadtime. No samples can be taken at the 100 percent and zero percent duty cycle boundaries because deadtime does not exist.

The second automatic deadtime correction mode, where ISENS[1:0] = 11, is accomplished by sampling the current status pin (ISx) at the half cycle during Center-Aligned operation and at the end of the cycle during Edge-Aligned operation.

Note: Values latched on the ISx pins are buffered so only one PWM register is used per PWM cycle. If a current status changes during a PWM period, the new value does not take effect until the next PWM period.

When initially enabled by setting the PWMEN bit, no current status has previously been sampled. Even PWM value registers initially control the three PWM pairs when configured for current status correction.

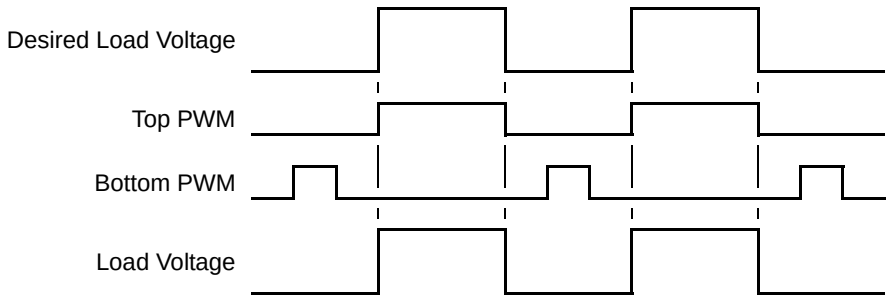


Figure 11-17. Correction with Positive Current

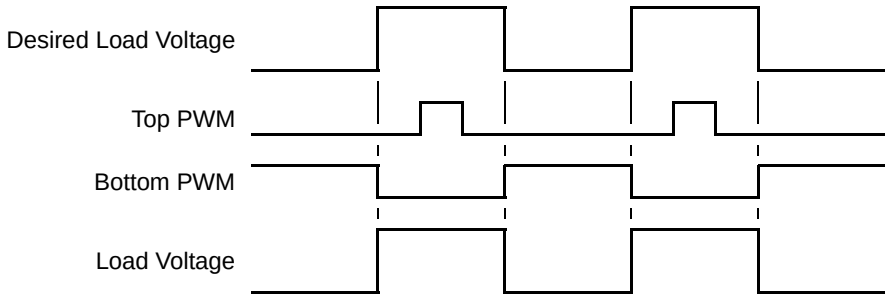


Figure 11-18. Correction with Negative Current

11.4.6 Output Polarity

Positive polarity means when the PWM is active its output is high. Conversely, *negative* polarity means when the PWM is active its output is low.

Output polarity of the PWMs is determined by two options:

1. TOPNEG controls the polarity of PWM0, PWM2 and PWM4 outputs, which typically drive the top transistors of the pair. When TOPNEG is set these outputs are active-low.
2. BOTNEG controls the polarity of PWM1, PWM3 and PWM5 outputs, which typically drive the bottom transistors of the pair. When BOTNEG is set these outputs are active-low.

Both TOPNEG and BOTNEG bits are in the Configure register (PMCFG). Please see **Figure 11-19.**

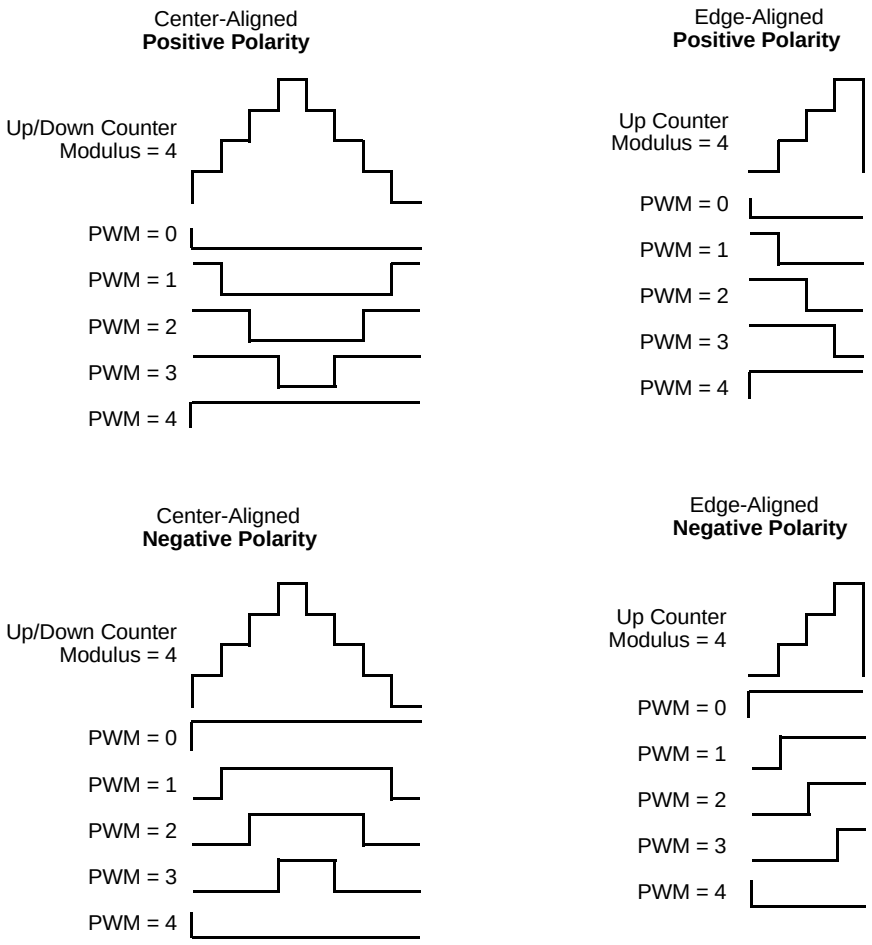


Figure 11-19. PWM Polarity

11.5 Software Output Control

Setting Output Control Enable (OUTCTRLx) bit, the PWM outputs is driven by software rather than the PWM generator.

In an Independent mode, with OUTCTRLx =1, the output bit OUTx, controls the PWMx channel. Setting and clearing the OUTx bit activates and deactivates the corresponding PWM channel.

The OUTCTRLx and OUTx bits are in the PWM Output Control (PMOUT) register.

During software output control, TOPNEG and BOTNEG still control output polarity.

In complementary channel operation odd and even OUTCTRLx must be switched concurrently for proper operation, the even-numbered OUTx bits replace the PWM generator outputs. Complementary channel pairs cannot be active simultaneously, and the deadtime generators continue to insert deadtime whenever an even OUTx bit toggles. Deadtime is not inserted when the odd OUTx bit toggles. The even OUTx bit controls complementary channel pairs when the odd OUTx bit is set. However, the even OUTx bit still controls complementary channel pairs with odd PWMx deactivated if the odd OUTx bit is cleared. In other words, setting the odd OUTx bit makes its corresponding PWMx the complement of its even pair, while clearing the odd OUTx bit deactivates the odd PWMx. Please refer to **Figure 11-20.**

Setting the OUTCTLx bits do not disable the PWM generators and current status sensing circuitry. They continue to run, but no longer control the output pins. When the OUTCTLx bits are cleared, the outputs of the PWM generator takes control of PWM outputs at the beginning of the next PWM cycle. Please refer to **Figure 11-20.**

Software can drive the PWM outputs even when PWM Enable (PWMEN) bit is set to zero.

Note: Avoid an unexpected deadtime insertion by clearing the OUTx bits before setting and after clearing the OUTCTLx bits.

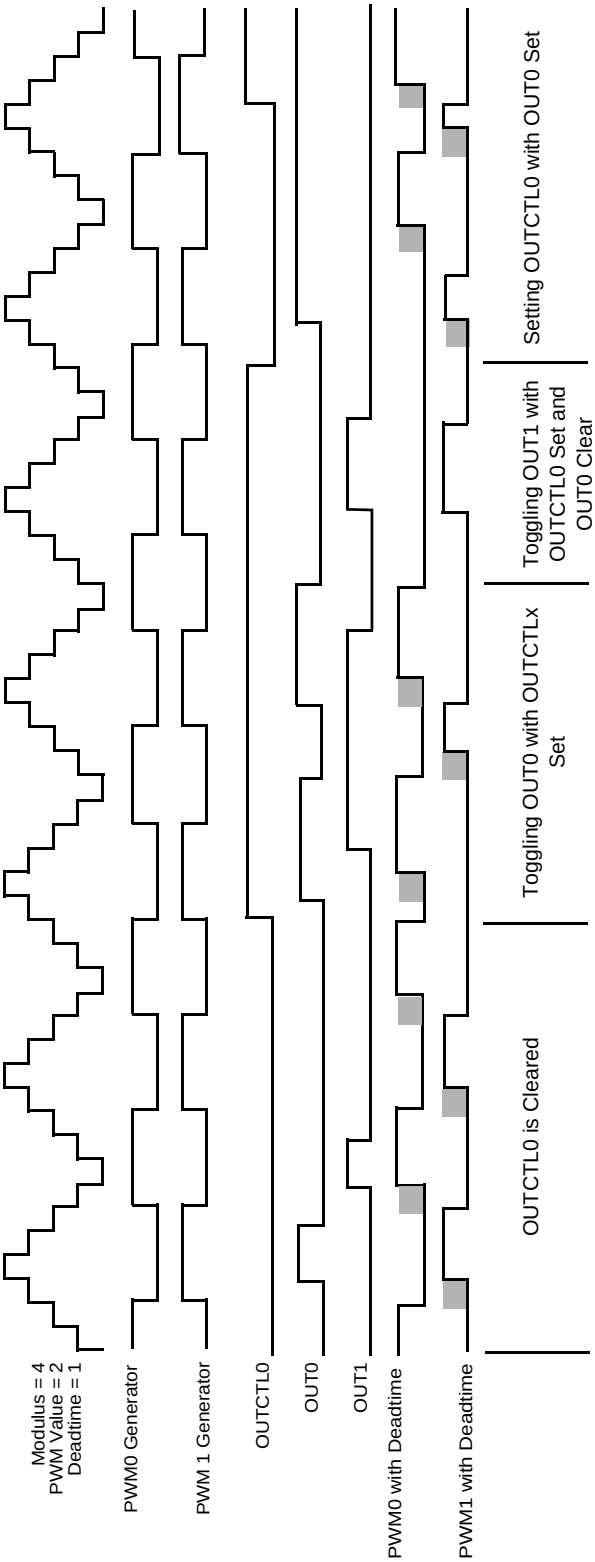


Figure 11-20. Software Output Control in Complementary Mode

11.6 PWM Generator Loading

11.6.1 Load Enable

The Load Okay (LDOK) bit enables loading the PWM generator with:

- A prescaler divisor from the PRSC1 and PRSC0 bits in PWM Control register
- A PWM period from the PWM Counter Modulus registers
- A PWM pulse width from the all PWM Value registers

LDOK ensures reloading of these PWM parameters simultaneously. Setting LDOK allows the prescale bits, PWMCM and PWMVALx registers to be loaded into a set of buffers. The loaded buffers are used by the PWM generator at the beginning of the next PWM reload cycle. Set LDOK by reading it, then writing a Logic 1 to it. After loading, LDOK is automatically cleared.

11.6.2 Load Frequency

The LDFQ3, LDFQ2, LDFQ1, and LDFQ0 bits in the PMCTL register select an integral loading frequency of one to 16-PWM reload opportunities. The LDFQ bits take effect at every PWM reload opportunity, regardless the state of the LDOK bit. The *half* bit in the PMCTL register controls half cycle reloads for Center-Aligned PWMs. If the *half* bit is set, a reload opportunity occurs at both beginning of PWM cycle or PWM half cycle. If the half bit is not set, a reload opportunity occurs only at the beginning of the cycle. Reload opportunities can only occur at the beginning of a PWM cycle in Edge-Aligned mode.

Note: Loading a new modulus on a half cycle will force the count to the new modulus value *minus one* on the next clock cycle. Half cycle reloads only changes reload rate in Center-Aligned mode. Enabling or disabling half cycle reloads in Edge-Aligned mode will have no effect on the reload rate.

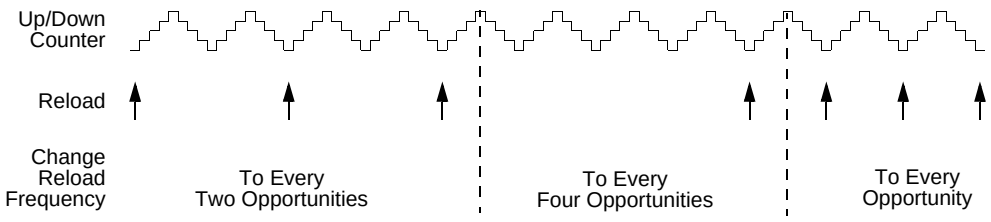


Figure 11-21. Full Cycle Reload Frequency Change

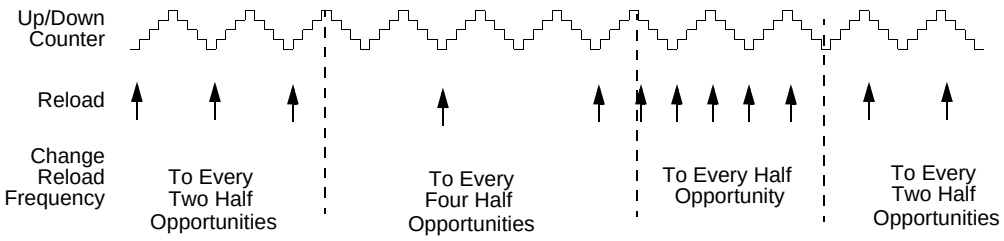


Figure 11-22. Half Cycle Reload Frequency Change

11.6.3 Reload Flag

At every reload opportunity the PWM Reload Flag (PWMF) bit in the PMCTL register is set. Setting PWMF happens even if an actual reload is prevented by the LDOK bit. If the PWM Reload Interrupt Enable (PWMRIE) bit is set, the PWMF generates core interrupt requests allowing software to calculate new PWM parameters in real time. When PWMRIE is not set, reloads still occur at the selected reload rate without generating interrupt requests. Clear PWMF by reading it then write a Logic 0 to it.

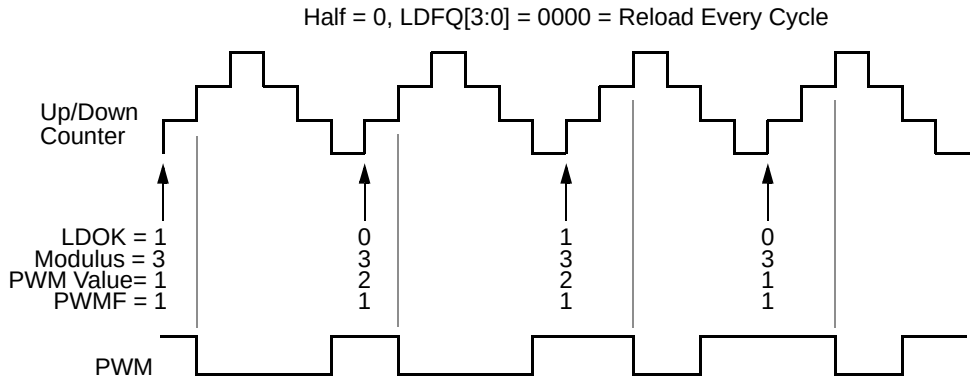


Figure 11-23. Full Cycle Center-Aligned PWM Value Loading

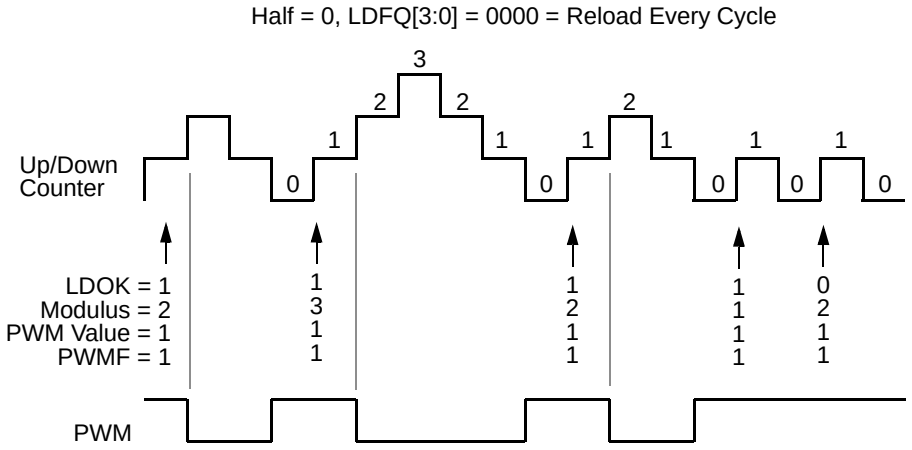


Figure 11-24. Full Cycle Center-Aligned Modulus Loading

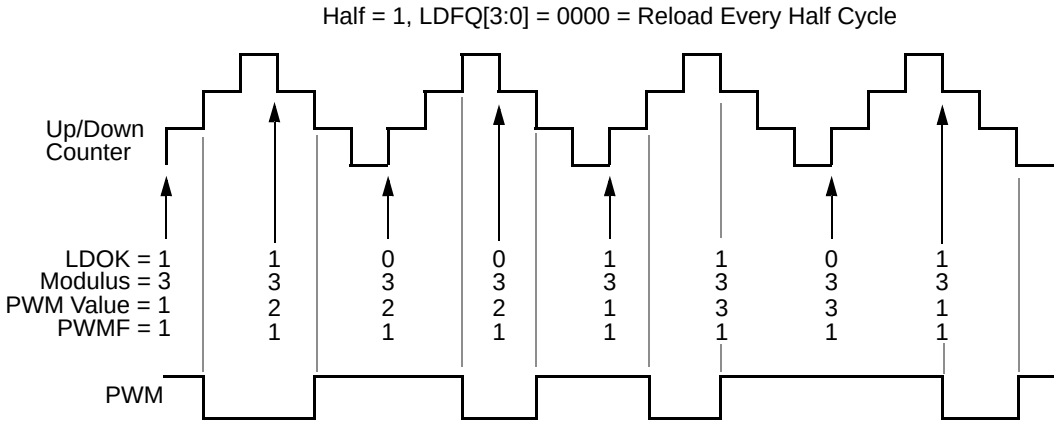


Figure 11-25. Half Cycle Center-Aligned PWM Value Loading

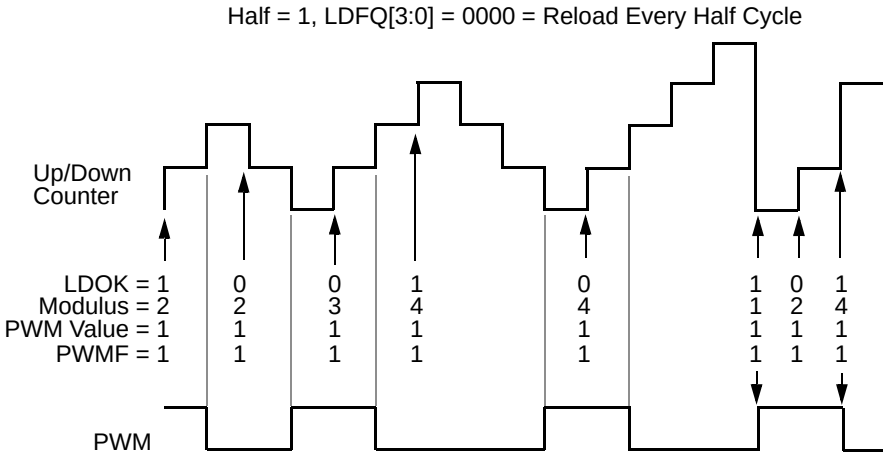


Figure 11-26. Half Cycle Center-Aligned Modulus Loading

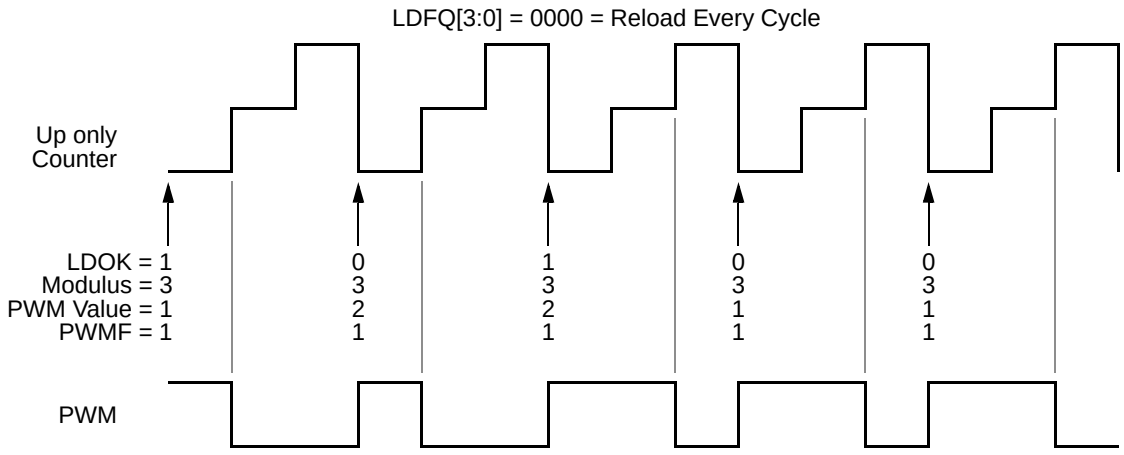


Figure 11-27. Edge-Aligned PWM Value Loading

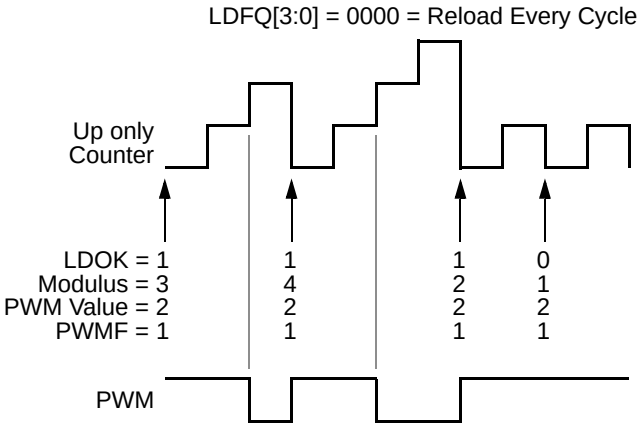


Figure 11-28. Edge-Aligned Modulus Loading

11.6.4 Synchronization Output

The PWM outputs a synchronization pulse connected as an input to the synchronization module, Timer C. A high-true pulse occurs for each reload of the PWM regardless of the state of the LDOK bit. When half cycle reloads are enabled, HALF = 1 in the PMCTL register, the pulse can occur on the half cycle.

11.6.5 Initialization

Initialize all registers and set the LOAD OKAY (LDOK) bit before setting the ENABLE (PWMEN) bit. With LDOK set, when the PWMEN bit is first set, a reload will immediately occur, thereby setting the PWMF bit. The PWMF bit generates an interrupt request if the

PWMRIE bit is set. In complementary channel operation with current status correction selected, even numbered PWM Value registers control the outputs for the first PWM cycle.

Note: Even if LDOK is not set, setting PWMEN also sets the PWMF. To prevent a core interrupt request, clear the PWMRIE bit before setting PWMEN.

Setting PWMEN for the first time after reset without first setting LDOK loads a prescaler divisor of one, a PWM value of $0000, and an unknown modulus. If the LDOK bit is not set after the PWMEN bit is cleared, then set (without a RESET) the value last loaded will be used in the PWM generated. If deadtime register is changed after PWMEN or OUTCTLx bit are set, an improper deadtime insertion could occur.

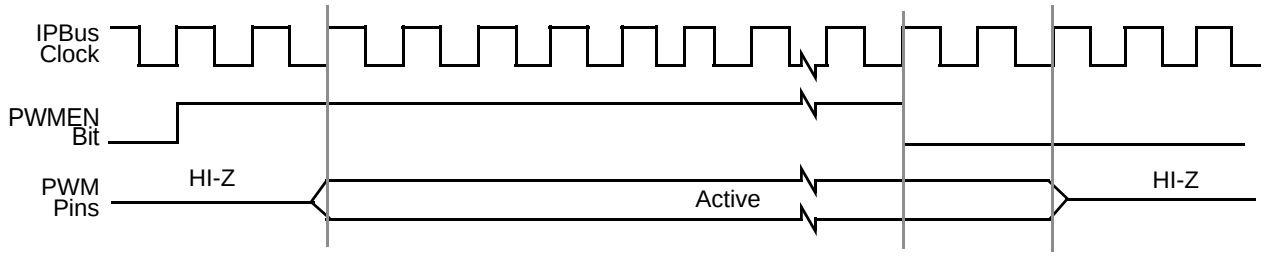


Figure 11-29. PWMEN and PWM Pins in Independent Operation (OUTCTL0–5 = 0)

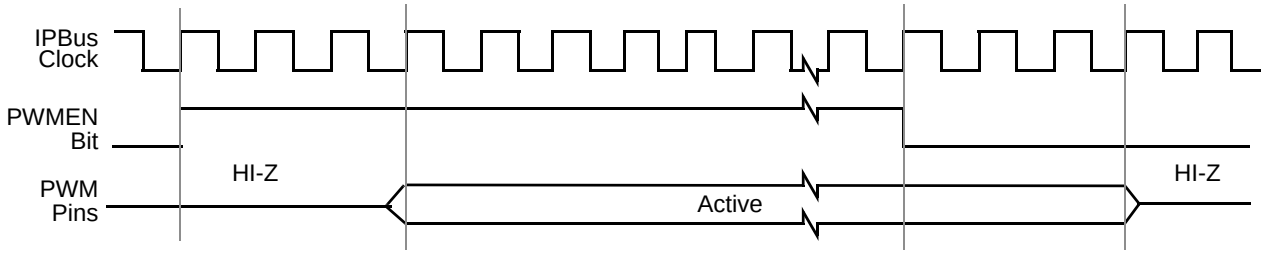


Figure 11-30. PWMEN & PWM Pins in Complement Operation (OUTCTL0,2,4 = 0)

When the PWMEN bit is cleared:

- The PWMx pins will be in their inactive status unless OUTCTLx = 1
- The PWM counter is cleared and does not count
- The PWM generator forces its outputs to zero
- The PWMF and pending interrupt requests are not cleared
- All fault circuitry remains active
- Software output control remains active if OUTCTLx = 1
- Deadtime insertion continues during software output control

11.7 Fault Protection

Fault protection can disable any combination of PWM pins. Faults are generated by a Logic 1 on any of the FAULT pins. Each FAULT pin can be mapped arbitrarily to any of the PWM pins. When fault protection hardware disables PWM pins, the PWM generator continues to run, only the output pins are deactivated. The fault decoder disables PWM pins selected by the fault logic and the disable mapping register. Please see **Figure 11-31.** Each bank of four bits in the disable mapping register control the mapping for a single PWM pin. Please refer to **Table 11-5.** The fault protection is enabled even when the PWM is not enabled; therefore, if a fault is latched in, it must be cleared prior to enabling the PWM to prevent an unexpected interrupt. Please see Section 11.9.3.4, “FAULTx Pin Acknowledge (FTACKx)—Bits 6, 4, 2, 0”.

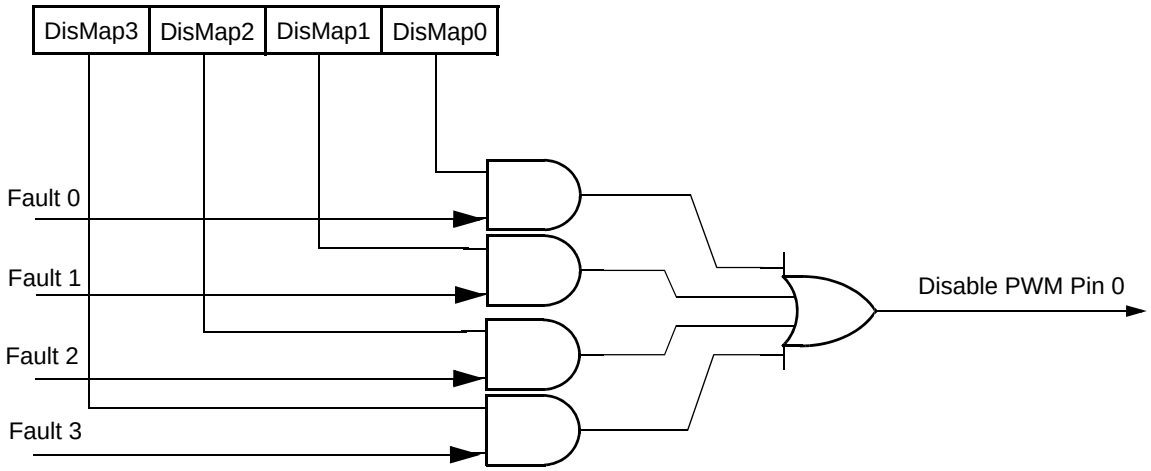


Figure 11-31. Figure 4-36 Fault Decoder for PWM 0

Table 11-5. Fault Mapping

PWM Pin	Controlling Register Bits
PWM0	DISMAP3—DISMAP0
PWM1	DISMAP7—DISMAP4
PWM2	DISMAP11—DISMAP8
PWM3	DISMAP15—DISMAP12
PWM4	DISMAP19—DISMAP16
PWM5	DISMAP23—DISMAP20

Note: For parts with less than four fault pins, the same controls apply. The unavailable DISMAP field bits should be set to zero. For example, if Fault3 is not available as an input, set DISMAP3 = 0.

11.7.1 Fault Pin Filter

Each fault pin has a filter to test for fault conditions. A fault input transition to a high state is not declared until the input is sampled high on two consecutive IPBus clocks. Only then FFLAGx and FPINx are set. The FPINx bit will remain set until the Fault input is detected low on two consecutive IPBus clocks. Clear FFLAGx by writing a Logic 1 to the corresponding fault acknowledge bit, FTACKx. If the FIEx, FAULTx pin interrupt enable bit is set, the FFLAGx flag generates an interrupt request. The interrupt request latch remains set until one of the following actions occur:

- Software clears the FFLAGx flag by writing a Logic 1 to the FTACKx bit
- Software clears the FIEx bit by writing a Logic 0 to it
- A reset occurs

11.7.2 Automatic Fault Clearing

In Automatic mode, when FMODEx is set, disabled PWM pins are enabled when the FAULTx pin returns to Logic 0 and a new PWM half cycle begins. Please refer to **Figure 11-32.** Clearing the FFLAGx flag does not affect disabled PWM pins when FMODEx is set.

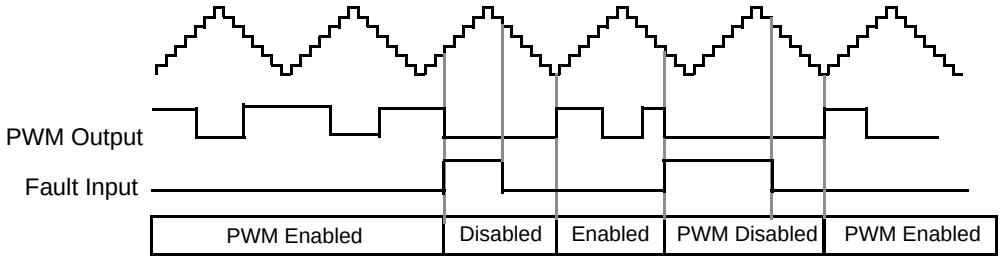


Figure 11-32. Automatic Fault Clearing

11.7.3 Manual Fault Clearing

In Manual mode, the fault pins are grouped in pairs, each pair sharing common functionality. A fault condition on Fault pins 0 and 2 can be cleared by software clearing the corresponding FFLAG bit, allowing the PWM(s) to enable at the next PWM half cycle regardless of the logic level at the Fault pin. **Figure 11-33.** A fault condition on Fault pins 1 and 3 can only be cleared by software clearing corresponding FFLAGx bit, allowing the PWM(s) to enable if a Logic Low at the Fault pin is detected at the start of the next PWM half cycle boundary. Please see **Figure 11-34.**

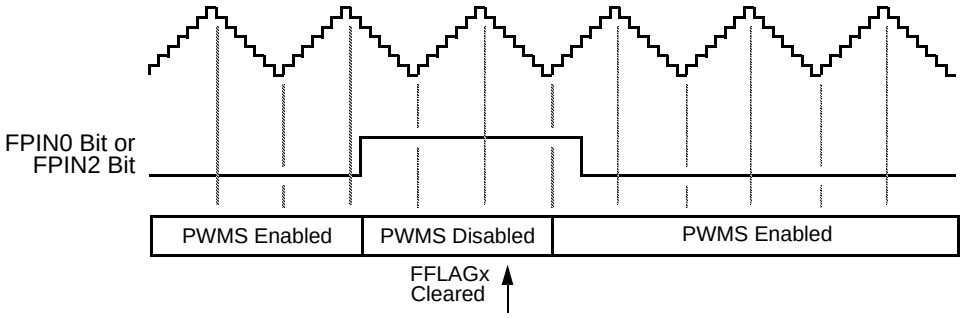


Figure 11-33. Manual Fault Clearing (Example 1)

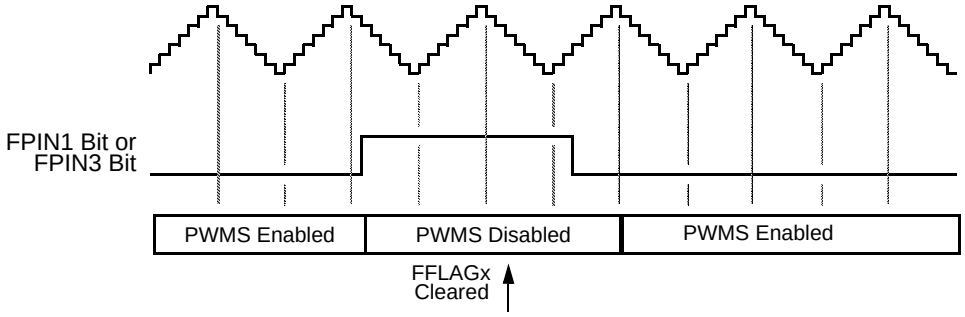


Figure 11-34. Manual Fault Clearing (Example 2)

Note: PWM half cycle boundaries occur at both the PWM cycle start and when the counter equals the modulus, so in Edge-Aligned operation full cycles and half cycles are equal.

Note: Fault protection also applies during software output control when the OUTCTLx bits are set. Fault clearing still occurs at half PWM cycle boundaries while the PWM generator is engaged where PWMEN equals one. But the OUTx bits can also control the PWM pins while the PWM generator is off where PWMEN equals zero. Thus, fault clearing occurs at IPBus cycles while the PWM generator is off and at the start of PWM cycles when the generator is engaged.

11.8 Pin Descriptions

The Pulse Width Modulator (PWM) is capable of having the following external pins.

11.8.1 PWM0–PWM5 Pins—(PWM0–5)

PWM0–PWM5 are output pins of the six PWM channels.

11.8.2 FAULT0–FAULT3 Pins—(FAULT0–3)

FAULT0–FAULT3 are input pins for disabling selected PWM outputs.

11.8.3 IS2 Pins—(IS0–2)

IS0–IS2 are current status pins for Top/Bottom pulse width correction in complementary channel operation while deadtime is asserted.

11.9 Register Definitions

Table 11-6. PWM Memory Map

Device	Peripheral	Address
801/802/ 803/805	PWMA_BASE	$0E00
	PWMB_BASE	$0EC0
807	PWMA_BASE	$1200
	PWMB_BASE	$1220

The address of a register is the sum of a base address and an address offset. The base address is defined at the core level and the address offset is defined at the module level. PWMA uses PWMA_BASE plus the given offset while PWMB uses PWMB_BASE plus the given offset in the table below.

Table 11-7. PWM Register Summary

Address + Offset	Address Acronym	Register Name	Access Type	Chapter Location
Base + $0	PMCTL	Control Register	Read/Write	Section 11.9.1
Base + $1	PMFCTL	Fault Control Register	Read/Write	Section 11.9.2
Base + $2	PMFSA	Fault Status & Acknowledge Register	Read/Write	Section 11.9.3
Base + $3	PMOUT	Output Control Register	Read/Write	Section 11.9.4
Base + $4	PMCNT	Counter Register	*Read-Only*	Section 11.9.5
Base + $5	PWMCM	Counter Modulo Register	Read/Write	Section 11.9.6
Base + $6 to $B	PWMVAL0-5	Value Register 0-5	Read/Write	Section 11.9.7
Base + $C	PMDEADTM	Deadtime Register	Read/Write	Section 11.9.8
Base + $D	PMDISMAP1	Disable Mapping Register 1	Read/Write	Section 11.9.9
Base + $E	PMDISMAP2	Disable Mapping Register 2	Read/Write	
Base + $F	PMCFG	Configure Register	Read/Write	Section 11.9.10
Base + $10	PMCCR	Channel Control Register	Read/Write	Section 11.9.11
Base + $11	PMPORT	Port Register	Read/Write	Section 11.9.12

Bit fields of each of the 18 registers are illustrated in **Figure 11-35.** Details of each follow.

Add. Offset	Register Name		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$0	PMCTL	R	LDFQ				HALF	IPOL2	IPOL1	IPOL0	PRSC		PWM RIE	PWMF	ISENS		LDOK	PWM EN
		W																
$1	PMFCTL	R	0	0	0	0	0	0	0	0	FIE3	FMODE3	FIE2	FMODE2	FIE1	FMODE1	FIE0	FMODE0
		W																
$2	PMFSA	R	FPIN3	FFLAG3	FPIN2	FFLAG2	FPIN1	FFLAG1	FPIN0	FFLAG0	0	0	DT5	DT4	DT3	DT2	DT1	DT0
		W										FTACK3		FTACK2		FTACK1		FTACK0
$3	PMOUT	R	PAD_ EN	0	OUTCTL[5:0]						0	0	OUT[5:0]					
		W																
$4	PMCNT	R	0	CR														
		W																
$5	PWMCM	R	0	PWMCM[14:0]														
		W																
$6-$B	PWMVAL0-5	R	PWMVAL															
		W																
$C	PMDEADTM	R	0	0	0	0	0	0	0	0	PWMDT							
		W																
$D	PMDISMAP1	R	DISMAP[15:0]															
		W																
$E	PMDISMAP2	R	0	0	0	0	0	0	0	0	DISMAP[23:16]							
		W																
$F	PMCFG	R	0	0	0	EDG	0	TOP NEG 45	TOP NED 23	TOP NEG 01	0	BOT NEG 45	BOT NEG 23	BOT NEG 01	INDEP 45	INDEP 23	INDEP 01	WP
		W																
$10	PMCCR	R	ENHA	0	MSK5	MSK4	MSK3	MSK2	MSK1	MSK0	0	0	VLMODE		0	SWP 45	SWP 23	SWP 01
		W																
$11	PMPORT	R	0	0	0	0	0	0	0	0	0	IS2	IS1	IS0	FAULT 3	FAULT 2	FAULT 1	FAULT 0
		W																

R	0	Read as 0
W		Reserved

Figure 11-35. PWM Register Map

11.9.1 PWM Control Register (PMCTL)

Base + $0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	LDFQ				HALF	IPOL2	IPOL1	IPOL0	PRSC		PWMRIE	PWMF	ISENS		LDOK	PWMEN
Write																
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-36. PWM Control Register (PMCTL)

See Table A-8, List of Programmer's Sheets

11.9.1.1 Load Frequency Bits (LDFQ)—Bits 15–12

These buffered read/write bits select the PWM load frequency according to **Table 11-8.** Reset clears the LDFQ bits, selecting loading every PWM opportunity. The occurrence of a PWM opportunity is determined by the half bit.

Note: The LDFQx bits take effect when the current load cycle is complete, regardless of the state of the Load Okay (LDOK) bit. Reading the LDFQx bits reads the buffered values and not necessarily the values currently in effect.

Table 11-8. PWM Reload Frequency

LDFQ	PWM Reload Frequency	LDFQ	PWM Reload Frequency
0000	Every PWM Opportunity	1000	Every 9 PWM Opportunities
0001	Every 2 PWM Opportunities	1001	Every 10 PWM Opportunities
0010	Every 3 PWM Opportunities	1010	Every 11 PWM Opportunities
0011	Every 4 PWM Opportunities	1011	Every 12 PWM Opportunities
0100	Every 5 PWM Opportunities	1100	Every 13 PWM Opportunities
0101	Every 6 PWM Opportunities	1101	Every 14 PWM Opportunities
0110	Every 7 PWM Opportunities	1110	Every 15 PWM Opportunities
0111	Every 8 PWM Opportunities	1111	Every 16 PWM Opportunities

11.9.1.2 Half Cycle Reload (HALF)—Bit 11

This read/write bit enables half cycle reloads in Center-Aligned PWM mode. This bit has no effect on Edge-Aligned PWMs.

- 0 = Half cycle reloads disabled
- 1 = Half cycle reloads enabled

11.9.1.3 Current Polarity 2 (IPOL2)—Bit 10

This buffered read/write bit selects the PWM value register for the PWM4 and PWM5 pins in top/bottom manual deadtime correction.

- 0 = PWM value register four in next PWM cycle
- 1 = PWM value register five in next PWM cycle

Note: The IPOLx bits take effect at the beginning of the next load cycle regardless of the state of the LDOK bit. Select top/bottom software correction by writing 0X to the current status bits, ISENS, in the PWM Control register. Reading the IPOLx bits reads the buffered values and not necessarily the values currently in effect.

11.9.1.4 Current Polarity 1 (IPOL1)—Bit 9

This buffered read/write bit selects the PWM value register for the PWM2 and PWM3 pins in top/bottom manual deadtime correction.

- 0 = PWM value register two in next PWM cycle
- 1 = PWM value register three in next PWM cycle

11.9.1.5 Current Polarity 0 (IPOL0)—Bit 8

This buffered read/write bit selects the PWM value register for the PWM0 and PWM1 pins in top/bottom manual deadtime correction.

- 0 = PWM value register zero in next PWM cycle
- 1 = PWM value register one in next PWM cycle

11.9.1.6 Prescaler (PRSC)—Bits 7–6

These buffered read/write bits select the PWM clock frequency illustrated in **Table 11-9.**

Table 11-9. PWM Prescaler

PRSC	PWM Clock Frequency
00	f_{IPBus}
01	$f_{IPBus}/2$
10	$f_{IPBus}/4$
11	$f_{IPBus}/8$

Note: Reading the PRSCx bits reads the buffered values, and not necessarily the values currently in effect. The PRSCx bits take effect at the beginning of the next PWM cycle and only when the LDOK bit is set.

11.9.1.7 PWM Reload Interrupt Enable (PWMRIE)—Bit 5

This read/write bit enables the PWMF flag to generate interrupt requests.

- 0 = PWMF interrupt request disabled
- 1 = PWMF interrupt request enabled

11.9.1.8 PWM Reload Flag (PWMF)—Bit 4

This read/write flag is set at the beginning of every reload cycle regardless of the state of the LDOK bit. Clear PWMF by reading it, then write a Logic 0 to it. If another reload occurs before the clearing sequence is complete, writing Logic 0 to PWMF has no effect.

- 0 = No new reload cycle since last PWMF clearing
- 1 = New reload cycle since last PWMF clearing

Note: Clearing PWMF satisfies pending PWMF interrupt request.

11.9.1.9 Current Status (ISENS)—Bits 3–2

These read/write bits select the top/bottom correction scheme, illustrated in **Table 11-2.** Reset clears the ISENSx bits. This selects manual correction or no correction, or automatic deadtime correction.

Note: The ISENSx bits are not buffered. Changing the current status sensing method can affect the present PWM cycle.

11.9.1.10 Load Okay (LDOK)—Bit 1

This read/write bit loads the prescaler bits of PMCTL and the entire PWMCM register and PWMVAL registers into a set of buffers. The buffered prescaler divisor, PWM counter modulus value, and PWM pulse width take effect at the next PWM reload. Set LDOK by reading it, then write a Logic 1 to it. LDOK is automatically cleared after the new values are loaded, or can be manually cleared before a reload by writing a Logic 0 to it.

- 0 = No action is taken
- 1 = Load prescaler, PWM modulus and PWM values into buffers

11.9.1.11 PWM Enable (PWMEN)—Bit 0

This read/write bit enables the PWM generator. When PWMEN equals zero, the PWM outputs are in their inactive states unless OUTCTLx equals one.

- 0 = PWM generator and PWM outputs disabled unless OUTCTRL = 1
- 1 = PWM generator and PWM outputs enabled

Note: PWM pin outputs are in tri-state if Output Pad Enable (PAD_EN) bit in PWM Output Control (PMOUT) is cleared.

11.9.2 PWM Fault Control Register (PMFCTL)

Base+ $1	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	0	0	0	FIE3	FMODE3	FIE2	FMODE2	FIE1	FMODE1	FIE0	FMODE0
Write																
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-37. PWM Fault Control Register (PMFCTL)

See Table A-8, List of Programmer's Sheets

11.9.2.1 Reserved—Bits 15–8

This bit field is reserved or not implemented. It is read as 0 and cannot be modified by writing.

11.9.2.2 FAULTx Pin Interrupt Enable (FIEx)—Bits 7, 5, 3, 1

This read/write bit enables the interrupt request generated by the FAULTx pin.

- 0 = FAULTx interrupt request disabled
- 1 = FAULTx interrupt request enabled

Note: The fault protection circuit is independent of the FIEx bits and is always active. If a fault is detected, the PWM pins are disabled according to the PWM disable mapping register.

11.9.2.3 FAULTx Pin Clearing Mode (FMODEx)—Bits 6, 4, 2, 0

This read/write bit selects automatic or manual clearing of FAULTx pin faults.

- 0 = Manual fault clearing of FAULTx pin faults
- 1 = Automatic fault clearing of FAULTx pin faults

11.9.3 PWM Fault Status and Acknowledge Register (PMFSA)

Base + $2	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	FPIN3	FFLAG3	FPIN2	FFLAG2	FPIN1	FFLAG1	FPIN0	FFLAG0	0	0	DT5	DT4	DT3	DT2	DT1	DT0
Write										FTACK3		FTACK2		FTACK1		FTACK0
RESET	U	0	U	0	U	0	U	0	0	0	0	0	0	0	0	0

Figure 11-38. PWM Fault Status and Acknowledge Register (PMFSA)

See Table A-8, List of Programmer's Sheets

11.9.3.1 FAULTx Pin (FPINx)—Bits 15, 13, 11, 9

These *read-only* bits reflect the current state of the filtered FAULTx bit. A reset has no effect on FPINx.

- 0 = Logic 0 on the FAULTx bit
- 1 = Logic 1 on the FAULTx bit

11.9.3.2 FAULTx Pin Flag (FFLAGx)—Bits 14, 12, 10, 8

These *read-only* flags are set within two IPBus cycles after a rising edge on the FAULTx pin. Clear FFLAGx by writing a Logic 1 to the FTACKx bits in the PMFSA register. A reset clears FFLAGx.

- 0 = No fault on the FAULTx bit
- 1 = Fault on the FAULTx bit

11.9.3.3 Reserved—Bit 7

This bit is reserved or not implemented. It is read as 0 and cannot be modified by writing.

11.9.3.4 FAULTx Pin Acknowledge (FTACKx)—Bits 6, 4, 2, 0

Writing a Logic 1 to FTACKx clears FFLAGx. Writing a Logic 0 has no effect. Reading these bits reads the appropriate DTx bit. Please see **Section 11.9.3.5, "Deadtime X (DTx)—Bits 5–0".** Reset clears FTACKx. The fault protection is enabled even when the PWM is not enabled; therefore if a fault is latched, it must be cleared prior to enabling the PWM to prevent an unexpected interrupt.

11.9.3.5 Deadtime X (DTx)—Bits 5–0

These are *read-only* bits. The DTx bits are grouped in pairs, DT0 and DT1, DT2 and DT3, DT4 and DT5. Each pair reflects the corresponding ISx pin value as sampled during deadtime period. A reset clears these bits.

11.9.4 PWM Output Control Register (PMOUT)

This is a read/write register.

Base + $3	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	PAD_EN	0	OUTCTL[5:0]						0	0	OUT[5:0]					
Write																
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-39. PWM Output Control Register (PMOUT)

See Table A-8, List of Programmer's Sheets

11.9.4.1 Output Pad Enable (PAD_EN)—Bit 15

The PWM output pads can be enabled or disabled by setting the PAD_EN bit. The power-up default has the pads disabled. This bit does not affect the functionality of the PWM, so the PWM module can be energized with the output pads disabled. This enable is to power-up with a safe default value for the PWM drivers.

- 0 = Output pads disabled (tri-stated)
- 1 = Output pads enabled (not tri-stated)

11.9.4.2 Reserved—Bit 14

This bit field is reserved or not implemented. It is read as 0 and cannot be modified by writing.

11.9.4.3 Output Control Enables (OUTCTRL5–0)—Bits 13–8

These read/write bits enable software control of their corresponding PWM pin. When OUTCTRLx is set, the OUTx bit activates and deactivates the PWMx output. A reset clears the OUTCTRL bits. When operating the PWM in Complementary mode, these bits must be switched in pairs for proper operation. Please see Section 11.5, “Software Output Control” for details.

- 0 = Software control disabled
- 1 = Software control enabled

11.9.4.4 Reserved—Bits 7–6

This bit field is reserved or not implemented. It is read as 0 and cannot be modified by writing.

11.9.4.5 Output Control (OUT5–0)—Bits 5–0

When the corresponding OUTCTRL bit is set, these read/write bits control the PWM pins, illustrated in **Table 11-10.**

Table 11-10. Software Output Control

Outx Bit	Complementary Channel Operation	Independent Channel Operation
OUT0	1—PWM0 is Active 0—PWM0 is Inactive	1—PWM0 is Active 0—PWM0 is Inactive
OUT1	1—PWM1 is Complement of PWM 0 0—PWM1 is Inactive	1—PWM1 is Active 0—PWM1 is Inactive
OUT2	1—PWM2 is Active 0—PWM2 is Inactive	1—PWM2 is Active 0—PWM2 is Inactive
OUT3	1—PWM3 is Complement of PWM 2 0—PWM3 is Inactive	1—PWM3 is Active 0—PWM3 is Inactive
OUT4	1—PWM4 is Active 0—PWM4 is Inactive	1—PWM4 is Active 0—PWM4 is Inactive
OUT5	1—PWM5 is Complement of PWM 4 0—PWM5 is Inactive	1—PWM5 is Active 0—PWM5 is Inactive

11.9.5 PWM Counter Register (PMCNT)

Base + $4	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	CR														
Write																
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-40. PWM Counter Register (PMCNT)

See Table A-8, List of Programmer's Sheets

11.9.5.1 Reserved—Bit 15

This bit is reserved or not implemented. It is read as 0 and cannot be modified by writing.

11.9.5.2 Counter Register (CR)—Bits 14–0

These *read-only* bits display the state of the 15-bit PWM counter.

11.9.6 PWM Counter Modulo Register (PWMCM)

Base + $5	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	PWMCM														
Write																
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-41. PWM Counter Modulo Register (PWMCM)

See Table A-8, List of Programmer's Sheets

11.9.6.1 Reserved—Bit 15

This bit is reserved or not implemented. It is read as 0 and cannot be modified by writing.

11.9.6.2 Counter Modulo (PWMCM)—Bits 14–0

The 15-bit unsigned value written to this buffered read/write register defines the number of PWM clocks to use in determining the PWM period. Do not write a modulus value of zeros into these bits.

Note: The PWM counter modulo register is buffered. The value written does not take effect until the LDOK bit is set and the next PWM load cycle begins. Reading PWMCM reads the value in a buffer. It is not necessarily the value the PWM generator is currently using.

11.9.7 PWM Value Registers (PWMVAL0–5)

The 16-bit signed value in these buffered, read/write registers is the PWM pulse width in PWM clock periods for each of the output channels.

Base + $6	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	PWMVAL															
Write																
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-42. PWM Value Register (PWMVAL0-5)

See Table A-8, List of Programmer's Sheets

11.9.7.1 Value (PWMVAL)—Bits 15–0

The PWM Value registers are buffered. The value written does not take effect until the LDOK bit is set and the next PWM load cycle begins. Reading PWMVALx reads the value in a buffer and not necessarily the value the PWM generator is currently using.

A PWM value less than, or equal to zero, deactivates the PWM output for the entire PWM period. A PWM value greater than, or equal to the modulus, activates the PWM output for the entire PWM period. Please see **Table 11-1.**

Note: The terms activate and deactivate refer to the high and low logic states of the PWM outputs.

11.9.8 PWM Deadtime Register (PMDEADTM)

Base + $C	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	0	0	0	PWMDT							
Write																
RESET	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Figure 11-43. PWM Deadtime Register (PMDEADTM)

See Table A-8, List of Programmer's Sheets

11.9.8.1 Reserved—Bits 15–8

This bit field is reserved or not implemented. It is read as 0 and cannot be modified by writing.

11.9.8.2 Deadtime (PWMDT)—Bits 7–0

The 8-bit value written to this write-protectable register is the number of PWM clock cycles in complementary channel operation. A reset sets the PWM deadtime register to a default value of 0x00FF, selecting a deadtime of 256-PWM clock cycles minus one IPBus clock cycle. This register is write protected after the Write Protect (WP) bit in the PWM configuration register is set. Please refer to **Section 11.9.10, "PWM Configure Register (PMCFG)".**

Note: Deadtime is affected by changes to the prescaler value. The deadtime duration is determined as follows: DT = P × PWMDT - 1 IPBus clocks, where DT is deadtime, P is the prescaler value, PWMDT is the programmed value of deadtime. For example: if the prescaler is programmed for a divide-by-two and PWMDT is set to five, then P = 2 and the deadtime value is equal to:

$$DT = 2 \times 5 - 1 = 9 \text{ IPBus clock cycles.}$$

11.9.9 PWM Disable Mapping Registers (PMDISMAP1-2)

These *write-protectable* registers determine which PWM pins are disabled by the fault protection inputs, provided in **Table 11-5.** Reset sets all of the bits used in the PWM disable mapping registers. These registers are write-protected after the WP bit in the PWM Configure register is set. Reserved bits 15-8 in the PMDISMAP2 register cannot be modified. The bits are read as 0.

Base + $D	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	DISMAP[15:0]															
Write																
RESET	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 11-44. PWM Disable Mapping Register 1 (PMDISMAP1)

See Table A-8, List of Programmer's Sheets

Base + $E	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	0	0	0	DISMAP [23:16]							
Write																
RESET	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Figure 11-45. PWM Disable Mapping Register 2 (PMDISMAP2)

See Table A-8, List of Programmer's Sheets

11.9.10 PWM Configure Register (PMCFG)

This *write-protectable* register contains the configuration bits determining PWM modes of operation detailed below. This register cannot be modified after the WP bit is set.

Base + $F	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	EDG	0	TOPNEG 45	TOPNEG 23	TOPNEG 01	0	BOTNEG 45	BOTNEG 23	BOTNEG 01	INDEP 45	INDEP 23	INDEP 01	WP
Write																
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-46. PWM Configure Register (PMCFG)

See Table A-8, List of Programmer's Sheets

11.9.10.1 Reserved—Bits 15–13

This bit is reserved or not implemented. It is read as 0 and cannot be modified by writing.

11.9.10.2 Edge-Aligned or Center-Aligned PWMs (EDG)—Bit 12

This *write-protectable* bit determines whether all PWM channels will use Edge-Aligned or Center-Aligned wave forms.

- 0 = Center-Aligned PWMs
- 1 = Edge-Aligned PWMs

11.9.10.3 Reserved—Bit 11

This bit is reserved or not implemented. It is read as 0 and cannot be modified by writing.

11.9.10.4 Top-Side PWM Polarity (TOPNEG)—Bits 10–8

This *write-protectable* bit determines the polarity for the top-side PWMs.

- 0 = Positive top-side polarity
- 1 = Negative top-side polarity

11.9.10.5 Bottom-Side PWM Polarity (BOTNEG)—Bits 6–4

This *write-protectable* bit determines the polarity for the bottom-side PWMs.

- 0 = Positive bottom-side polarity
- 1 = Negative bottom-side polarity

11.9.10.6 Independent or Complement Pair Operation (INDEP)—Bits 3–1

This *write-protectable* bit determines if the PWM channels will be independent PWMs or complementary PWM pairs.

- 0 = Complementary PWM pair
- 1 = Independent PWMs

Note: Each pair of PWM channels can be configured: channel zero to one, channel two to three, and channel four to five.

11.9.10.7 Write Protect (WP)—Bit 0

This bit enables write-protection for all write-protectable registers. While clear, *WP allows write-protected registers and bits to be written*. When set, WP prevents writes to write-protectable registers and bits. Once set, WP can be cleared only by a reset. *Write-protectable* registers and bits include: PMDISMAP1–2, PMDEADTM, PMCFG, and the ENHA bit in the Channel Control Register (PMCCR). The VLMODE, SWP45, SWP23, and SWP01 bits in the PMCCR are protected when the Enable Hardware Acceleration (ENHA) bit is set to zero in the PMCCR. ENHA is in turn, protected by setting the WP bit in the PMCFG.

- 0 = Write-protectable registers may be written
- 1 = Write-protectable registers are write protected

Note: The write to PMCFG setting the WP bit is the last write accepted to the register until reset.

11.9.11 PWM Channel Control Register (PMCCR)

Base + $10	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	ENHA	0	MSK5	MSK4	MSK3	MSK2	MSK1	MSK0	0	0	VLMODE		0	SWP45	SWP23	SWP01
Write																
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-47. PWM Channel Control Register (PMCCR)

See Table A-8, List of Programmer's Sheets

This *write-protectable* register contains the configuration bits determining PWM modes of operation detailed below. The ENHA bit cannot be modified after the WP bit in the PMCFG register is set. In turn, ENHA provides protection for the VLMODE, SWP45, SWP23 and SWP01 bits. Mask bits 0-5 are not write-protectable.

11.9.11.1 Enable Hardware Acceleration (ENHA)—Bit15

This bit enables writing to the VLMODE, SWP45, SWP23, and SWP01 bits. The bit is *write-protectable* by WP bit in the PMCFG register.

- 0 = Disable writing to VLMODE, SWP45, SWP23, and SWP01 bits
- 1 = Enable writing to VLMODE, SWP45, SWP23, and SWP12 bits

11.9.11.2 Reserved—Bit 14

This bit is reserved or not implemented. It is read as 0 and cannot be modified by writing.

11.9.11.3 Mask (MSK5–0)—Bits 13–8

These six bits determine the mask for each of the PWM logical channels.

In Independent Mode:

- 0 = Unmasked
- 1 = Masked, channel deactivated

In Complementary Mode:

Only MSK0, MSK2, and MSK4 take effect. MSK1, MSK3, and MSK5 are unused.

- 0 = Normal operation
- 1 = Top Channel deactivated and Bottom Channel activated

11.9.11.4 Reserved—Bits 7–6

These bits are reserved or not implemented. They are read as 0 and cannot be modified by writing.

11.9.11.5 Value Register Load Mode (VLMODE)—Bits 5–4

These two bits determine the way the PWM value registers are being loaded. These bits are write-protected when ENHA is zero.

- 00 = Each PWM value register is accessed independently
- 01 = Writing to PWM value register zero also writes to PWM value registers one to five
- 10 = Writing to PWM value register zero also writes to PWM value registers one to three
- 11 = Reserved

11.9.11.6 Reserved—Bit 3

This bit is reserved or not implemented. It is read as 0 and cannot be modified by writing.

11.9.11.7 Swap45 (SWP45)—Bit 2

This bit is write-protected when ENHA is zero.

- 0 = No swap
- 1 = Channel four and channel five are swapped

11.9.11.8 Swap23 (SWP23)—Bit 1

This bit is write-protected when ENHA is zero.

- 0 = No swap
- 1 = Channel two and channel three are swapped

11.9.11.9 Swap01 (SWP01)—Bit 0

This bit is write-protected when ENHA is zero.

- 0 = No swap
- 1 = Channels zero and one are swapped

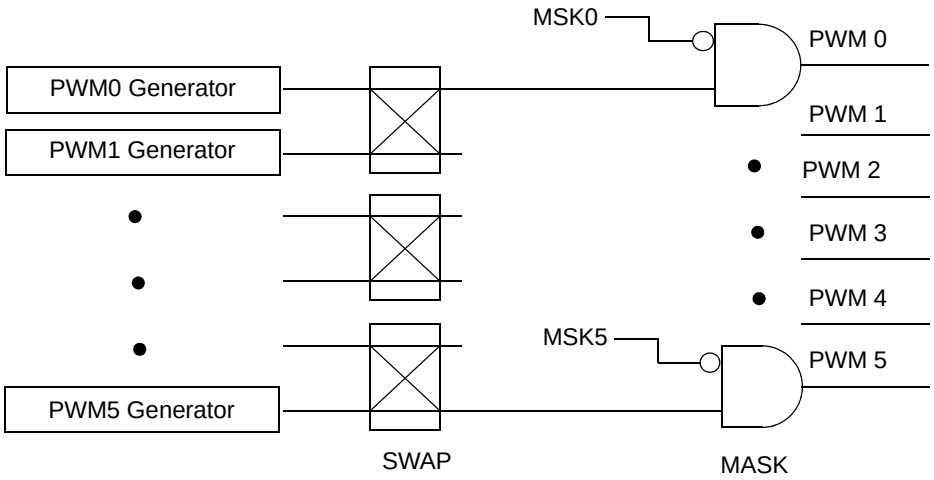


Figure 11-48. Channel Swapping

11.9.12 PWM Port Register (PMPORT)

This register contains values of the three current status inputs, bits six, five, and four. Additionally, it contains the four fault inputs, bits three, two, one, and zero. This register may be read while the PWM is active.

Base + $11	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	0	0	0	0	PORT						
Write																
RESET	0	0	0	0	0	0	0	0	0	U	U	U	U	U	U	U

Figure 11-49. PWM Port Register (PMPORT)

See Table A-8, List of Programmer's Sheets

11.9.12.1 Reserved—Bits 15–7

This bit field is reserved or not implemented. It is read as 0 and cannot be modified by writing.

11.9.12.2 Port (PORT)—Bits 6–0

This read-only field contains the values of the current status and fault inputs as shown in **Table 11-11**.

Table 11-11. PMPORT[PORT] Values

PORT Bit	Value
6	IS2 status input
5	IS1 status input
4	IS0 status input
3	FAULT3 fault input
2	FAULT2 fault input
1	FAULT1 fault input
0	FAULT0 fault input

11.10 Clocks

The system IPBus clock is the only clock required by this module. Along with the PWM prescaler, it determines the amount of time allocated for a single PWM bit value.

11.11 Interrupts

Five PWM sources can generate two interrupt requests to the core:

- Reload Flag (PWMF)—PWMF is set at the beginning of every reload cycle. The reload interrupt enable bit, PWMRIE, enables PWMF to generate core interrupt requests. PWMF and PWMRIE are in PWM Control (PMCTL) register.
- Fault Flags (FFLAG0–FFLAG3)—The FFLAGx bit is set when a Logic 1 occurs on the FAULTx pin. The fault pin interrupt enable bits, FIE0–FIE3, enable the FFLAGx flags to generate core interrupt requests. FFLAG0–FFLAG3 are in the Fault Status (PMFSA) register. FIE0–FIE3 are in the Fault Control (PMFCTL) register.

11.12 Resets

All PWM registers are reset to their default values upon any system reset.

Table 11-12. Document Revision History for Chapter 11

Version History	Description of Change
Rev. 8	Formatting, layout, spelling, and grammar corrections. Added revision history table. Changed the field name in the PMCNT register (was CNT, is CR). Changed the field name in the PWMCM register (was CM, is PWMCM). Changed the field name in the PWMVAL register (was VAL, is PWMVAL). Updated Section 11.9.12 with the proper field name and description.

Chapter 12
Serial Communications Interface (SCI)

12.1 Introduction

This chapter describes the Serial Communications Interface (SCI) module. The module allows asynchronous serial communications with peripheral devices and other controllers.

12.2 Features

- Full-duplex or Single-Wire Operation
- Standard mark/space Non-Return-to-Zero (NRZ) format
- 13-bit baud rate selection
- Programmable 8- or 9-bit data format
- Separately enabled transmitter and receiver
- Separate receiver and transmitter interrupt requests
- Programmable polarity for transmitter and receiver
- Two receiver wake-up methods:
  - Idle line
  - Address mark
- Interrupt-driven operation with seven flags:
  - Transmitter empty
  - Transmitter idle
  - Receiver full
  - Receiver overrun
  - Noise error
  - Framing error
  - Parity error
- Receiver framing error detection
- Hardware parity checking

- 1/16 bit-time noise detection

12.3 Block Diagram

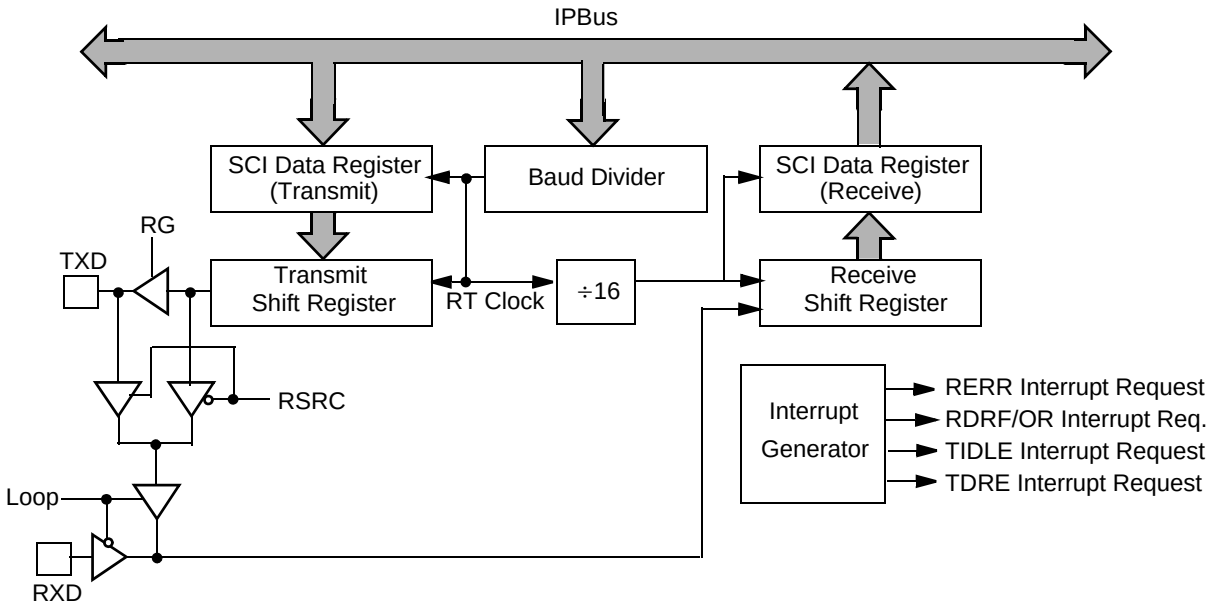


Figure 12-1. SCI Block Diagram

12.4 Functional Description

Figure 12-1 explains the SCI module structure. The SCI allows full duplex, asynchronous, Non-Return-to-Zero (NRZ) serial communication between the controller and remote devices, including other controllers. The SCI transmitter and receiver operate independently although they use the same baud rate generator. The controller monitors the status of the SCI, writes the data to be transmitted, and processes received data.

The SCI has one each input and output signals. Data transmits on the TXD pin and receives on the RXD pin. SCI Data Register (SCIDR) holds received bytes and bytes to be transmitted, actually consists of two physically different registers. To send software writes a byte to SCIDR; to receive, software reads a byte from SCIDR. However, if software has not read the first byte by the time the second byte is received, the second byte will be lost and Overrun (OR) flag will be set.

When initializing the SCI, be certain to set the proper peripheral enable bits in the General-Purpose Input/Output (GPIO) registers as well as any pull-up enables if the SCI pins are multiplexed with GPIO pins.

12.4.1 Data Frame Format

The SCI uses the standard NRZ mark/space data frame format illustrated in **Figure 12-2.**

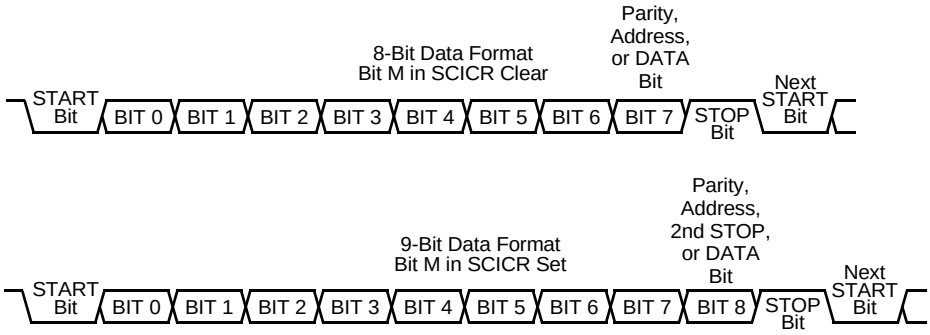


Figure 12-2. SCI Data Frame Formats

Each data character is contained in a frame including a START bit, eight or nine DATA bits, and a STOP bit. Clearing the Mode (M) bit in the SCI Control Register (SCICR) configures the SCI for 8-bit data characters. A frame with eight DATA bits has a total of 10 bits. Formats are provided in **Table 12-1.**

Table 12-1. Example 8-Bit Data Frame Formats

Start Bit	Data Bits	Address Bit	Parity Bit	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. Please see Section 12.4.4.6, Receiver Wake-Up.

Setting the M bit configures the SCI for 9-bit data characters. A frame with nine DATA bits has a total of 11 bits. Formats are provided in **Table 12-2.**

Table 12-2. Example 9-Bit Data Frame Formats

Start Bit	Data Bits	Address Bit	Parity Bit	Stop Bit
1	9	0	0	1
1	8	0	0	2[2]
1	8	0	1	1
1	8	1[1]	0	1

1.The address bit identifies the frame as an address character. Please see Section 12.4.4.6, Receiver Wake-Up.
2. The user must implement the second stop bit by setting the MSB of the data bits when transmitting and by masking the MSB when receiving.

12.4.2 Baud Rate Generation

A 13-bit modulus counter in the baud rate generator derives the baud rate for both the receiver and the transmitter. A value of 1 to 8191 written to the SBR bit in the SCI Baud Rate (SCIBR) register determines the module clock divisor. A value of zero disables the Baud Rate Generator. The clock generated by SCIBR is called RT clock; a frequency of 16 times the baud rate. The RT clock is synchronized with the IPBus clock, driving the receiver. The RT 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 two sources of error:

1. The Integer division of the module clock may not give the exact target frequency.
2. Synchronization with the bus clock can cause phase shift.

Table 12-3 lists examples of achieving target baud rates with a module clock frequency of 40MHz.

Table 12-3. Example Baud Rates (Module Clock = 40MHz)

SBR Bits	Receiver Clock (Hz)	Transmitter Clock (Hz)	Target Baud Rate	Error (%)
65	615,384.6	38,461.5	38,400	0.16
130	307,692.3	19,230.8	19,200	0.16
260	153,846.1	9,615.4	9,600	0.16
521	76,775.4	4,798.5	4,800	0.03
1042	38,387.7	2,399.2	2,400	0.03
2083	19,203.1	1,200.2	1,200	0.02
4167	9,599.2	600.0	600	0.01

Note: Maximum baud rate is IPBus clock rate divided by 16. System overhead may preclude processing the data at this speed.

12.4.3 Transmitter

Figure 12-3 illustrates the transmitter functions block diagram with detailed discussion of the transmitter in the following sections.

12.4.3.1 Character Length

The SCI transmitter accommodates either 8- or 9-bit data characters, determined by the state of the M bit in the SCICR.

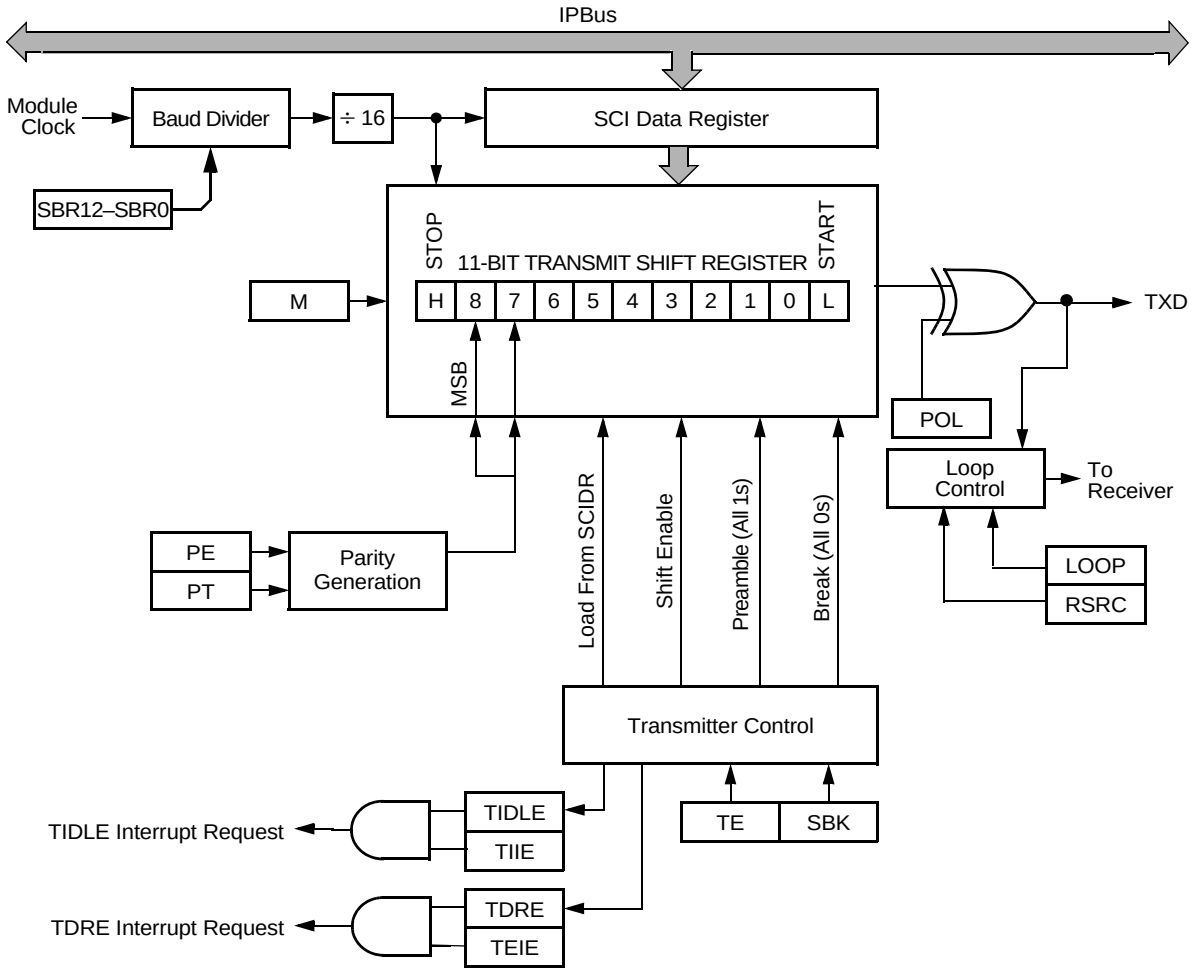


Figure 12-3. SCI Transmitter Block Diagram

12.4.3.2 Character Transmission

During an SCI transmission, the Transmit Shift register moves a frame out to the TXD pin. The SCI Data Register (SCIDR) is the buffer between the Internal Data Bus and the Transmit Shift registers.

Modifying the Transmitter Enable (TE) bit from 0 to 1 automatically loads the Transmit Shift register with a preamble containing all Logic 1s with no START, STOP, or PARITY bit. After the preamble shifts out, control logic automatically transfers the data from the SCIDR into the Transmit Shift register. A Logic 0 START bit automatically goes into the Least Significant Bit (LSB) position of the Transmit Shift register. A Logic 1 STOP bit goes into the Most Significant Bit (MSB) position of the frame.

Hardware supports odd or even parity. When PARITY is enabled, the MSB of the data character is replaced by the PARITY bit.

The Transmit Data Register Empty (TDRE) flag in the SCISR becomes set when the SCIDR transfers a character to the Transmit Shift register. The TDRE flag indicates the SCIDR can accept new transmitted data. If the TEIE bit in the SCICR is also set, the TDRE flag generates a Transmitter Empty Interrupt Request.

When the SCI Transmit Shift register is not transmitting a frame and TE = 1, the TXD pin goes to the idle condition, Logic 1. If software clears TE while a transmission is in progress, the frame in the SCI Transmit Shift register continues to shift-out the transmitter, then relinquishes control of the port I/O pin upon completion of the current transmission. This action causes the TXD pin to go into a HighZ state even if there is data pending in the SCIDR. 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 initiate a SCI transmission:

- Enable the transmitter by writing a Logic 1 to the TE bit in the SCICR.
- Wait for the TDRE flag to be set.
- Clear the TDRE flag by first reading the SCISR, then write to the SCIDR.
- Repeat Steps 2 and 3 for each subsequent transmission.

To separate messages with preambles having minimum idle line time, use this sequence between messages:

- Write the last character of the first message to SCIDR.
- Wait for the TDRE flag to go high, indicating the transfer of the last frame to the Transmit Shift register.
- Queue a preamble by clearing, then set the TE bit.
- Write the first character of the second message to SCIDR.

12.4.3.3 Break Characters

Writing a Logic 1 to the Send Break (SBK) bit in the SCICR loads the Transmit Shift register with a break character. A break character contains all Logic 0s without START, STOP, or PARITY bits. Break character length depends on the Mode (M) bit in the SCICR. 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 Transmit Shift register finishes transmitting the last break character subsequently transmitting at least one Logic 1. The automatic Logic 1 at the end of the last break character guarantees the recognition of the START bit of the next frame.

The SCI 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 SCI registers:

- Sets the Framing Error (FE) flag
- Sets the Receive Data Register Full (RDRF) flag
- Clears the SCI Data Register (SCIDR)
- May set the Overrun (OR) flag, Noise Flag (NF), Parity Error (PE) flag, or the Receiver Active Flag (RAF). Please see SCISR in **Section 12.6.3, "SCI Status Register (SCISR)".**

12.4.3.4 Preambles

A preamble contains all Logic 1s with no START, STOP, or PARITY bit. A preamble length depends on the M bit in the SCICR. The preamble is a synchronizing mechanism initiating the first transmission begun after modifying the TE bit from zero to one.

To queue a preamble between two transmissions:

- After writing the last character of the first transmission to the Transmit register, wait for TDRE flag to be set.
- When the TDRE flag becomes set, clear and set the TE bit. This will queue a preamble after the last character of the first transmission.
- After setting the TE bit, immediately write the first character of the second transmission to the SCIDR.

12.4.4 Receiver

Figure 12-4 illustrates the block diagram of the SCI receiver function.

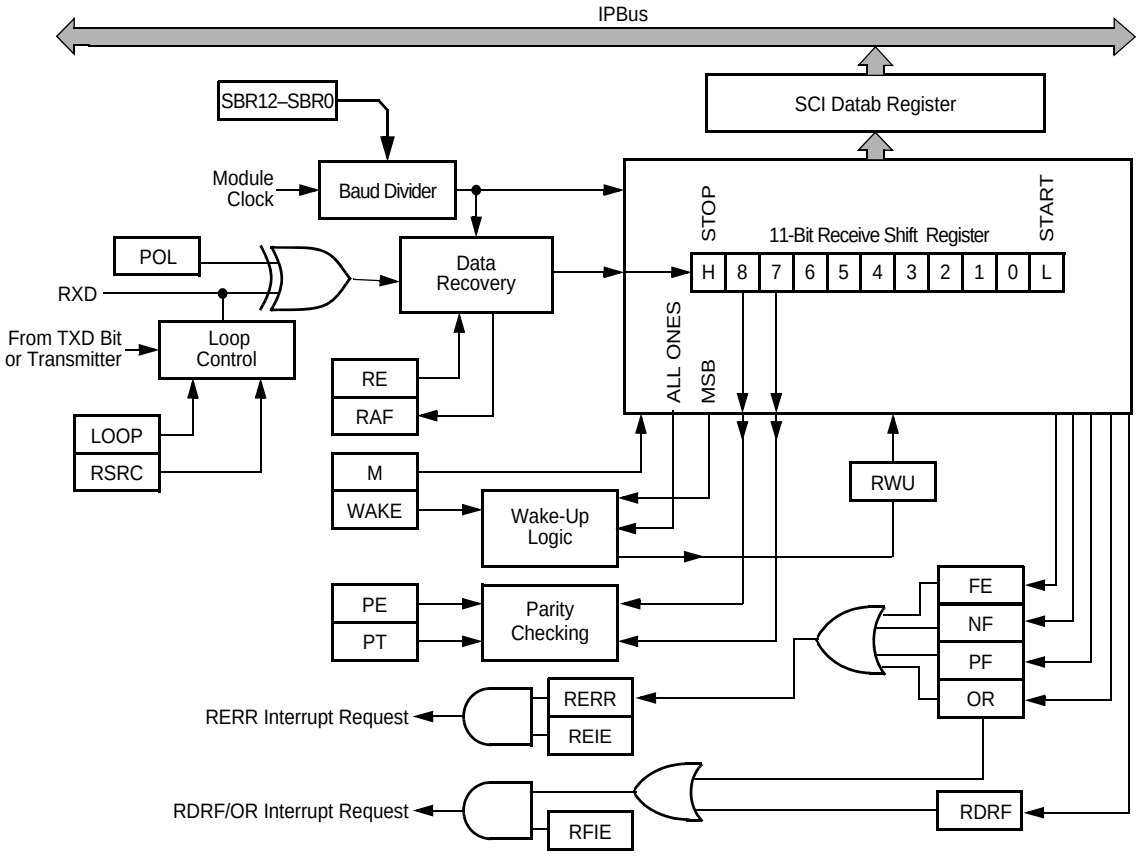


Figure 12-4. SCI Receiver Block Diagram

12.4.4.1 Character Length

The SCI receiver can accommodate either 8- or 9-bit data characters determined by the state of the M bit in the SCICR.

12.4.4.2 Character Reception

During an SCI reception, the Receive Shift register alters a frame in from the RXD pin. The data is read from the SCIDR.

After a complete frame shifts into the Receive Shift register, the data portion of the frame transfers to the SCIDR. The Receive Data Register Full (RDRF) flag in the SCISR is set, indicating the received character can be read. If the Receive Full Interrupt Enable (RFIE) bit in the SCICR is also set, the RDRF flag generates an RDRF interrupt request.

12.4.4.3 Data Sampling

The receiver samples the RXD pin at the Rate Tolerance (RT) clock rate. To adjust the baud rate mismatch, the RT clock illustrated in **Figure 12-5,** is resynchronized:

- After every START bit
- After the receiver detects a data bit change from Logic 1 to 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.

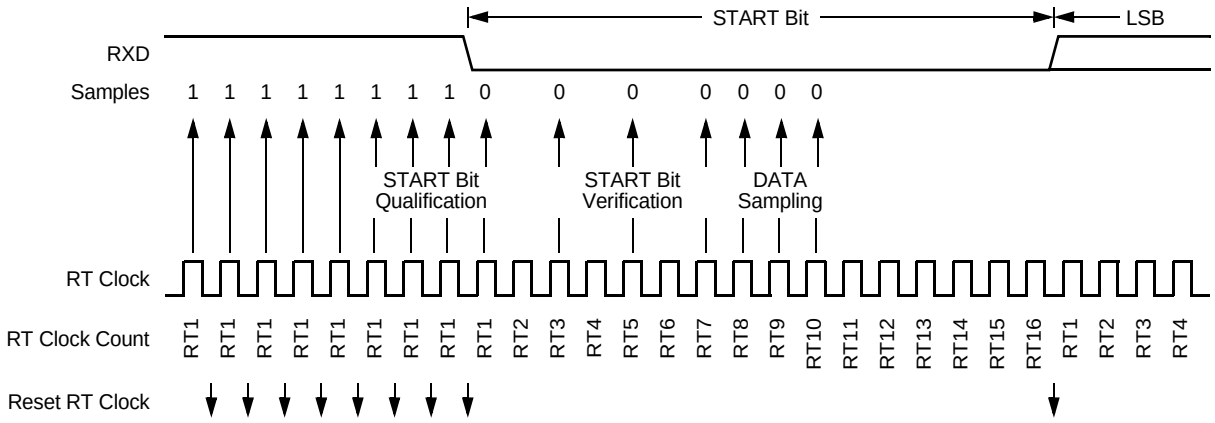


Figure 12-5. Receiver Data Sampling

To verify the START bit and detect noise, Data Recovery Logic takes samples at RT3, RT5, and RT7. **Table 12-4** summarizes the results of the START bit verification samples and the Noise Flag (NF). A majority vote of the three samples is used as the value of the bit.

Table 12-4. 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 two of three samples are 0s (not all 0s), Noise Flag (NF) is set. 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, Data Recovery Logic takes samples at RT8, RT9, and RT10. A majority vote of the three samples is used as the value of the bit. **Table 12-5** summarizes the results of the DATA bit samples. If all three samples are not the same, Noise Flag (NF) is set.

Table 12-5. 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 NF is set and the receiver assumes the bit is a START bit (Logic 0).

To verify a STOP bit and to detect noise, data recovery logic takes samples at RT8, RT9, and RT10. A majority vote of the three samples is used as the value of the bit. **Table 12-6** summarizes the results of the STOP bit samples. If all three samples are not the same, Noise Flag (NF) is set. If STOP bit detecting fails, Framing Error Flag (FE) is set.

Table 12-6. 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

12.4.4.4 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 (FE) flag in SCISR. Reception of a break character also sets the FE flag because a break character has no STOP bit. The FE flag is set concurrently with the RDRF flag. The FE flag inhibits further data reception until it is cleared. The FE flag is cleared by reading the SCISR, thereafter write any value to the SCISR.

12.4.4.5 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 either Noise Error, Framing Error, or both.

As the receiver samples an incoming frame, it resynchronizes the RT clock on any valid falling edge within the frame. Resynchronization within frames may correct misalignments between transmitter bit times and receiver bit times. The worst case is the data is all 0s or 1s without resynchronization occuring during the frame transmit.

12.4.4.5.1 Slow Data Tolerance

Figure 12-6 illustrates how much a slow received frame can be misaligned without causing a noise or framing error. The slow STOP bit begins at RT8 instead of RT1, but it arrives in time for the STOP bit data samples at RT8, RT9, and RT10.

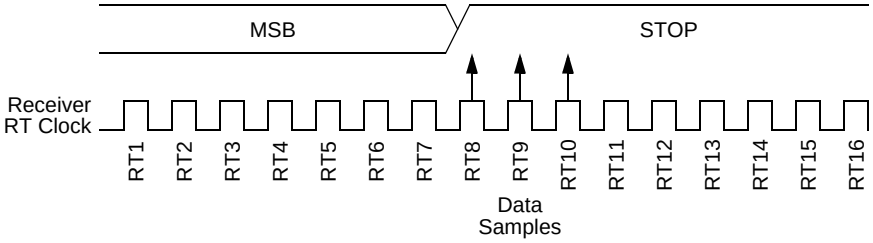


Figure 12-6. Slow Data

For an 8-bit (all 0s) data character, data sampling of the STOP bit takes the receiver:

$$9\text{-bit} \times 16 \text{ RT cycles} + 10 \text{ RT cycles} = 154 \text{ RT cycles}$$

With the misaligned character, illustrated in **Figure 12-6,** the receiver counts 154 RT cycles at the point when the count of the transmitting device is:

$$9\text{-bit} \times 16 \text{ RT cycles} + 3 \text{ RT cycles} = 147 \text{ RT cycles}$$

The maximum percentage difference between the receiver count and the transmitter count of a slow 8-bit data character with no errors is:

$$\left|\frac{154 - 147}{154}\right| \times 100 = 4.54\%$$

For a 9-bit (all 0s) data character, data sampling of the STOP bit takes the receiver:

10-bit × 16 RT cycles + 10 RT cycles = 170 RT cycles

With the misaligned character illustrated in **Figure 12-6,** the receiver counts 170 RT cycles at the point when the count of the transmitting device is:

10-bit × 16 RT cycles + 3 RT cycles = 163 RT cycles

The maximum percentage difference between the receiver count and the transmitter count of a slow 9-bit character with no errors is:

$$\left|\frac{170 - 163}{170}\right| \times 100 = 4.12\%$$

12.4.4.5.2 Fast Data Tolerance

Figure 12-7 illustrates how much a fast received frame can be misaligned without causing a noise error or a framing error. The fast STOP bit ends at RT10 instead of RT16 but it is still sampled at RT8, RT9, and RT10.

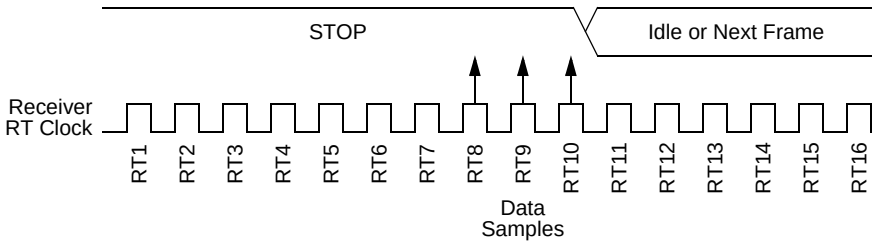


Figure 12-7. Fast Data

For an 8-bit (all 0s or 1s) data character, data sampling of the STOP bit takes the receiver:

9-bit × 16 RT cycles + 10 RT cycles = 154 RT cycles

With the misaligned character, illustrated in **Figure 12-7,** the receiver counts 154 RT cycles at the point when the count of the transmitting device is:

10-bit × 16 RT cycles = 160 RT cycles

The maximum percentage difference between the receiver count and the transmitter count of a fast 8-bit character with no errors is:

$$\left|\frac{154 - 160}{154}\right| \times 100 = 3.90\%$$

For a 9-bit (all 0s or 1s) data character, data sampling of the STOP bit takes the receiver:

$$\text{10-bit} \times \text{16 RT cycles} + \text{10 RT cycles} = \text{170 RT cycles}$$

With the misaligned character, illustrated in **Figure 12-7,** the receiver counts 170 RT cycles at the point when the count of the transmitting device is:

$$\text{11-bit} \times \text{16 RT cycles} = \text{176 RT cycles}$$

The maximum percentage difference between the receiver count and the transmitter count of a fast 9-bit character with no errors is:

$$\left|\frac{170 - 176}{170}\right| \times 100 = 3.53\%$$

12.4.4.6 Receiver Wake-Up

In order for the SCI to ignore transmissions intended only for other receivers in multiple receiver systems, the receiver can be put into a standby state. Setting the Receiver Wake-Up (RWU) bit in the SCICR puts the receiver into a standby state while receiver interrupts are disabled.

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 the SCICR determines how the SCI is brought out of the standby state to process an incoming message. The WAKE bit enables either Idle Input Line Wake-Up or Address Mark Wake-Up:

- Idle Input Line Wake-Up (WAKE = 0)—In this wake-up method, an idle condition on the RXD pin clears the RWU bit, waking up the SCI. Idle Input Line Wake-Up requires messages be separated by at least one preamble, and no message contains preambles. The initial frame or frames of every message contains addressing information. All receivers evaluate the addressing information. Receivers of the message then process the following frames. Any receiver a message does not address can set its RWU bit, returning to the standby state. The RWU bit remains set and the receiver remains on standby until another preamble appears on the RXD pin.

  The receiver-waking preamble does not set the Receiver Idle (RIDLE) bit or the Receive Data Full Register (RDRF) flag.

With the WAKE bit clear, setting the RWU bit after the RXD pin has been idle can cause the receiver to wake-up immediately.

- Address Mark Wake-Up (WAKE = 1)—In this wake-up method, a Logic 1 in the MSB position of a frame clears the RWU bit, awakening the SCI. The Logic 1 in the MSB position marks a frame as an address frame containing addressing information. The address frame also sets the RDRF bit in the SCISR. All receivers evaluate the addressing information. Receivers of the message then process the following frames. Any receiver a message does not address can set its RWU bit, returning to the standby state. The RWU bit remains set and the receiver remains on standby until another address frame appears on the RXD pin. Address Mark Wake-Up allows messages to contain preambles but it requires the MSB to be reserved for use in address frames.

12.5 Special Operating Modes

Table 12-7 summarizes how to configure for Normal Operation, Loop Back, or Single-Wire Operation as described in **Section 12.5.1, "Single-Wire Operation"** and **Section 12.5.2, "Loop Operation".**

Table 12-7. Loop Functions

LOOP	RSRC	Function
0	X	Normal Operation
1	0	Loop Mode with Internal TXD Fed Back to RXD
1	1	Single-Wire Mode with TXD Output Fed Back to RXD

12.5.1 Single-Wire Operation

Normally, the SCI uses two pins for transmitting and receiving. In the Single-Wire Operation, the RXD pin is disconnected from the SCI and is available for other peripherals, illustrated in **Figure 12-8.** The SCI uses the TXD pin for both receiving and transmitting.

Setting the TE bit in the SCICR enables the transmitter, configuring TXD as the output for transmitted data. Clearing the TE bit disables the transmitter, configuring TXD as the input for received data.

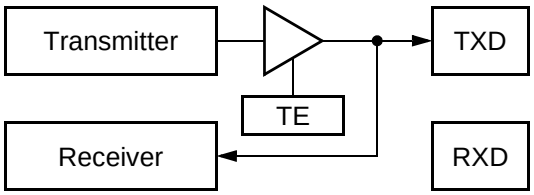


Figure 12-8. Single-Wire Operation (LOOP = 1, RSRC = 1)

Enable Single-Wire Operation by setting the LOOP bit and the Receiver Source (RSRC) bit in the SCICR. Setting the LOOP bit disables the path from the RXD pin to the receiver. Setting the RSRC bit connects the receiver input to the output of the TXD pin driver. To enable Receiver, Receiver Enable (RE) bit in the SCICR must be set.

12.5.2 Loop Operation

In Loop Operation, the transmitter output goes to the receiver input. The RXD pin is disconnected from the SCI and is available as a GPIO pin. Please see **Figure 12-9.**

Setting the TE bit in the SCICR enables the transmitter, connecting the transmitter output to the TXD pin. Clearing the TE bit disables the transmitter, disconnecting the transmitter output from the TXD pin.

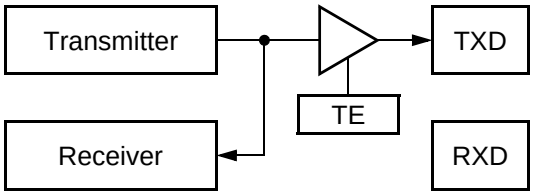


Figure 12-9. Loop Operation (LOOP = 1, RSRC = 0)

Enable Loop Operation by setting the LOOP bit and clearing the RSRC bit in the SCICR. Setting the LOOP bit disables the path from the RXD pin to the receiver. Clearing the RSRC bit connects the transmitter output to the receiver input. To enable Loop Operation both the Transmitter Enable (TE) and Receiver Enable (RE) bits in the SCICR must be set.

12.5.3 Low-Power Options

12.5.3.1 Run Mode

Clearing the Transmitter Enable (TE) or Receiver Enable (RE) bits in the SCICR reduces power consumption in Run mode. SCI registers are still accessible when TE or RE bits are cleared, but clocks to the SCI module are disabled.

12.5.3.2 Wait Mode

SCI operation in Wait mode depends on the state of the SWAI bit in the SCICR.

- If SWAI is clear, SCI operates normally when CPU is in Wait mode.
- If SWAI is set, SCI clock generation ceases and the SCI module enters a power conservation state when the CPU is in Wait mode. Setting SWAI does not affect the state of the RE bit or the TE bit.

When SWAI is set, any transmission or reception in progress stops at Wait mode entry. The transmission or reception resumes when either an internal or external interrupt brings the processor out of Wait mode.

When SWAI is set the SCI module cannot generate interrupt requests during Wait mode.

Any enabled SCI interrupt request can bring the processor out of Wait mode as long as SWAI is clear.

12.5.3.3 Stop Mode

The SCI is inactive in STOP mode for reduced power consumption. The Stop instruction does not affect the register states. SCI operation resumes after an external interrupt brings the processor out of Stop mode.

12.6 Register Definitions

Table 12-1. SCI Memory Map

Device	Name	Address
801/802/ 803/805	SCI0_BASE	$0F00
	SCI1_BASE	$0F10
807	SCI0_BASE	$1300
	SCI1_BASE	$1310

There are four accessible registers on SCI described in **Table 12-2** and summarized in **Table 12-3.** A register address is the sum of a base address and an address offset. The base address is defined

at the system level and the address offset is defined at the module level. The SCI has four registers.

Table 12-2. SCI Register Summary

Address Offset	Register Acronym	Register Description	Access Type	Chapter Location
Base + $0	SCIBR	Baud Rate Register	Read/Write	Section 12.6.1
Base + $1	SCICR	Control Register	Read/Write	Section 12.6.2
Base + $2	SCISR	Status Register	*Read-Only*	Section 12.6.3
Base + $3	SCIDR	Data Register	Read/Write	Section 12.6.4

Bit fields of each of the four registers are illustrated in **Figure 12-10.** Details of each follow.

Add. Offset	Register Name		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$0	SCIBR	R	0	0	0	SBR												
		W																
$1	SCICR	R	LOOP	SWAI	RSRC	M	WAKE	POL	PE	PT	TEIE	TIIE	RIE	REIE	TE	RE	RWU	SBK
		W																
$2	SCISR	R	TDRE	TIDLE	RDRF	RIDLE	OR	NF	FE	PF	0	0	0	0	0	0	0	RAF
		W																
$3	SCIDR	R	0	0	0	0	0	0	0	RECEIVE DATA								
		W								TRANSMIT DATA								

R 0 = Read as 0
W = Reserved

Figure 12-10. SCI Register Map

12.6.1 SCI Baud Rate Register (SCIBR)

This register can be read at any time. Bits 12 through 0 can be written at any time, but bits 15 through 13 are reserved.

Base + $0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	SBR												
Write																
RESET	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0

Figure 12-11. SCI Baud Rate Register (SCIBR)

See Table A-8, List of Programmer's Sheets

The count in this register determines the baud rate of the SCI. The formula for calculating baud rate is:

$$\text{SCI Baud Rate} = \frac{\text{IPBus Clock}}{16 \times \text{SBR}}$$

See **Section 12.4.2, "Baud Rate Generation"** for more details and examples.

Note: The baud rate generator is disabled until the TE or the RE bits are set for the first time after reset. The baud rate generator is disabled when SBR = 0.

12.6.1.1 Reserved—Bits 15–13

This bit field is reserved or not implemented. It is read as 0 and cannot be modified by writing.

12.6.1.2 SCI Baud Rate (SBR)—Bits 12–0

Contents of the Baud Rate register has a value of 1 to 8191.

12.6.2 SCI Control Register (SCICR)

The SCI Control Register (SCICR) can be read and written at anytime.

Base + $1	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read / Write	LOOP	SWAI	RSRC	M	WAKE	POL	PE	PT	TEIE	TIIE	RIE	REIE	TE	RE	RWU	SBK
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 12-12. SCI Control Register (SCICR)

See Table A-8, List of Programmer's Sheets

12.6.2.1 Loop Select Bit (LOOP)—Bit 15

This bit enables loop operation. Please see **Section 12.5.2, "Loop Operation".** The loop operation disconnects the RXD pin from the SCI and the transmitter output goes into the receiver input.

- 0 = Normal operation enabled
- 1 = Loop operation enabled

12.6.2.2 Stop in Wait Mode (SWAI)—Bit 14

The SWAI bit disables the SCI in Wait mode. See**Section 12.5.3.2, "Wait Mode".**

- 0 = SCI enabled in Wait mode

- 1 = SCI disabled in Wait mode

12.6.2.3 Receiver Source (RSRC)— Bit 13

When LOOP = 1, the RSRC bit determines the internal feedback path for the receiver. See **Section 12.5.1, "Single-Wire Operation"** and **Section 12.5.2, "Loop Operation"** for more details.

- 0 = Receiver input internally connected to transmitter output
- 1 = Receiver input connected to TXD pin

12.6.2.4 Data Format Mode (M)—Bit 12

This bit 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

12.6.2.5 Wake-Up Condition (WAKE)—Bit 11

This bit determines which condition wakes up the SCI.

- 0 = Idle Line Wake-Up
- 1 = Address Mark Wake-Up (a Logic 1 in the MSB position of a receive data character)

Note: Address Mark Wake-Up is not a valid option when the parity function is enabled (PE = 1) since the ADDRESS bit and the PARITY bit both occupy the MSB.

12.6.2.6 Polarity (POL)—Bit 10

This bit determines whether to invert the data as it goes from the transmitter to the TXD pin and from the RXD pin to the receiver. All bits, START, DATA, and STOP, are inverted as they leave the Transmit Shift register and before they enter the Receive Shift register.

- 0 = Doesn't invert transmit and receive data bits (Normal mode)
- 1 = Invert transmit and receive data bits (Inverted mode)

Note: It is recommended the POL bit be toggled only when both TE and RE = 0.

12.6.2.7 Parity Enable (PE)—Bit 9

This bit enables the parity function. When enabled, the parity function replaces the MSB of the data character with a parity bit.

- 0 = Parity function disabled
- 1 = Parity function enabled

Note: Address Mark Wake-Up (WAKE = 1) is not a valid option when the parity function is enabled because the ADDRESS bit and the PARITY bit both occupy the MSB.

12.6.2.8 Parity Type (PT)—Bit 8

This bit determines whether the SCI generates and checks for even parity or odd parity of the data bits. With *even parity*, an *even* number of *ones, clears* the PARITY bit, while an *odd* number of *ones, sets* the PARITY bit. However, with *odd parity*, an *odd* number of *ones, clears* the PARITY bit, while an *even* number of *ones, sets* the PARITY bit.

- 0 = Even parity
- 1 = Odd parity

12.6.2.9 Transmitter Empty Interrupt Enable (TEIE)—Bit 7

This bit enables the TDRE flag to generate interrupt requests.

- 0 = TDRE interrupt requests disabled
- 1 = TDRE interrupt requests enabled

12.6.2.10 Transmitter Idle Interrupt Enable (TIIE)—Bit 6

This bit enables the TIDLE flag to generate interrupt requests.

- 0 = TIDLE interrupt requests disabled
- 1 = TIDLE interrupt requests enabled

12.6.2.11 Receiver Full Interrupt Enable (RIE)—Bit 5

This bit enables the RDRF flag, or the OR flag to generate interrupt requests.

- 0 = RDRF and OR interrupt requests disabled
- 1 = RDRF and OR interrupt requests enabled

12.6.2.12 Receive Error Interrupt Enable (REIE)—Bit 4

This bit enables the receive error flags (NF, PF, FE, and OR) to generate interrupt requests. The status bits can be checked during the error interrupt process.

- 0 = Error interrupt requests disabled
- 1 = Error interrupt requests enabled

12.6.2.13 Transmitter Enable (TE)—Bit 3

This bit enables the SCI transmitter and configures the TXD pin as the SCI transmitter output. The TE bit can be used to queue an idle preamble.

- 0 = Transmitter disabled
- 1 = Transmitter enabled

12.6.2.14 Receiver Enable (RE)—Bit 2

This bit enables the SCI Receiver.

- 0 = Receiver disabled
- 1 = Receiver enabled

12.6.2.15 Receiver Wake-Up (RWU)—Bit 1

This bit enables the wake-up function, inhibiting further receiver interrupt requests. Normally, hardware wakes the receiver by automatically clearing RWU. Please refer to **Section 12.4.4.6, "Receiver Wake-Up"** for a description of Receive Wake-Up.

- 0 = Normal operation
- 1 = Standby state

12.6.2.16 Send Break (SBK)—Bit 0

Setting SBK sends one break character (all Logic 0s, include START, DATA, STOP). As long as SBK is set, transmitter sends uninterrupted break characters.

- 0 = No break characters
- 1 = Transmit break characters

12.6.3 SCI Status Register (SCISR)

This register can be read at anytime; however, it cannot be modified by writing. Writes clear flags.

Base + $2	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	TDRE	TIDLE	RDRF	RIDLE	OR	NF	FE	PF	0	0	0	0	0	0	0	RAF
Write																
RESET	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 12-13. SCI Status Register (SCISR)

See Table A-8, List of Programmer's Sheets

12.6.3.1 Transmit Data Register Empty Flag (TDRE)—Bit 15

This bit is set when the Transmit Shift register receives a character from the SCIDR. Clear TDRE by reading SCISR, then write to the SCIDR.

- 0 = No character transferred to Transmit Shift register
- 1 = Character transferred to Transmit Shift register; TDRE

12.6.3.2 Transmitter Idle Flag (TIDLE)—Bit 14

This bit is set when the TDRE flag is set and no data, preamble, or break character is being transmitted. When TIDLE is set, the TXD pin becomes idle (Logic 1). Clear TIDLE by reading the SCISR, then write to the SCIDR.

- 0 = Transmission in progress
- 1 = No transmission in progress

12.6.3.3 Receive Data Register Full Flag (RDRF)—Bit 13

This bit is set when the data in the Receive Shift register transfers to the SCIDR. Clear RDRF by reading the SCISR, then read the SCIDR.

- 0 = Data not available in SCIDR
- 1 = Received data available in SCIDR

12.6.3.4 Receiver Idle Line Flag (RIDLE)—Bit 12

This bit is set when ten consecutive Logic 1s (if M = 0) or eleven consecutive Logic 1s (if M = 1) appear on the receiver input. Once the RIDLE flag is cleared by the receiver detecting a Logic 0, a valid frame must again set the RDRF flag before an idle condition can set the RIDLE flag.

- 0 = Receiver input is either active now or has never become active since the RIDLE flag was last cleared by reset
- 1 = Receiver input has become idle (after receiving a valid frame)

Note: When the Receiver Wake-Up (RWU) bit is set, an idle line condition does not set the RIDLE flag.

12.6.3.5 Overrun Flag (OR)—Bit 11

This bit is set when software fails to read the SCIDR before the Receive Shift register receives the next frame. The data in the Shift register is lost, but the data already in the SCIDR is not affected. Clear OR by reading the SCISR, then write the SCISR with any value.

- 0 = No overrun

- 1 = Overrun

12.6.3.6 Noise Flag (NF)—Bit 10

This bit is set when the SCI detects noise on the receiver input. The NF bit is set during the same cycle as the RDRF flag, but it is not set in the case of an overrun. Clear NF by reading the SCISR, then write the SCISR with any value.

- 0 = No noise
- 1 = Noise

12.6.3.7 Framing Error Flag (FE)—Bit 9

This bit is set when a Logic 0 is accepted as the STOP bit. The FE bit is set during the same cycle as the RDRF flag but it is not set in the case of an overrun. FE inhibits further data reception until it is cleared. Clear FE by reading the SCISR, then write the SCISR with any value.

- 0 = No framing error
- 1 = Framing error

12.6.3.8 Parity Error Flag (PF)—Bit 8

This bit is set when the Parity Enable (PE) bit is set and the parity of the received data does not match its parity bit. Clear PF by reading the SCISR, then write the SCISR with any value.

- 0 = No parity error
- 1 = Parity error

12.6.3.9 Reserved—Bits 7–1

These bits are reserved or not implemented. They are read as zero and cannot be modified by writing.

12.6.3.10 Receiver Active Flag (RAF)—Bit 0

This bit 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 false start bits (usually from noise or baud rate mismatch) or when the receiver detects a preamble.

- 0 = No reception in progress
- 1 = Reception in progress

12.6.4 SCI Data Register (SCIDR)

The SCIDR can be read and modified at any time. Reading accesses the SCI Receive Data register. Writing to the register accesses the SCI Transmit Data register.

Base + $3	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	0	0	RECEIVE DATA								
Write								TRANSMIT DATA								
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 12-14. SCI Data Register (SCIDR)

See Table A-8, List of Programmer's Sheets

12.6.4.1 Reserved—Bits 15–9

These bits are reserved or not implemented. They are read as zero and cannot be modified by writing.

12.6.4.2 Receive/Transmit Data—Bits 8–0

Writing to these bits loads the transmit data. Reading these bits accesses the receive data.

Note: When configured for 8-bit data, bits 7 to 0 contain the data received.

12.7 Clocks

All timing is derived from the IPBus clock, operating at the system clock rate. Please see **Section 12.4.2, "Baud Rate Generation"** for a description of how the data rate is determined.

12.8 Resets

Any system reset completely resets the SCI.

12.9 Interrupts

Table 12-3. SCI Interrupt Sources

Interrupt Source	Flag	Local Enable	Description
Transmitter	TDRE	TEIE	Transmit Empty
	TIDLE	TIIE	Transmit Idle
Receiver	RDRF	RFIE	Receive Full
	OR		
	FE	REIE	Receive Error
	PE		
	NF		
	OR		

12.9.1 Transmitter Empty Interrupt

This interrupt is enabled by setting the TEIE bit of the SCICR. When this interrupt is enabled, an interrupt is generated when data is transferred from the SCIDR to the Transmit Shift register.

12.9.2 Transmitter Idle Interrupt

This interrupt is enabled by setting the TIIE bit of the SCICR. This interrupt indicates the TIDLE flag is set and the transmitter is no longer sending data, preamble, or break characters. The interrupt service routine should initiate a preamble, a break, write a data character to the SCIDR or disable the transmitter.

12.9.3 Receiver Full Interrupt

This interrupt is enabled by setting the RFIE bit of the SCICR. This interrupt indicates either receive data is available in the SCIDR or a data overrun occurred. The interrupt service reoutine should read SCISR to determine which of RDRF flag, OR flag, or both were set.

12.9.4 Receive Error Interrupt

This interrupt is enabled by setting the REIE bit of the SCICR. This interrupt indicates any of the listed errors was detected by the receiver:

1. Noise Flag (NF) set
2. Parity Error Flag (PF) set

3. Framing Error (FE) flag set
4. Overrun (OR) flag set

The interrupt service routine should read the SCISR to determine which of the error flags was set. The error flag is cleared by writing anything to the SCISR. Then the appropriate action should be taken by the software to handle the error condition.

Table 12-4. Document Revision History for Chapter 12

Version History	Description of Change
Rev. 8	Formatting, layout, spelling, and grammar corrections. Added revision history table. Corrected the name of bit 5 in SCICR (was RFIE, is RIE).

Chapter 13
Serial Peripheral Interface (SPI)

13.1 Introduction

This chapter describes the Serial Peripheral Interface (SPI) module. The module allows full-duplex, synchronous, serial communication between the 16-bit controller and peripheral devices, including other 16-bit controllers.

13.2 Features

Characteristics of the SPI module include:

- Full-duplex operation
- Master and Slave modes
- Double-buffered operation with separate transmit and receive registers
- Programmable length transmissions (two to 16 bits)
- Programmable transmit and receive shift order (MSB first or last bit transmitted)
- Eight Master mode frequencies (maximum = module clock ÷ 2)
- Maximum Slave mode frequency = module clock
- Clock ground for reduced Radio Frequency (RF) interference
- Serial clock with programmable polarity and phase
- Two separately enabled interrupts
  - SPI Receiver Full (SPRF)
  - SPI Transmitter Empty (SPTE)
- Mode Fault Error flag interrupt capability

Note: Throughout this chapter, there are references to the SPI module clock. The SPI module clock is the IPBus clock.

13.3 Block Diagram

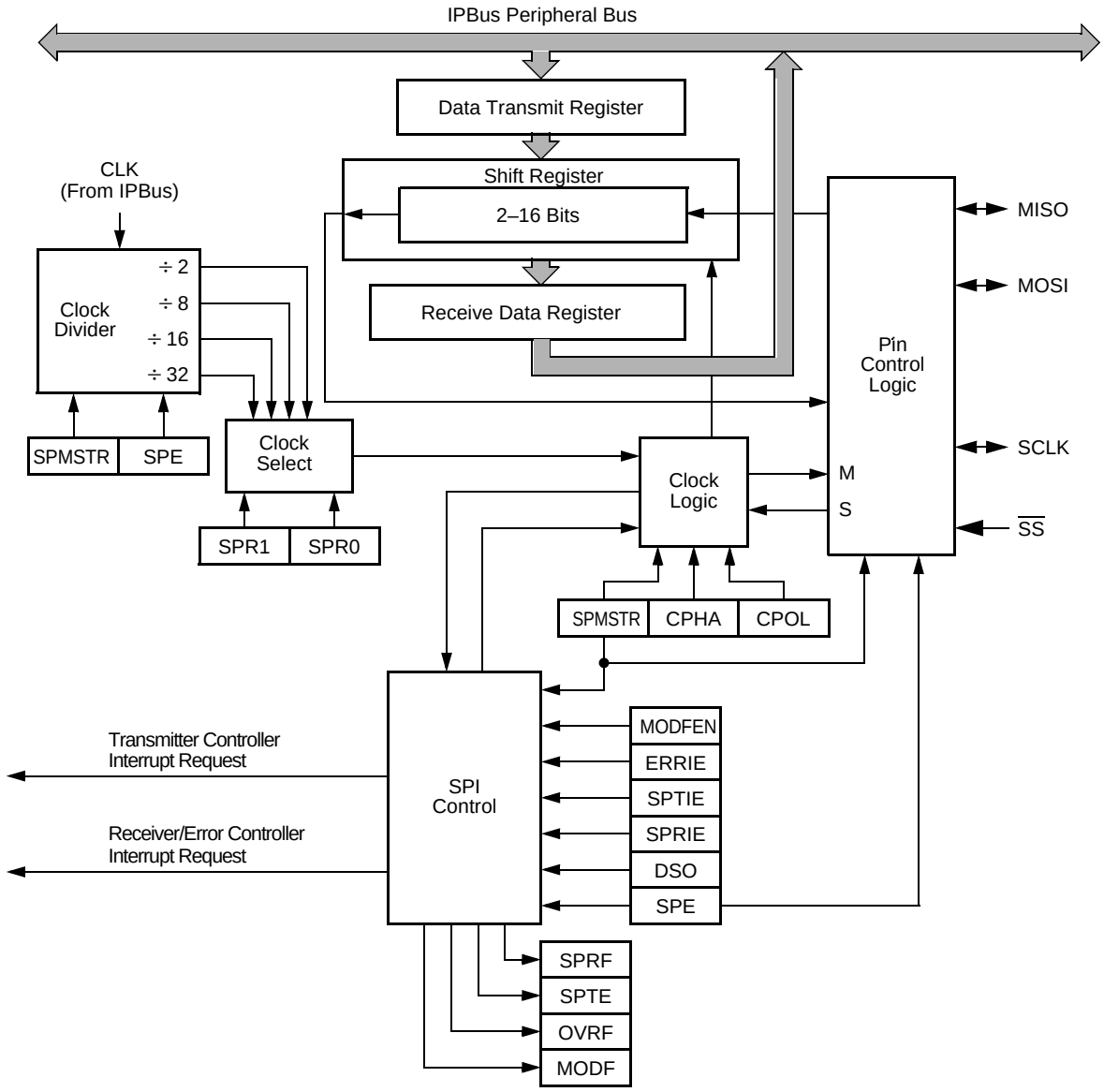


Figure 13-1. SPI Block Diagram

13.4 Operating Modes

The SPI has two operating modes:

- Master
- Slave

An operating mode is selected by the SPMSTR bit in the SPI Status and Control Register (SPSCR) as follows:

- SPMSTR = 0 Slave mode
- SPMSTR = 1 Master mode

Note: The SPMSTR bit should be configured before enabling the SPI (setting the SPE bit in the SPSCR). The master SPI should be enabled before enabling any slave SPI. All slave SPIs should be disabled before disabling the master SPI.

13.4.1 Master Mode

The SPI operates in Master mode when the SPI master bit, SPMSTR, is set. Only a master SPI module can initiate transmissions. With the SPI enabled, software begins the transmission from the master SPI module by writing to the SPI Data Transmit Register (SPDTR). If the Shift register is empty, the data immediately transfers to the Shift register, setting the SPI Transmitter Empty (SPTE) bit. The data begins shifting out on the Master Out/Slave In (MOSI) pin under the control of the SPI Serial Clock (SCLK).

The SPR[2:0] bits in the SPI Status and Control Register (SPSCR) control the Baud Rate Generator, determining the speed of the Shift register. The Baud Rate Generator of the master also controls the Shift register of the slave peripheral via the SCLK pin.

As the data shifts out on the MOSI pin of the master, external data shifts in from the slave on the Master In/Slave Out (MISO) pin. The transmission ends when the SPI Receiver Full (SPRF) bit in the SPSCR becomes set. At the same time the SPRF becomes set, the data from the slave transfers to the SPI Data Receive Register (SPDRR). In a normal operation, SPRF signals the end of a transmission. Software clears the SPRF by reading the SPSCR with SPRF set and then reading the SPDRR. Writing to the SPDTR clears the SPTE bit.

Figure 13-2 is an example configuration for a Full-Duplex Master-Slave Configuration. Having the Slave Select ($\overline{SS}$) bit of the master 16-bit controller held high is only necessary if MODFEN = 1. Tying the Slave 16-bit controller $\overline{SS}$ bit to ground should only be executed if CPHA = 1.

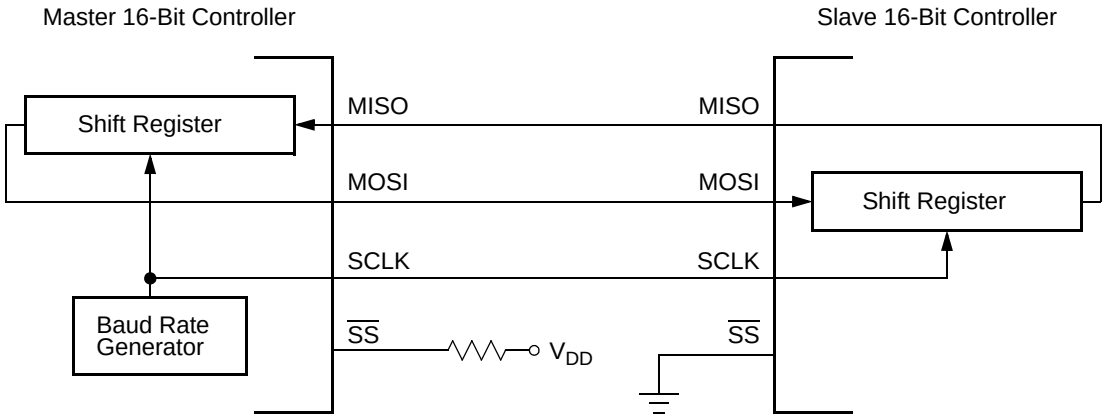


Figure 13-2. Full Duplex Master/Slave Connections

13.4.2 Slave Mode

The SPI operates in Slave mode when the SPMSTR bit is cleared. While in Slave mode, the SCLK pin acts as the input for the serial clock from the master 16-bit controller. Before a data transmission occurs, the $\overline{SS}$ pin of the slave SPI must be at Logic 0. $\overline{SS}$ must remain low until the transmission is complete or a Mode Fault Error occurs.

Note: The SPI must be enabled (SPE = 1) for slave transmissions to be received.

Note: Data in the transmitter Shift register will be unaffected by SCLK transitions in the event the SPI is operating as a slave but is deselected.

In a slave SPI module, data enters the Shift register under the control of the Serial Clock, (SCLK), from the master SPI module. After a full length data transmission enters the Shift register of a slave SPI, it transfers to the SPI Data Receive Register (SPDDR) and the SPI Receiver Full (SPRF) bit in the SPSCR is set. If the SPI Receive Interrupt Enable (SPRIE) bit in the SPSCR is set, a Receive Interrupt is also generated. To prevent an overflow condition, slave software must read the SPDRR before another full length data transmission enters the Shift register.

The maximum frequency of the SCLK for the SPI configured as a slave is the module clock ÷ 2, or half the module clock. Frequency of the SCLK for the SPI configured as a slave does not have to correspond to any particular SPI baud rate. The baud rate only controls the speed of the SCLK generated by the SPI configured as a master. Therefore, the frequency of the SCLK for a SPI configured as a slave can be any frequency less than or equal to half of the module clock.

When the master SPI starts a transmission, the data in the slave Shift register begins shifting out on the MISO pin. The slave can load its Shift register with new data for the next transmission by writing to its SPDTR. The slave must write to its SPDTR at least one bus cycle before the master starts the next transmission. Otherwise, the data already in the slave Shift register shifts out on

the MISO pin. Data written to the Slave Shift register during a transmission remains in a buffer until the end of the transmission.

When the CPHA bit is set, the first edge of SCLK starts a transmission. When CPHA is cleared, the falling edge of $\overline{SS}$ starts a transmission.

Note: SCLK must be in the proper idle state (depends on setting of CPOL bit) before the slave is enabled to prevent SCLK from appearing as a clock edge.

13.5 Pin Descriptions

There are four external SPI pins. Each is summarized in **Table 13-1.**

Table 13-1. External I/O Signals

Signal Name	Description	Direction
MISO	Master-In Slave-Out Pad Pin	Bi-Directional
MOSI	Master-Out Slave-In Pad Pin	Bi-Directional
SCLK	Serial Clock Pad Pin	Bi-Directional
$\overline{SS}$	Slave Select Pad Pin (Active Low)	Input

13.5.1 Master In/Slave Out (MISO)

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

Slave output data on the MISO pin is enabled only when the SPI is configured as a slave. The SPI is configured as a slave when the SPMSTR bit, discussed in **Section 13.9.1, "SPI Status and Control Register (SPSCR),"** is Logic 0 and its $\overline{SS}$ pin is at Logic 0. To support a multiple slave system, a Logic 1 on the $\overline{SS}$ pin puts the MISO pin in a High Impedance state.

13.5.2 Master Out/Slave In (MOSI)

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

13.5.3 Serial Clock (SCLK)

The serial clock synchronizes data transmission between master and slave devices. In a master 16-bit controller, the SCLK pin is the clock output. In a slave 16-bit controller, the SCLK pin is the clock input. In full duplex operation, the master and slave 16-bit controller exchange data in the same number of clock cycles as the number of bits of transmitted data.

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

The $\overline{SS}$ pin has various functions depending on the current state of the SPI. For a SPI configured as a slave, the $\overline{SS}$ is used to select a slave. When the Clock Phase (CPHA) bit in the SPSCR is cleared, the $\overline{SS}$ is used to define the start of a transmission, so it must be toggled high and low between each full length data transmitted for the CPHA = 0 format. However, it can remain low between transmissions for the CPHA = 1 format as illustrated in **Figure 13-4.**

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

Note: A Logic 1 voltage on the $\overline{SS}$ pin of a slave SPI puts the MISO pin in a high impedance state. The slave SPI ignores all incoming SCLK clocks, even if it was already in the middle of a transmission. A Mode Fault occurs if the $\overline{SS}$ pin changes state during a transmission.

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

Table 13-2. SPI I/O Configuration

SPE	SPMSTR	MODFEN	SPI Configuration	State of $\overline{SS}$ Logic
0	x	x	Not Enabled	$\overline{SS}$ ignored by SPI
1	0	x	Slave	Input-only to SPI
1	1	0	Master without MODF	$\overline{SS}$ ignored by SPI
1	1	1	Master with MODF	Input-only to SPI

x = Don’t care

13.6 Transmission Formats

During a SPI transmission, data is simultaneously transmitted (shifted out serially) and received (shifted in serially). A serial clock synchronizes shifting and sampling on the two serial data lines. A slave select line allows selection of an individual slave SPI device; slave devices not

selected do not interfere with SPI bus activities. On a master SPI device, the slave select line can optionally be used to indicate multiple-master bus contention.

13.6.1 Data Transmission Length

The SPI can support data lengths from two to 16 bits. This can be configured in the Data Size Register (SPDSR). When the data length is less than 16 bits, the Receive Data register will pad the upper bits with zeros.

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

13.6.2 Data Shift Ordering

The SPI can be configured to transmit or receive the MSB of the desired data first or last. This is controlled by the Data Shift Order (DSO) bit in the SPSCR. Regardless which bit is transmitted or received first, the data shall always be written to the SPDTR and read from the Receive Data Register (SPDRR) with the LSB in bit zero and the MSB in the correct position, depending on the data transmission size.

13.6.3 Clock Phase and Polarity Controls

Software can select any of four combinations of Serial Clock (SCLK) phase and polarity using two bits in the SPSCR. The Clock Polarity is specified by the (CPOL) control bit. In turn, it selects an active high or low clock and has no significant effect on the transmission format.

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

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

13.6.4 Transmission Format When CPHA = 0

Figure 13-3 exhibits a SPI transmission with CPHA as Logic 0.

Note: The figure should not be used as a replacement for data sheet parametric information.

Two waveforms for the SCLK are shown:

1. CPOL = 0
2. CPOL = 1

The diagram may be interpreted as a master or slave timing diagram since the Serial Clock (SCLK), Master In/Slave Out (MISO), and Master Out/Slave In (MOSI) pins are directly connected between the master and the slave. The MISO signal is the output from the slave, and the MOSI signal is the output from the master.

The $\overline{SS}$ line is the slave select input to the slave. The slave SPI drives its MISO output only when its slave select input ($\overline{SS}$) is at Logic 0, because only the selected slave drives to the master. The $\overline{SS}$ pin of the master is not shown, but it is assumed to be inactive. The $\overline{SS}$ pin of the master must be high or a Mode Fault Error will occur. When CPHA = 0, the first SCLK edge is the MSB capture strobe. Therefore, the slave must begin driving its data before the first SCLK edge and a falling edge on the $\overline{SS}$ pin is used to start the slave data transmission. The slave's $\overline{SS}$ pin must be toggled back to high and then low again between each full length data transmitted as depicted in **Figure 13-4.**

Note: **Figure 13-3** assumes 16-bit data lengths and the MSB shifted out first.

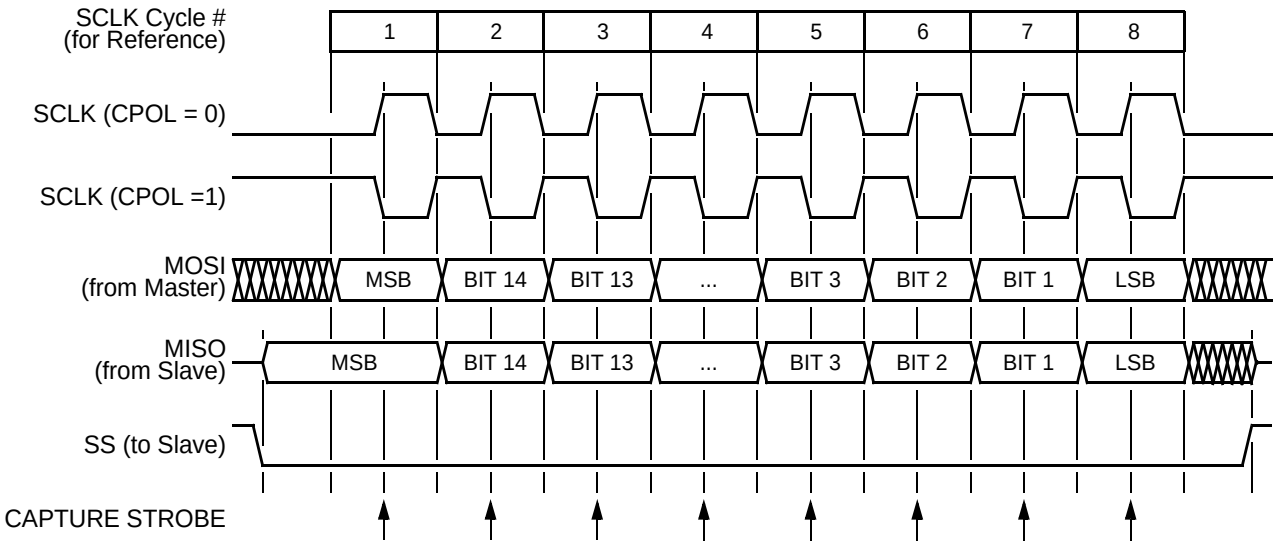


Figure 13-3. Transmission Format (CPHA = 0)

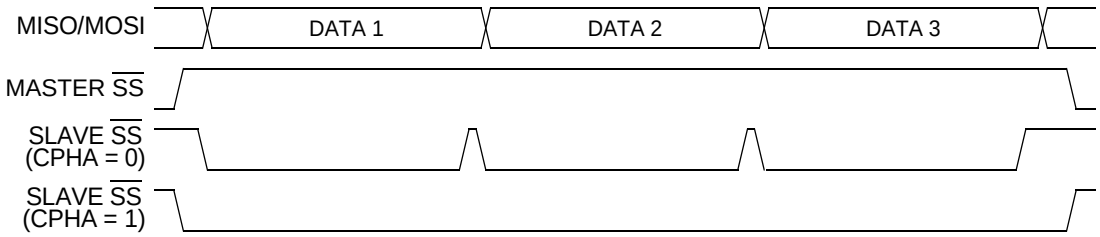


Figure 13-4. CPHA/SS Timing

When CPHA = 0 for a slave, the falling edge of SS indicates the beginning of the transmission. This causes the SPI to leave its idle state and begin driving the MISO pin with the first bit of its data. Once the transmission begins, no new data is allowed into the Shift register from the SPDTR. Therefore, the SPI Data register of the slave must be loaded with transmit data before the falling edge of SS. Any data written after the falling edge is stored in the SPDTR and transferred to the Shift register after the current transmission.

13.6.5 Transmission Format When CPHA = 1

A SPI transmission is shown in **Figure 13-5** where CPHA is Logic 1.

Note: The figure should not be used as a replacement for data sheet parametric information.

Two waveforms are shown for SCLK: 1 for CPOL = 0 and another for CPOL = 1. The diagram may be interpreted as a master or slave timing diagram since the serial clock (SCLK), Master In/Slave Out (MISO), and Master Out/Slave In (MOSI) pins are directly connected between the master and the slave. The MISO signal is the output from the slave, and the MOSI signal is the output from the master. The SS line is the slave select input to the slave. The slave SPI drives its MISO output only when its slave select input (SS) is at Logic 0, so only the selected slave drives to the master. The SS pin of the master is not shown but is assumed to be inactive. The SS pin of the master must be high or a Mode Fault Error occurs. When CPHA = 1, the master begins driving its MOSI pin on the first SCLK edge. Therefore, the slave uses the first SCLK edge as a start transmission signal. The SS pin can remain low between transmissions. This format may be preferable in systems having only one master and slave driving the MISO data line.

Note: **Figure 13-5** assumes 16-bit data lengths and the MSB shifted out first.

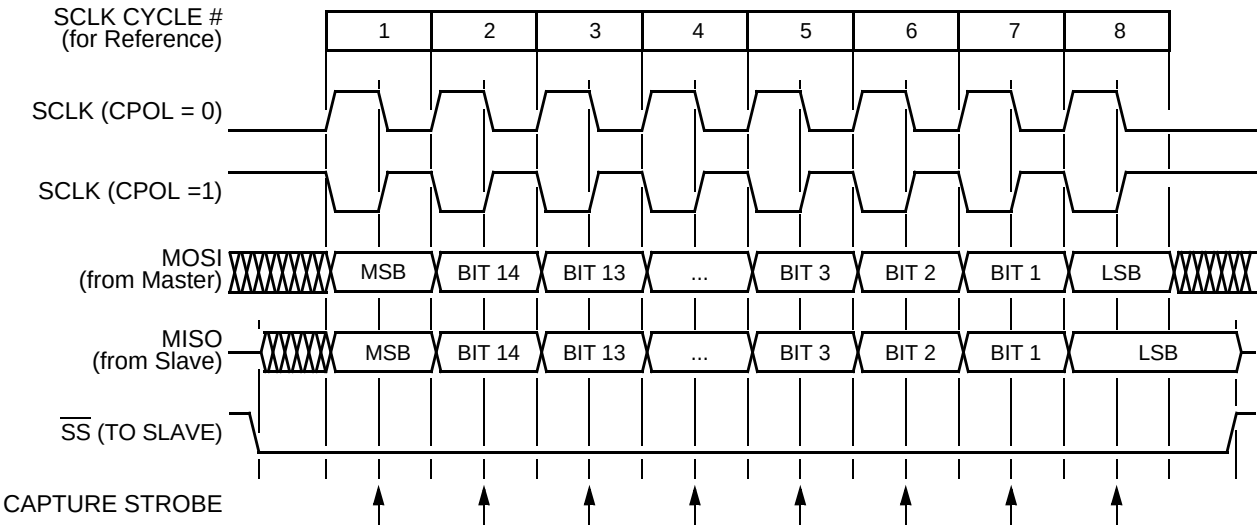


Figure 13-5. Transmission Format (CPHA = 1)

When CPHA = 1 for a slave, the first edge of the SCLK indicates the beginning of the transmission. This causes the SPI to leave its idle state and begin driving the MISO pin with the first bit of its data. Once the transmission begins, no new data is allowed into the Shift register from the SPDTR. Therefore, the SPI Data register of the slave must be loaded with transmit data before the first edge of SCLK. Any data written after the first edge is stored in the SPDTR and transferred to the Shift register after the current transmission.

13.6.6 Transmission Initiation Latency

When the SPI is configured as a master (SPMSTR = 1), writing to the SPDTR starts a transmission. CPHA has no effect on the delay to the start of the transmission, but it does affect the initial state of the SCLK signal. When CPHA = 0, the SCLK signal remains inactive for the first half of the first SCLK cycle. When CPHA = 1, the first SCLK cycle begins with an edge on the SCLK line from its inactive to its active level. The SPI clock rate, selected by SPR[2:0], affects the delay from the write to SPDTR and the start of the SPI transmission. The internal SPI clock in the master is a free-running derivative of the internal clock. To conserve power, it is enabled only when both the SPE and SPMSTR bits are set. Since the SPI clock is free-running, it is uncertain where the write to the SPDTR occurs relative to the slower SCLK. This uncertainty causes the variation in the initiation delay, demonstrated in **Figure 13-6.** This delay is no longer than a single SPI bit time. That is, the maximum delay is two bus cycles for DIV2, four bus cycles for DIV4, eight bus cycles for DIV8, and so on up to a maximum of 256 cycles for DIV256.

Note: **Figure 13-6** assumes 16-bit data lengths and the MSB shifted out first.

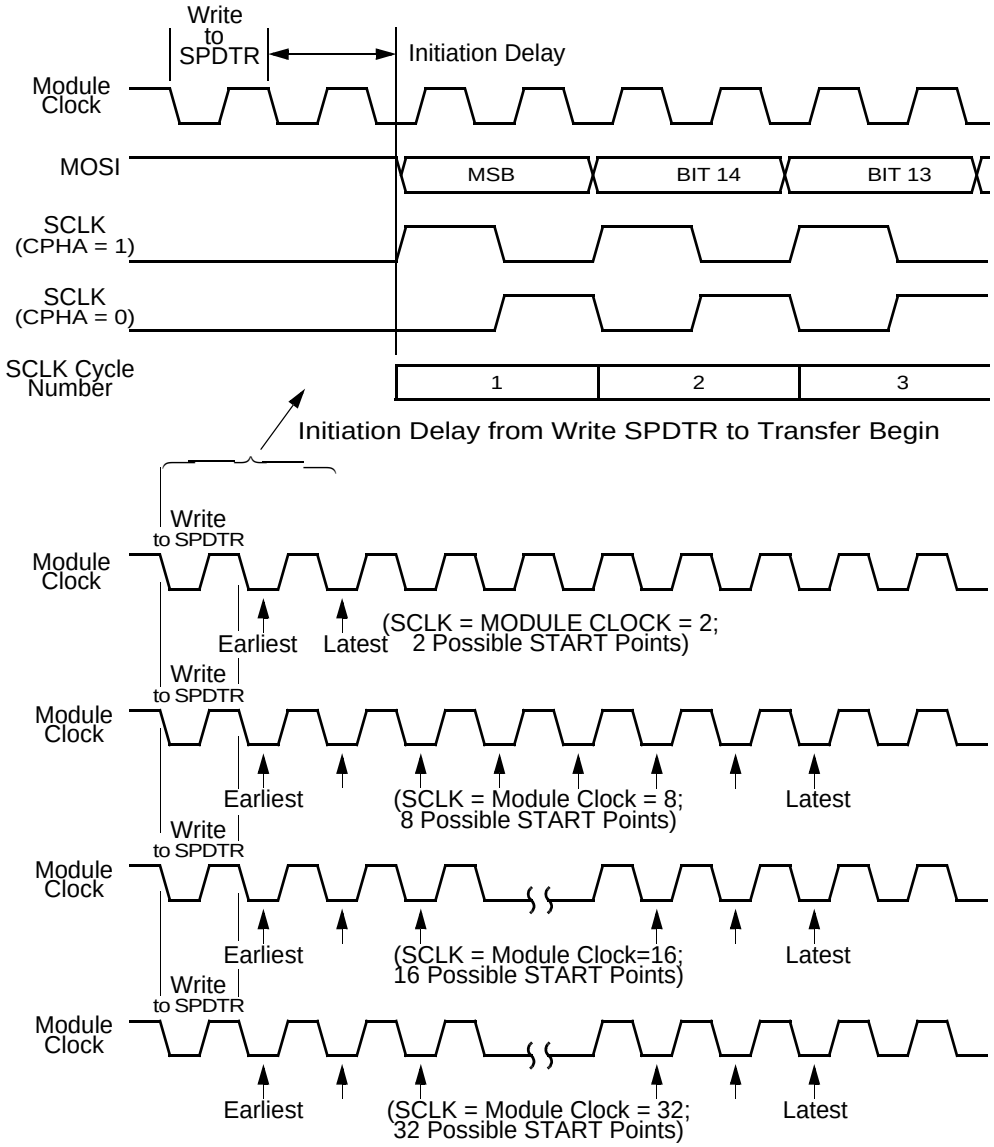
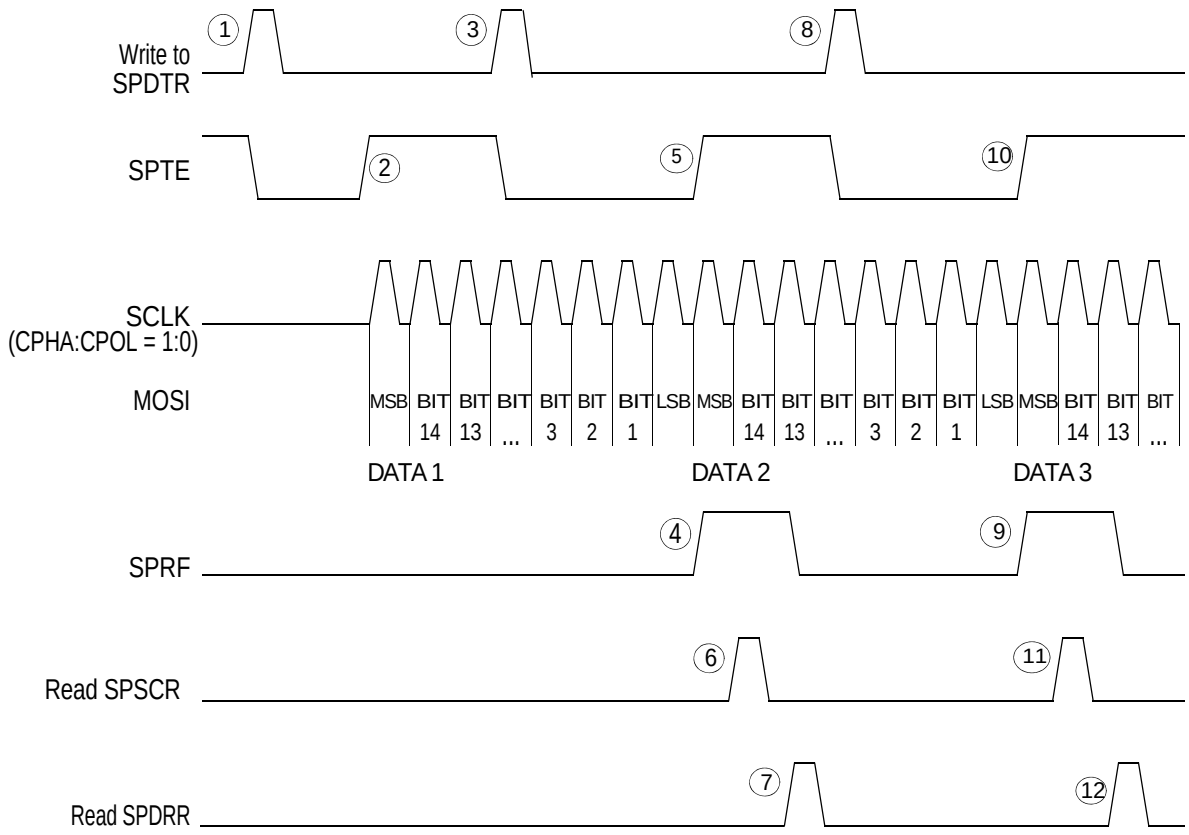


Figure 13-6. Transmission Start Delay (Master)

13.7 Transmission Data

The double-buffered SPDTR allows data to be queued and transmitted. For a SPI configured as a master, the queued data is transmitted immediately after the previous transmission has completed. The SPTE flag indicates when the transmit data buffer is ready to accept new data. Write to the SPDTR only when the SPTE bit is high. **Figure 13-7** illustrates the timing associated with doing back-to-back transmissions with the SPI (SCLK has CPHA: CPOL = 1:0).

Note: **Figure 13-7** assumes 16-bit data lengths and the MSB shifted out first.



1. 16-BIT CONTROLLER WRITES DATA 1 TO SPDTR, CLEARING THE SPTE BIT.
2. DATA 1 TRANSFERS FROM TRANSMIT DATA REGISTER TO SHIFT REGISTER, SETTING SPTE REGISTER TO SHIFT REGISTER, SETTING.
3. 16-BIT CONTROLLER WRITES DATA 2 TO SPDTR, QUEUEING DATA2 AND CLEARING SPTE BIT.
4. FIRST INCOMING WORD TRANSFERS FROM THE SHIFT REGISTER TO THE RECEIVE DATA REGISTER, SETTING THE SPRF BIT.
5. DATA 2 TRANSFERS FROM THE TRANSMIT DATA REGISTER TO SHIFT REGISTER, SETTING THE SPTE BIT.
6. 16-BIT CONTROLLER READS SPSCR WITH THE SPRF BIT SET.
7. 16-BIT CONTROLLER READS SPDRR, CLEARING SPRF BIT.
8. 16-BIT CONTROLLER WRITES DATA3 TO DTR, QUEUEING DATA3 AND CLEARING THE SPTE BIT.
9. SECOND INCOMING DATA TRANSFERS FROM SHIFT REGISTER TO RECEIVE DATA REGISTER SETTING THE SPTE BIT.
10. DATA3 TRANSFERS FROM THE TRANSMIT DATA REGISTER TO THE SHIFT REGISTER, SETTING THE SPTE BIT.
11. 16-BIT CONTROLLER READS SPSCR WITH THE SPRF BIT.
12. 16-BIT CONTROLLER READS SPDRR, CLEARING THE SPRF BIT.

Figure 13-7. SPRF/SPTE Interrupt Timing

The transmit data buffer permits back-to-back transmissions without the slave precisely timing its writes between transmissions as is necessary in a system with a single data buffer. Also, if no new data is written to the data buffer, the last value contained in the Shift register is the next data to be transmitted.

An idle master or idle slave without loaded data in its transmit buffer, sets the SPTE again no more than two bus cycles after the transmit buffer empties into the Shift register. This allows a queue to send up to a 32-bit value. For an already active slave, the load of the Shift register cannot occur until the transmission is completed. This implies a back-to-back write to the SPDTR is not possible. The SPTE indicates when the next write can occur.

13.8 Error Conditions

The following flags signal SPI error conditions:

- Overflow (OVRF) — Failing to read the SPI Data register before the next full length data enters the Shift register sets the OVRF bit. The new data will not transfer to the Receive Data register, and the unread data can still be read. OVRF is in the SPSCR.
- Mode Fault Error (MODF) — The MODF bit indicates the voltage on the Slave Select pin ($\overline{SS}$) is inconsistent with SPI mode. MODF is in the SPSCR.

13.8.1 Overflow Error

The Overflow Flag (OVRF) becomes set if the last bit of a current transmission is received and the Receive Data register still has unread data from a previous transmission. If an overflow occurs, all data received after the overflow, and before the OVRF bit is cleared, does not transfer to the Receive Data Register (SPRDR). It does not set the SPI Receiver Full (SPRF) bit. The unread data transferred to the Receive Data register before the overflow occurred can still be read. Therefore, an overflow error always indicates the loss of data. Clear the overflow flag by reading the SPSCR, then read the SPRDR.

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

If the SPRF interrupt is enabled and the OVRF interrupt is not, watch for an overflow condition. **Figure 13-8** explains how it is possible to miss an overflow. The first element of the same figure illustrates how it is possible to read the SPSCR and SPDRR to clear the SPRF without problems. However, as illustrated by the second transmission example, the OVRF bit can be set between the time SPSCR and SPDRR are read.

In this case, an overflow can easily be missed. Since no more SPRF interrupts can be generated until this OVRF is serviced, it is not obvious data is being lost as more transmissions are completed. To prevent this loss, either enable the OVRF interrupt or take another read of the SPSCR following the read of the SPDRR. This ensures the OVRF was not set before the SPRF was cleared and future transmissions can set the SPRF bit.

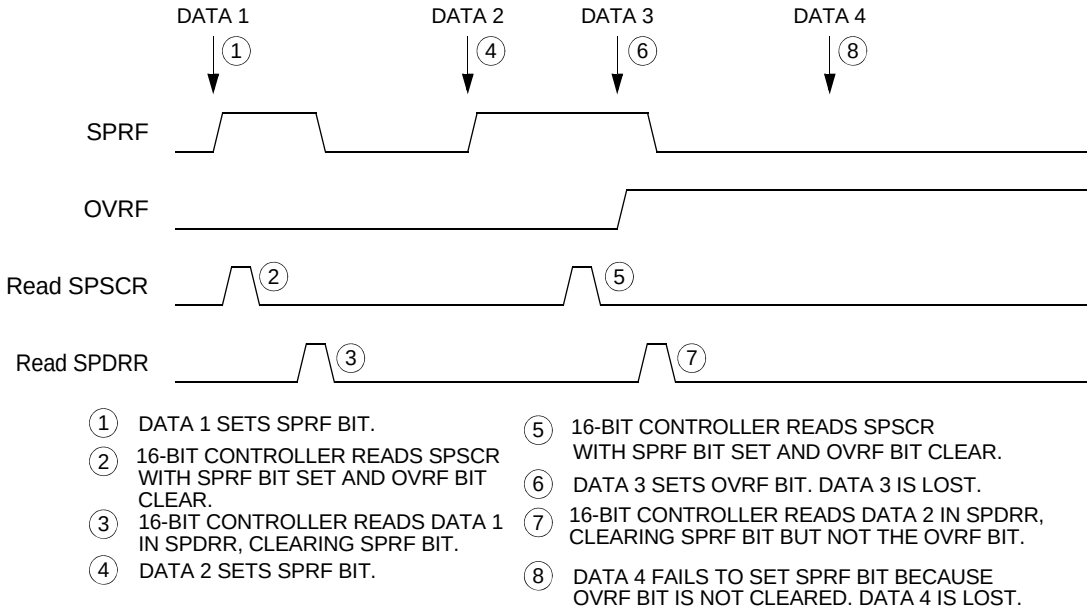


Figure 13-8. Missed Read of Overflow Condition

Figure 13-9 illustrates the described process. Generally, to avoid a second SPSCR read, enable the OVRF by setting the ERRIE bit.

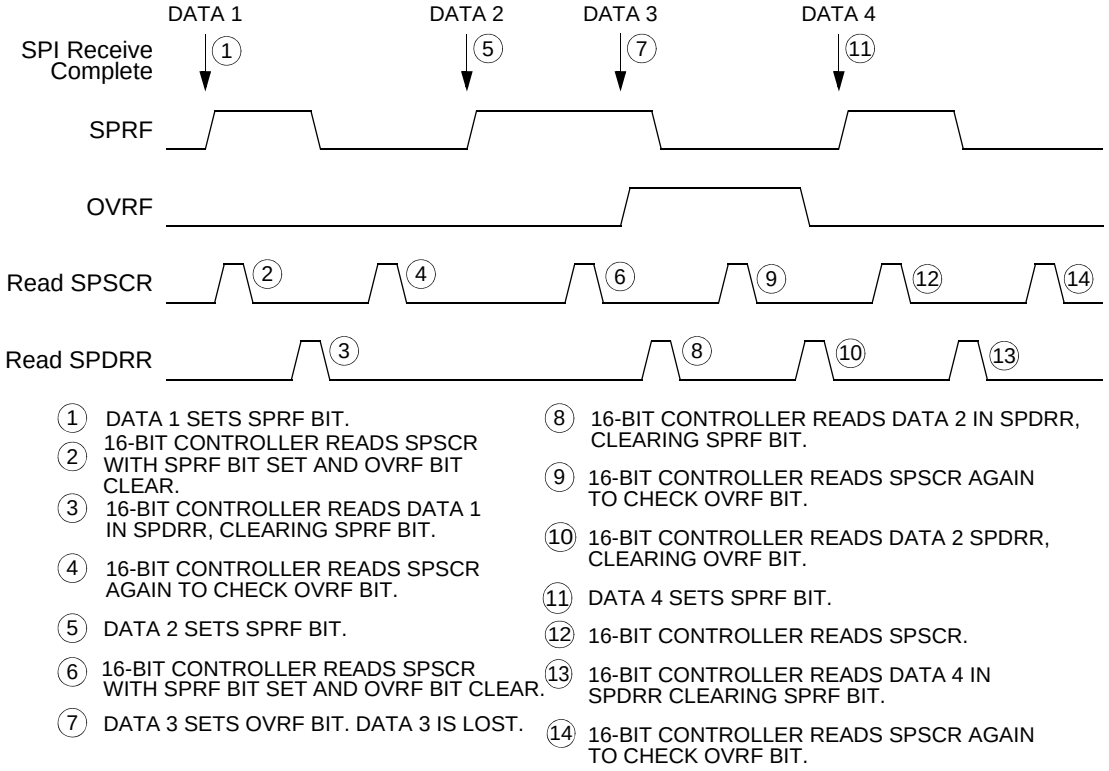


Figure 13-9. Clearing SPRF When OVRF Interrupt Is Not Enabled

13.8.2 Mode Fault Error

Setting the SPMSTR bit selects Master mode, configuring the SCLK and MOSI pins as outputs and the MISO pin as an input. Clearing SPMSTR selects Slave mode, configuring the SCLK and MOSI pins as inputs and the MISO pin as an output. The Mode Fault (MODF) bit becomes set any time the state of the $\overline{SS}$ pin is inconsistent with the mode selected by SPMSTR. To prevent SPI pin contention and damage to the 16-bit controller, a Mode Fault Error occurs if:

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

To set the MODF flag, Mode Fault Error Enable (MODFEN) bit must be set. Clearing the MODFEN bit does not clear the MODF flag but it does prevent the MODF from being set again after the MODF is cleared.

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

13.8.2.1 Master SPI Mode Fault

In a master SPI with Mode Fault Enable (MODFEN) bit set, Mode Fault (MODF) flag is set if $\overline{SS}$ goes to Logic 0. A Mode Fault in a Master SPI causes the following events to occur:

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

In a master SPI, the MODF flag will not be cleared until the $\overline{SS}$ pin is at a Logic 1 or the SPI is configured as a slave.

Note: When CPHA = 0, a MODF occurs if a slave is selected ($\overline{SS}$ is at Logic 0) and later unselected ($\overline{SS}$ is at Logic 1) even if no SCLK is sent to that slave. This happens because $\overline{SS}$ at Logic 0 indicates the start of the transmission (MISO driven out with the value of MSB) for CPHA = 0. When CPHA = 1, a slave can be selected and then later unselected with no transmission occurring. Therefore, MODF does not occur since a transmission was never begun.

13.8.2.2 Slave SPI Mode Fault

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

Note: A Logic 1 voltage on the $\overline{SS}$ pin of a slave SPI puts the MISO pin in a high impedance state. Also, the slave SPI ignores all incoming SCLK clocks, even if it was already in the middle of a transmission.

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

To clear the MODF flag, read the SPSCR with the MODF bit set and then write to the SPSCR. This entire clearing mechanism must occur with no MODF condition existing or else the flag is not cleared.

In a slave SPI, if the MODF flag is not cleared by writing a one to the MODF bit, the condition causing the Mode Fault still exists. In this case, the interrupt caused by the MODF flag can be cleared by disabling the EERIE or MODFEN bits (if set) or by disabling the SPI. Disabling the SPI using the SPE bit will cause a partial reset of the SPI and may cause the loss of a message currently being received or transmitted.

13.9 Register Definitions

Table 13-3. SPI Memory Map

Device	Register Map	
80x	SPI_BASE	$0F20

Table 13-3 lists the SPI registers in ascending address, including their acronyms and address of each register. The read/write registers should be accessed only with word accesses. Accesses other than word lengths result in undefined results. **Table 13-6** portrays a map summary of the registers.

Table 13-4. SPI Register Summary

Address Offset	Register Acronym	Register Name	Access Type	Register Location
Base + $0	SPSCR	Status and Control Register	Read/Write	Section 13.9.1
Base + $1	SPDSR	Data Size Register	Read/Write	Section 13.9.2
Base + $2	SPDRR	Data Receive Register	Read/Write	Section 13.9.3
Base + $3	SPDTR	Data Transmit Register	Read/Write	Section 13.9.4

Bit fields of the four registers are summarized in **Figure 13-10.** Details of each follow.

Add. Offset	Register Name		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$0	SPSCR	R	0	DSO	SPRF	ERRIE	OVRF	MODF	SPTE	MOD FEN	SPR1	SPR0	SPRIE	SPMSTR	CPOL	CPHA	SPE	SPTIE
		W																
$1	SPDSR	R	0	0	0	0	0	0	0	0	0	0	0	0	DS3	DS2	DS1	DS0
		W																
$2	SPDRR	R	R15	R14	R13	R12	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0
		W																
$3	SPDTR	R																
		W	T15	T14	T13	T12	T11	T10	T9	T8	T7	T6	T5	T4	T3	T2	T1	T0

R	0	Read as 0
W		Reserved

Figure 13-10. SPI Register Map Summary

13.9.1 SPI Status and Control Register (SPSCR)

The SPSCR:

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

SPI_BASE+$0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	DSO	SPRF	ERRIE	OVRF	MODF	SPTE	MODFEN	SPR1	SPR0	SPRIE	SPMSTR	CPOL	CPHA	SPE	SPTIE
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 13-11. SPI Status and Control Register (SPSCR)

See Table A-8, List of Programmer's Sheets

Note: Using BFCLR or BFSET instructions on the SPSCR can cause unintended side effects on the status bits. This is due to these instructions being a read/write modify instruction.

13.9.1.1 Reserved—Bit 15

This bit is reserved or not implemented. It is read as 0 and cannot be modified by writing.

13.9.1.2 Data Shift Order (DSO)—Bit 14

This read/write bit determines whether the MSB or LSB bit is transmitted or received first. Both Master and Slave SPI modules must transmit and receive the same length packets. Regardless how this bit is set, when reading from the SPDRR or writing to the SPDTR, the LSB will always be at bit location zero. If the data length is less than 16 bits, the data will be zero padded on the upper bits.

- 0 = MSB transmitted first
- 1 = LSB transmitted first

13.9.1.3 SPI Receiver Full (SPRF)—Bit 13

This *read-only* flag is set each time data transfers from the Shift register to the SPDRR. SPRF generates an interrupt request if the SPRIE bit in the SPI Control register is set. This bit is cleared by reading the SPDRR.

- 0 = Data Receive Register (SPDRR) not full
- 1 = Data Receive Register (SPDRR) full

13.9.1.4 Error Interrupt Enable (ERRIE)—Bit 12

This read/write bit enables the MODF, if MODFEN is also set, and OVRF bits to generate interrupt requests.

- 0 = MODF and OVRF cannot generate interrupt requests
- 1 = MODF and OVRF can generate interrupt requests

13.9.1.5 Overflow (OVRF)—Bit 11

This *read-only* flag is set if software does not read the data in the SPDRR before the next full data enters the Shift register. In an overflow condition, the data already in the SPDRR is unaffected, and the data shifted in last is lost. Clear the OVRF bit by reading the SPSCR and then reading the SPDRR. OVRF generates a Receiver/Error interrupt if the EERIE bit is set. Please see **Section 13.8.1, "Overflow Error"** for more details.

- 0 = No overflow
- 1 = Overflow

13.9.1.6 Mode Fault (MODF)—Bit 10

This *read-only* flag is set in a slave SPI if the $\overline{SS}$ pin goes high during a transmission with the MODFEN bit set. In a master SPI, the MODF flag is set if the $\overline{SS}$ pin goes low at any time with the MODFEN bit set. Clear the MODF bit by writing a one to the MODF bit when it is set. MODF generates a Receive/Error interrupt if the EERIE bit is set. Please see **Section 13.8.2, "Mode Fault Error"** for more details

- 0 = $\overline{SS}$ pin at appropriate Logic level
- 1 = $\overline{SS}$ pin at inappropriate Logic level

13.9.1.7 SPI Transmitter Empty (SPTE)—Bit 9

This *read-only* flag is set each time the SPDTR transfers data into the Shift register. SPTE generates an interrupt request if the SPTIE bit in the SPI Control register is set. This bit is cleared by writing to the SPDTR.

- 0 = Data Transmit Register (SPDTR) not empty
- 1 = Data Transmit Register (SPDTR) empty
- Do not write to the SPI Data register unless the SPTE bit is high.

13.9.1.8 Mode Fault Enable (MODFEN)—Bit 8

This read/write bit, when set to one, allows the MODF flag to be set. If the MODF flag is set, clearing the MODFEN does not clear the MODF flag.

If the MODFEN bit is low, the level of the $\overline{SS}$ pin does not affect the operation of an enabled SPI configured as a master. For an enabled SPI configured as a slave, having MODFEN low only prevents the MODF flag from being set. It does not affect any other part of SPI operation.

13.9.1.9 SPI Baud Rate Select (SPR1 and SPR0)—Bits 7–6

While in the Master mode, these read/write bits select one of eight baud rates depicted in **Table 13-5.** SPR[2:0] have no effect in Slave mode. Use the formula below to calculate the SPI baud rate.

$$\text{Baud Rate} = \frac{\text{Module Clock}}{\text{Baud Rate Divisor}}$$

Table 13-5. SPI Master Baud Rate Selection

SPR[1:0]	Baud Rate Divisor (BD)
00	2
01	8
10	16
11	32

13.9.1.10 SPI Receiver Interrupt Enable (SPRIE)—Bit 5

This read/write bit enables interrupt requests generated by the SPRF bit. The SPRF bit is set when data transfers from the Shift register to the Receive Data register.

- 0 = SPRF interrupt requests disabled
- 1 = SPRF interrupt requests enabled

13.9.1.11 SPI Master (SPMSTR)—Bit 4

This read/write bit selects Master or Slave modes operation.

- 0 = Slave mode
- 1 = Master mode

13.9.1.12 Clock Polarity (CPOL)—Bit 3

This read/write bit determines the logic state of the SCLK pin between transmissions. To transmit data between SPI modules, the SPI modules must have identical CPOL values. Please see **Figure 13-3** and **Figure 13-5.**

13.9.1.13 Clock Phase (CPHA)—Bit 2

This read/write bit controls the timing relationship between the serial clock and SPI data. Please see **Figure 13-4** and **Figure 13-5.** To transmit data between SPI modules, the SPI modules must have identical CPHA values. When CPHA = 0, the $\overline{SS}$ pin of the Slave SPI module must be set to Logic 1 between data transmissions, illustrated in **Figure 13-4.**

13.9.1.14 SPI Enable (SPE)—Bit 1

This read/write bit enables the SPI module. Clearing SPE causes a partial reset of the SPI. When setting/clearing this bit, *no* other bits in the SPSCR should be changed. Failure to following this statement may result in spurious clocks.

- 0 = SPI module disabled
- 1 = SPI module enabled

13.9.1.15 SPI Transmit Interrupt Enable (SPTIE)—Bit 0

This read/write bit enables interrupt requests generated by the SPTE bit. SPTE is set when data transfers from the SPDTR to the Shift register.

- 0 = SPTE interrupt requests disabled
- 1 = SPTE interrupt requests enabled

13.9.2 SPI Data Size Register (SPDSR)

This read/write register determines the data length for each transmission. The master and slave must transfer the same size data on each transmission. A new value will only take effect at the time the SPI is enabled, SPE bit in SPSCR set from a zero to a one. In order to have a new value take effect, disable then re-enable the SPI bit in the SPSCR with the new value in the register.

SPI_BASE+$1	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	0	0	0	0	0	0	0	DS3	DS2	DS1	DS0
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 13-12. SPI Data Size Register (SPDSR)

See Table A-8, List of Programmer's Sheets

13.9.2.1 Data Size (DS)—Bits 3–0

Please see **Table 13-6** for detailed transmission data.

Table 13-6. Data Size

DS[3:0]	Size of Transmission
$0	Not Allowed
$1	2 Bits
$2	3 Bits
$3	4 Bits
$4	5 Bits
$5	6 Bits
$6	7 Bits
$7	8 Bits
$8	9 Bits
$9	10 Bits
$A	11 Bits
$B	12 Bits
$C	13 Bits
$D	14 Bits
$E	15 Bits
$F	16 Bits

13.9.3 SPI Data Receive Register (SPDRR)

This *read-only* register will show the last data word received after a complete transmission. The SPRF bit is set when new data is transferred to this register.

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

Figure 13-13. SPI Data Receive Register (SPDRR)

See Table A-8, List of Programmer's Sheets

13.9.4 SPI Data Transmit Register (SPDTR)

This *write-only* register holds data to be transmitted. When the SPTE bit is set, new data should be written to this register. If new data is not written while in the Master mode, a new transaction will not be initiated until this register is written. When in Slave mode, the old data will be re-transmitted. All data should be written with the LSB at bit zero.

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

Figure 13-14. SPI Data Transmit Register (SPDTR)

See Table A-8, List of Programmer's Sheets

13.10 Resets

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

- SPTE flag is set
- Any Slave mode transmission currently in progress is aborted
- Any Master mode transmission currently in progress is continued to completion
- SPI state counter is cleared, making it ready for a new complete transmission
- All the SPI port logic is disabled

The following items are reset only by a system reset:

- The SPDTR and SPDRR registers
- All control bits in the SPSCR (MODFEN, ERRIE, SPR[2:0])
- The status flags SPRF, OVRF, and MODF

By not resetting the control bits when SPE is low, it is possible to clear SPE between transmissions without having to set all control bits again when SPE is set back high for the next transmission.

By not resetting the SPRF, OVRF, and MODF flags, it is possible to service the interrupts after the SPI has been disabled. Disable SPI by writing zero to the SPE bit. SPI can also be disabled by a Mode Fault occurring in a SPI configured as a master.

13.11 Interrupts

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

Table 13-7. SPI Interrupts

Flag	Interrupt Enabled By	Description
SPTE (Transmitter Empty)	SPI Transmitter Interrupt Request (SPTIE = 1,SPE = 1)	The SPI transmitter interrupt enable bit (SPTIE) enables the SPTE flag to generate transmitter interrupt requests provided that the SPI is enabled (SPE = 1). The SPTE bit becomes set every time data transfers from the SPDTR to the Shift register. The clearing mechanism for the SPTE flag is a write to the SPDTR.
SPRF (Receiver Full)	SPI Receiver Interrupt Request (SPRIE = 1)	The SPI Receiver Interrupt Enable bit (SPRIE) enables the SPRF bit to generate receiver interrupt requests regardless of the state of the SPE bit. The SPRF is set every time data transfers from the Shift register to the SPDRR. The clearing mechanism for the SPRF flag is to read the SPDRR.
OVRF (Overflow)	SPI Receiver/Error Interrupt Request (ERRIE = 1)	The error interrupt enable bit (ERRIE) enables both the MODF and OVRF bits to generate a receiver/error interrupt request.
MODF (Mode Fault)	SPI Receiver/Error Interrupt Request (ERRIE = 1, MODFEN = 1)	The Mode Fault enable bit (MODFEN) can prevent the MODF flag from being set so that only the OVRF bit is enabled by the ERRIE bit to generate receiver/error 16-bit controller interrupt requests.

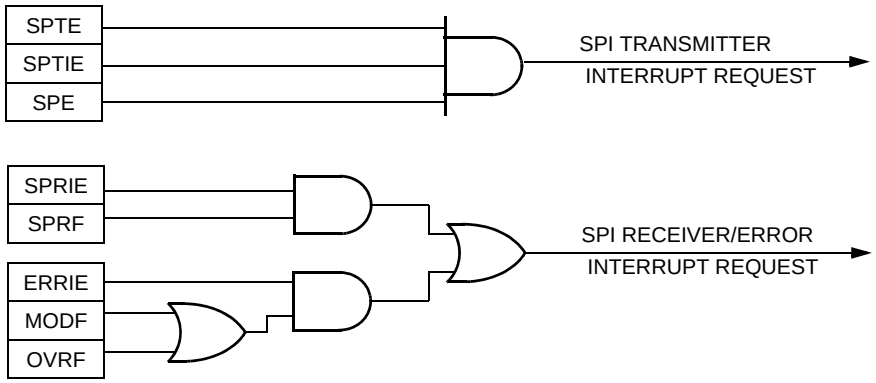


Figure 13-15. SPI Interrupt Request Generation

Table 13-8. Document Revision History for Chapter 13

Version History	Description of Change
Rev. 8	Formatting, layout, spelling, and grammar corrections. Added revision history table. In Figure 13-10, corrected the name of bit 12 in SPSCR (was EERIE, is ERRIE).

Chapter 14
Quad Timer Module (TMR)

14.1 Introduction

There are up to four different possible Quad Timer modules on the different configurations of the 56F800 series. The 56F801/802 have two Quad Timer modules, C and D, while the 56F803/805/807 have four Quad Timer modules: A, B, C, and D. Timer modules C and D have dedicated pins. Timer module C has two dedicated pins on the 56F805/807. Timer modules A and B share pins with the Quad Decoder modules.

Each timer module contains four identical Counter/Timer groups, also called channels. Each 16-bit counter timer group contains a Prescaler, a Counter, a Load Register, a Hold Register, a Capture Register, two Compare Registers, a Status and Control Register, and a Control Register. All of the registers, except the prescaler, are read/write registers.

Note: This document uses the terms Timer and Counter interchangeably because the Counter/Timers may perform either or both tasks.

The Prescaler provides different time bases useful for clocking the Counter/Timer. The Counter provides the ability to count internal or external events. The Load register provides the initialization value to the Counter when the Counter's terminal value has been reached. The Hold register captures the Counter's value when other counters are being read. This feature supports the reading of cascaded counters. The Capture register implements an external signal to take a snapshot of the Counter's current value. The Compare registers provide the values enabling counters to be compared. If a match occurs, the OFLAG signal can be set, cleared, or toggled. At match time, an interrupt is generated if enabled. Within a Time module, a set of four Counters/Timers, the input pins are apportioned.

14.2 Features

- Each timer module consists of four 16-bit counters/timers
- Count up/down
- Counters are cascadable
- Programmable count modulus
- Max count rate equals peripheral clock/2 when counting external events

- Max count rate equals peripheral clock when using internal clocks
- Count once or repeatedly
- Counters are preloadable
- Counters can share available input pins
- Each counter has a separate prescaler
- Each counter has capture and compare capability

14.3 Block Diagram

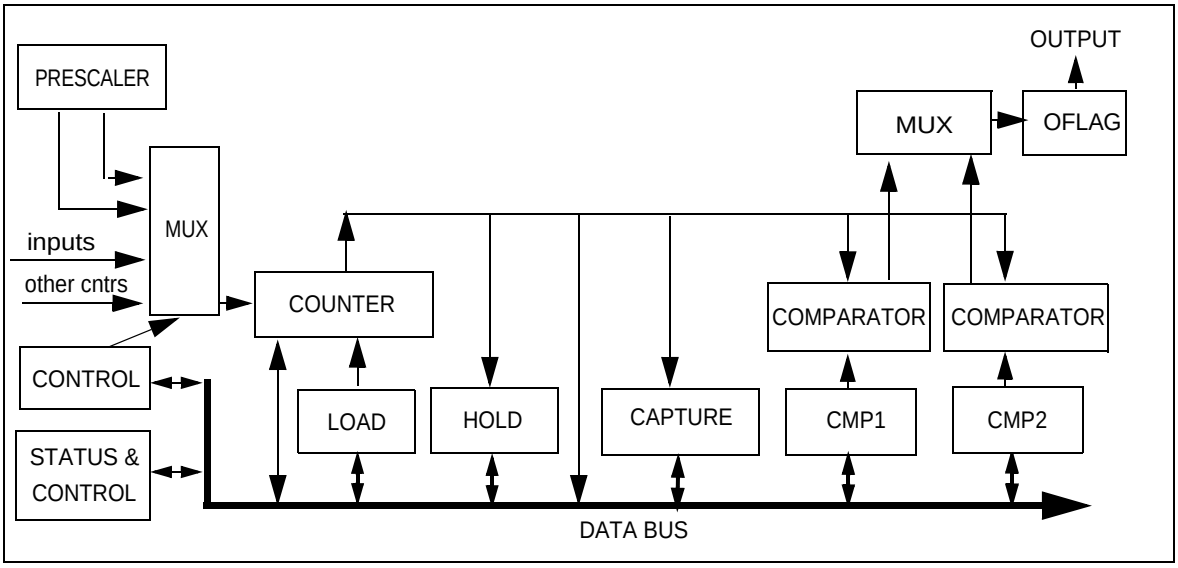


Figure 14-1. Counter/Timer Block Diagram

14.4 Pin Descriptions

The pins available for the 56F80x devices are different for each configuration. Please be especially attentive to the available pins. Please refer to Section 14.8, “Timer Group A, B, C, and D Functionality”.

14.5 Functional Description

The Counter/Timer has two basic modes of operation:

1. It can count internal or external events.
2. It can count an internal clock source while an external input signal is asserted thus timing the width of the external input signal.

14.5.1 Counting Options

The counter can count the rising, falling, or both edges of the selected input pin. The counter can decode and count quad encoded input signals. The counter can count up and down using dual inputs in a *count with direction* format. The counter's terminal count value (modulo) is programmable. The value loaded into the counter after reaching its terminal count is programmable. The counter can count repeatedly, or it can stop after completing one count cycle. The counter can be programmed to count to a programmed value and then immediately re initialize, or it can count through the compare value until the count moves to zero.

14.5.2 External Inputs

The external inputs to each Counter/Timer are capable of being shared among each of the four Counter/Timers within a module. The external inputs can be used as count commands and timer commands. They can trigger the current counter value to be captured and then be used to generate interrupt requests. The polarity of the external inputs are selectable.

14.5.3 OFLAG Output Signal

The primary output of each Counter/Timer is the output signal OFLAG. The OFLAG output signal can be set, cleared, or toggled when the counter reaches the programmed value. The OFLAG output signal may be outputted to an external pin shared with an external input signal. The OFLAG output signal enables each counter to generate square waves, PWM, or pulse stream outputs. The polarity of the OFLAG output signal is selectable.

14.5.4 Master Signal

Any Counter/Timer can be assigned as a master. A master's compare signal can be broadcast to the other Counters/Timers within a module. The other counters can be configured to re initialize their counters and/or force their OFLAG output signals to predetermined values when a master's Counter/Timer compare event occurs.

14.6 Counting Mode Definitions

The selected external count signals are sampled at the module's base clock rate and then run through a transition detector. The maximum count rate is one-half of the base clock rate. Internal clock sources can be used to clock the counters up to the base clock rate.

If a counter is programmed to count to a specific value and then stop, the Count mode in the CTRL register is cleared when the count terminates.

14.6.1 Stop Mode

If the Count mode field is set to 000, the counter is inert. No counting will occur.

14.6.2 Count Mode

If Count mode field is set to 001, the counter will count the rising edges of the selected clock source. This mode is useful for counting generating periodic interrupts for timing purposes, or counting external events such as *widgets* on a conveyor belt passing a sensor. If the selected input is inverted by setting the Input Polarity Select (IPS) bit, then the negative edge of the selected signal is counted.

14.6.3 Edge-Count Mode

If Count mode field is set to 010, the counter will count the both edges of the selected clock source. This mode is useful for counting the changes in the external environment such as a simple encoder wheel.

14.6.4 Gated-Count Mode

If Count mode field is set to 011, the counter will count while the selected secondary input signal is high. This mode is used to time the duration of external events. If the selected input is inverted by setting the IPS bit, then the counter will count while the selected secondary input is low.

14.6.5 Quad-Count Mode

If Count mode field is set to 100, the counter will decode the primary and secondary external inputs as quad encoded signals. Quad signals are usually generated by rotary or linear sensors used to monitor movement of motor shafts or mechanical equipment. The quad signals are square waves 90 degrees out-of-phase. The decoding of quad signal provides both count and direction information.

A timing diagram illustrating the basic operation of a quad incremental position encoder is shown below:

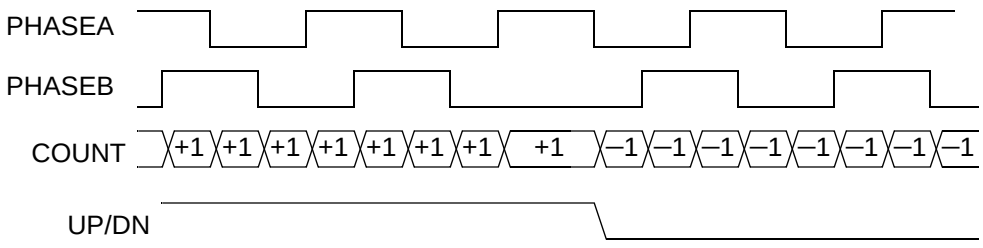


Figure 14-2. Timing Diagram

14.6.6 Signed-Count Mode

If Count mode field is set to 101, the counter counts the primary clock source while the selected secondary source provides the selected count direction, up/down. If the secondary source, direction select, input is high, the counter will count in a negative direction. If the select direction signal is low, the counter will count in a positive direction

14.6.7 Triggered-Count Mode

If Count mode field is set to 110, the counter will begin counting the rising edges of the primary clock source after a positive transition of the secondary input occurs; negative transition if IPS = 1. The counting will continue until a compare event occurs or another input transition is detected. Odd edges restart the counting. Even edges stop the counting until a compare event occurs.

14.6.8 One-Shot Mode

This is a sub mode of triggered event Count mode when the Count mode field is set to 110 while:

- Count length (LENGTH) is set
- The OFLAG Output mode is set to 101
- ONCE bit of the Control Register (CTRL) is set to 1

Then the counter works in a One-shot mode. An external event causes the counter to count. When terminal count is reached, the OFLAG output is asserted. This delayed output assertion can be used to provide timing delays. When used with timer C2 or C3, this One-shot mode may be used to delay the ADC acquisition of new samples until a specified period of time has passed since the Pulse Width Modulated (PWM) module asserted the synchronized signal.

14.6.9 Cascade-Count Mode

If Count mode field is set to 111, the counter must be connected to the output of another counter and selected by the Primary Count Source. The counter will count-up and down as compare events occur in the selected source counter. This cascade, or daisy-chained mode, enables multiple counters to be cascaded to yield longer counter lengths. The cascade Count mode uses a special high speed signal path without regarding the state of the OFLAG signal. If the selected source counter experiences a CMP1 compare event while counting in a positive direction, the counter will increment. If the selected source counter experiences a CMP2 compare event while counting in a negative direction, the counter will decrement.

Whenever any counter is read within a counter module, all of the counters' values within a module are captured in their respective Hold registers. This action supports the reading of a cascaded counter chain. First read any counter of a cascaded counter chain, then read the Hold registers of the other counters in the chain. The cascaded Counter mode is synchronous.

Note: It is possible to connect counters together by using the other, non-cascade, counter modes and selecting the outputs of other counters as a clock source. In this case, the counters are operating in a *ripple* mode, where higher order counters will transition a clock later than a purely synchronous design.

14.6.10 Pulse-Output Mode

If the counter is setup for Count mode (Mode = 001), and the OFLAG Output mode is set to 111, gated clock output, and the COUNT ONCE bit is set, then the counter will output a pulse stream of pulses with the same frequency of the selected clock source. The number of output pulses is equal to the compare value minus the initialization value. This mode is useful for driving step motor systems.

14.6.11 Fixed-Frequency PWM Mode

A sub mode of Count mode. If the Count mode field is set to 001 while:

- Count through roll-over (LENGTH = 0)
- Continuous count (ONCE = 0)
- OFLAG Output mode is 110 (set on compare, cleared on initialization)

The counter output then yields a PWM signal with a frequency equal to the count clock frequency divided by 65,536 and a pulse width duty cycle equal to the compare value divided by 65,536. This mode of operation is often used to drive PWM amplifiers or low cost DAC.

14.6.12 Variable-Frequency PWM Mode

If the counter is setup for:

- COUNT mode (Mode = 001)
- Count till compare (Count length = 1)
- Continuous count (Count OnCE = 0)
- OFLAG Output mode is 100 (toggle OFLAG and alternate compare registers)

The counter output then yields a PWM signal. Its frequency and pulse width are determined by the values programmed into the CMP1 and CMP2 registers, and the input clock frequency. This method of PWM generation has the advantage of allowing almost any desired PWM frequency

and/or constant on or off periods. This mode of operation is often used to drive PWM amplifiers used to power motors and inverters.

14.6.13 Compare Registers Usage

The dual Compare (CMP1 and CMP2) registers provide a bidirectional modulo count capability. The CMP1 register is used when the counter is counting up, and the CMP2 register is used when the counter is counting down. The only exception is alternating compare mode.

The CMP1 register should be set to the desired maximum count value or $FFFF to indicate the maximum unsigned value prior to roll-over, and the CMP2 register should be set to the maximum negative count value or $0000 to indicate the minimum unsigned value prior to roll-under.

If the Output mode is set to 100, the OFLAG will toggle while using alternating compare registers. In this variable frequency PWM mode, the CMP2 value defines the desired pulse width of the on-time, and the CMP1 register defines the off-time. The variable frequency PWM mode is defined for positive counting only.

One must be careful when changing CMP1 and CMP2 while the counter is active. If the counter has already passed the new value, it will count to $FFFF or $0000 (roll over) and then begin counting toward the new value. The check is for count to equal CMPx, not
Count $\geq$ CMP1 or Count $\leq$ CMP2.

Additionally, care must be taken when modifying the CMP1 and CMP2 when they are used with the varioius output modes. The interrupt generation for the compare is performed on the leading edge of the timer clock and the compare for the OFLAG Output mode is perform on the trailing edge. With some Primary Count Sources, this enables the ISR to possibly modify the CMP1 or CMP2 values before the compare for the OFLAG mode is performed.

14.6.14 Capture Register Usage

The Capture register stores a copy of the counter's value, an input transition is detected. The register depends on the Capture mode and IPS settings. Once a capture event occurs, no further updating of the Capture register will occur until the Input Edge Flag (IEF) is cleared by writing 0 to the IEF bit of the SCR. Please refer to **Section 14.7.2.9, "Input Capture Mode (Capture Mode)—Bits 7–6".**

14.7 Register Definitions

Table 14-1. TMR Memory Map

Device	Peripheral	Address
801/802/ 803/805	TMRA_BASE	$0D00
	TMRB_BASE	$0D20
	TMRC_BASE	$0D40
	TMRD_BASE	$0D60
807	TMRA_BASE	$1100
	TMRB_BASE	$1120
	TMRC_BASE	$1140
	TMRD_BASE	$1160

The address of a register is the sum of a base address and an address offset. The base address is defined at the MCU level and the address offset is defined at the module level. Please refer to **Table 3-11** for the appropriate TMR_BASE definition. The base address given for each register will be either TMRA_BASE, TMRB_BASE, TMRC_BASE, or TMRD_BASE depending on which Quad Timer is being used.

Make certain to check which Quad Timers are available on the chip being used. The 56F801 has a dedicated Quad Timer D. Controller 56F803 has a dedicated Quad Timer D and optional Quad Timer A, multiplexed to the Quad Decoder while 56F805 and 56F807 have two dedicated Quad Timers C and D. They also have two optional Quad Timers A and B multiplexed to Quad Decoders.

A suffix is added to each register reflecting the Quad Timer module and channel (group) being accessed: TMRA0, TMRA1, TMRA2, TMRA3, TMRB0, TMRB1, TMRB2, TMRB3, TMRC0, TMRC1, TMRC2, TMRC3, TMRD0, TMRD1, TMRD2, TMRD3.

For example, the CNTR register for Quad Timer module A, Channel Zero (TMRA0) module will be called TMRA0_CNTR. Each Timer/Counter in the 56F80x has the following registers:

- Quad Timer Counter (CNTR) register
- Quad Timer Load (LOAD) register
- Quad Timer Hold (HOLD) register
- Quad Timer Capture (CAP) register
- Quad Timer Compare (CMP1 and CMP2) registers
- Quad Timer Status and Control (SCR) register
- Quad Timer Control (CTRL) register

For further information about:

- TMRA registers please refer to **Section 14.8.1**
- TMRB registers please refer to **Section 14.8.2**
- TMRC registers please refer to **Section 14.8.3**
- TMRD registers please refer to **Section 14.8.4**

Table 14-2. TMR Register Summary

Address Offset	Register Acronym	Register Name	Access Type	Register Location
TMRX_Base + $6 – $1E	CTRL	Control Register	Read/Write	Section 14.7.1
TMRX_Base + $7 – $1F	SCR	Status & Control Register	Read/Write	Section 14.7.2
TMRX_Base + $0 – $18	CMP1	Compare Register 1	Read/Write	Section 14.7.3
TMRX_Base + $1 – $19	CMP2	Compare Register 2	Read/Write	Section 14.7.4
TMRX_Base + $2 – $1A	CAP	Capture Register	Read/Write	Section 14.7.5
TMRX_Base + $3 – $1B	LOAD	Load Register	Read/Write	Section 14.7.6
TMRX_Base + $4 – $1C	HOLD	Hold Register	Read/Write	Section 14.7.7
TMRX_Base + $5 – $1D	CNTR	Counter Register	Read/Write	Section 14.7.8

Bit fields of the eight registers, each with 16 modules, are summarized in **Figure 14-3.** Details of each follow.

Add. Offset	Register Name		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TMRxn	CTRL	R	COUNT MODE			PRIMARY COUNT SOURCE				SECONDARY SOURCE		ONCE	LENGTH	DIR	Co Init	OUTPUT MODE		
		W																
TMRxn	SCR	R	TCF	TCFIE	TOF	TOFIE	IEF	IEFIE	IPS	INPUT	CAPTURE MODE		MSTR	EEOF	VAL		OPS	OEN
		W														FORCE		
TMRxn	CMP1	R	COMPARISON VALUE[15:0]															
		W																
TMRxn	CMP2	R	COMPARISON VALUE[15:0]															
		W																
TMRxn	CAP	R	CAPTURE[15:0]															
		W																
TMRxn	LOAD	R	LOAD[15:0]															
		W																
TMRxn	HOLD	R	HOLD VALUE[15:0]															
		W																
TMRxn	CNTR	R	COUNTER[15:0]															
		W																

x = A,B,C, or D n = 0,1,2, 3

R	0	Read as 0
W		Reserved

Figure 14-3. TMR Register Map Summary

14.7.1 Control Registers (CTRL)

TMRA0_CTRL (Timer A, Channel 0 Control)—Address: TMRA_BASE + $6
TMRA1_CTRL (Timer A, Channel 1 Control)—Address: TMRA_BASE + $E
TMRA2_CTRL (Timer A, Channel 2 Control)—Address: TMRA_BASE + $16
TMRA3_CTRL (Timer A, Channel 3 Control)—Address: TMRA_BASE + $1E

TMRB0_CTRL (Timer B, Channel 0 Control)—Address: TMRB_BASE + $6
TMRB1_CTRL (Timer B, Channel 1 Control)—Address: TMRB_BASE + $E
TMRB2_CTRL (Timer B, Channel 2 Control)—Address: TMRB_BASE + $16
TMRB3_CTRL (Timer B, Channel 3 Control)—Address: TMRB_BASE + $1E

TMRC0_CTRL (Timer C, Channel 0 Control)—Address: TMRC_BASE + $6
TMRC1_CTRL (Timer C, Channel 1 Control)—Address: TMRC_BASE + $E
TMRC2_CTRL (Timer C, Channel 2 Control)—Address: TMRC_BASE + $16
TMRC3_CTRL (Timer C, Channel 3 Control)—Address: TMRC_BASE + $1E

TMRD0_CTRL (Timer D, Channel 0 Control)—Address: TMRD_BASE + $6
TMRD1_CTRL (Timer D, Channel 1 Control)—Address: TMRD_BASE + $E
TMRD2_CTRL (Timer D, Channel 2 Control)—Address: TMRD_BASE + $16
TMRD3_CTRL (Timer D, Channel 3 Control)—Address: TMRD_BASE + $1E

TMR_BASE+$6, E,16, or 1E	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	COUNT MODE			PRIMARY COUNT SOURCE				SECONDARY SOURCE		ONCE	LENGTH	DIR	Co INIT	OUTPUT MODE		
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 14-4. Control Register (CTRL)

See Table A-8, List of Programmer's Sheets

14.7.1.1 Count Mode—Bits 15–13

These bits control the basic counting and behavior of the counter.

- 000 = No operation
- 001 = Count rising edges of primary source[1]
- 010 = Count rising and falling edges of primary source
- 011 = Count rising edges of primary source while secondary input high active
- 100 = Quad count mode, uses primary and secondary sources
- 101 = Count primary source rising edges, secondary source specifies direction (1 = minus)[2]

1. Rising Edges counted only when IPS = 0. Falling edges counted when IPS = 1.
2. Rising Edges counted only when IPS = 0. Falling edges counted when IPS = 1.

- 110 = Edge of secondary source triggers primary count till compare
- 111 = Cascaded counter mode, up/down[1]

14.7.1.2 Primary Count Source—Bits 12–9

These bits select the Primary Count Source.

- 0000 = Counter 0 pin
- 0001 = Counter 1 pin
- 0010 = Counter 2 pin
- 0011 = Counter 3 pin
- 0100 = Counter 0 OFLAG
- 0101 = Counter 1 OFLAG
- 0110 = Counter 2 OFLAG
- 0111 = Counter 3 OFLAG
- 1000 = Prescaler (IPBus clock divide by 1)
- 1001 = Prescaler (IPBus clock divide by 2)
- 1010 = Prescaler (IPBus clock divide by 4)
- 1011 = Prescaler (IPBus clock divide by 8)
- 1100 = Prescaler (IPBus clock divide by 16)
- 1101 = Prescaler (IPBus clock divide by 32)
- 1110 = Prescaler (IPBus clock divide by 64)
- 1111 = Prescaler (IPBus clock divide by 128)

Note: A timer selecting its own output for input is not a legal choice. The result is no counting.

1. Primary Count Source must be set to one of the counter outputs.

14.7.1.3 Secondary Count Source (SCS)—Bits 8–7

These bits provide additional information, such as direction, used for counting.

- 00 = Counter 0 pin
- 01 = Counter 1 pin
- 10 = Counter 2 pin
- 11 = Counter 3 pin

14.7.1.4 Count Once (ONCE)—Bit 6

This bit selects continuous or one shot counting mode.

- 0 = Count repeatedly
- 1 = Count until compare and then stop. If *counting up*, successful compare occurs when counter reaches CMP1 value. If *counting down*, successful compare occurs when counter reaches CMP2 value. When the compare occurs, the timer module changes its Count Mode to Stop Mode (Count Mode = 0).

14.7.1.5 Count Length (LENGTH)—Bit 5

This bit determines whether the counter counts to the compare value and then reinitializes itself to the value specified in the *load* register, or the counter continues counting past the compare value, the binary roll over.

- 0 = Roll over
- 1 = Count until compare, then re initialize. If counting up, successful compare occurs when counter reaches CMP1 value. If counting down, successful compare occurs when counter reaches CMP2 value.[1]

14.7.1.6 Count Direction (DIR)—Bit 4

This bit selects either the normal count direction *up*, or the reverse direction, *down*.

- 0 = Count up
- 1 = Count down

1. When output mode $4 is used, alternating values of CMP1 and CMP2 are used to generate successful compares. For example, when output mode is $4, the counter counts until CMP1 value is reached, reinitializes, then counts until CMP2 value is reached, reinitializes, then counts until CMP1 value is reached, etc.

14.7.1.7 Co-Channel Initialization (Co Init)—Bit 3

This bit enables another Counter/Timer within the module to force the reinitialization of this Counter/Timer when it has an active compare event.

- 0 = Co-channel Counter/Timers can not force a reinitialization of this Counter/Timer
- 1 = Co-channel Counter/Timers may force a reinitialization of this Counter/Timer

14.7.1.8 Output Mode (OM)—Bits 2-0

These bits determine the mode of operation for the OFLAG output signal.

- 000 = Asserted while counter is active
- 001 = Clear OFLAG output on successful compare
- 010 = Set OFLAG output on successful compare
- 011 = Toggle OFLAG output on successful compare
- 100 = Toggle OFLAG output using alternating compare registers
- 101 = Set on compare, cleared on secondary source input edge
- 110 = Set on compare, cleared on counter rollover
- 111 = Enable gated clock output while counter is active

14.7.2 Status and Control Registers (SCR)

TMRA0_SCR (Timer A, Channel 0 Status and Control)—Address: TMRA_BASE + $7
TMRA1_SCR (Timer A, Channel 1 Status and Control)—Address: TMRA_BASE + $F
TMRA2_SCR (Timer A, Channel 2 Status and Control)—Address: TMRA_BASE + $17
TMRA3_SCR (Timer A, Channel 3 Status and Control)—Address: TMRA_BASE + $1F

TMRB0_SCR (Timer B, Channel 0 Status and Control)—Address: TMRB_BASE + $7
TMRB1_SCR (Timer B, Channel 1 Status and Control)—Address: TMRB_BASE + $F
TMRB2_SCR (Timer B, Channel 2 Status and Control)—Address: TMRB_BASE + $17
TMRB3_SCR (Timer B, Channel 3 Status and Control)—Address: TMRB_BASE + $1F

TMRC0_SCR (Timer C, Channel 0 Status and Control)—Address: TMRC_BASE + $7
TMRC1_SCR (Timer C, Channel 1 Status and Control)—Address: TMRC_BASE + $F
TMRC2_SCR (Timer C, Channel 2 Status and Control)—Address: TMRC_BASE + $17
TMRC3_SCR (Timer C, Channel 3 Status and Control)—Address: TMRC_BASE + $1F

TMRD0_SCR (Timer D, Channel 0 Status and Control)—Address: TMRD_BASE + $7
TMRD1_SCR (Timer D, Channel 1 Status and Control)—Address: TMRD_BASE + $F
TMRD2_SCR (Timer D, Channel 2 Status and Control)—Address: TMRD_BASE + $17
TMRD3_SCR (Timer D, Channel 3 Status and Control)—Address: TMRD_BASE + $1F

TMR_BASE+$7, F,17, or 1F	15	14	13	12	11	10	9	8	[7:6]		5	4	3	2	1	0
Read	TCF	TCFIE	TOF	TOFIE	IEF	IEFIE	IPS	INPUT	Capture Mode		MSTR	EEOF	VAL	0	OPS	OEN
Write														FORCE		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 14-5. Status and Control Register (SCR)

See Table A-8, List of Programmer's Sheets

14.7.2.1 Timer Compare Flag (TCF)—Bit 15

This bit is set when a successful compare occurs. This bit is cleared by writing 0 to this bit location. The timer group will not assert another TCF interrupt until this bit has been cleared.

14.7.2.2 Timer Compare Flag Interrupt Enable (TCFIE)—Bit 14

When set, the timer compare interrupt is enabled.

14.7.2.3 Timer Overflow Flag (TOF)—Bit 13

This bit is set when the counter rolls over its maximum value $FFFF or $0000, depending on count direction. This bit is cleared by writing 0 to this bit location. The timer group will not assert another TOF interrupt until this bit is cleared.

14.7.2.4 Timer Overflow Flag Interrupt Enable (TOFIE)—Bit 12

When set, this bit activates interrupts when the timer overflow flag (TOF) bit is also set.

14.7.2.5 Input Edge Flag (IEF)—Bit 11

This bit is set when a capture occurs. The bit is cleared by writing 0 to the bit position.

Note: The timer group will not asset another input edge interrupt until this bit is cleared.

14.7.2.6 Input Edge Flag Interrupt Enable (IEFIE)—Bit 10

When set, the input edge interrupt is enabled.

14.7.2.7 Input Polarity Select (IPS)—Bit 9

When set, this bit inverts the polarity of both the primary and secondary inputs. It does not invert the polarity of an input signal when the signal is being driven by another timer group.

14.7.2.8 External Input Signal (INPUT)—Bit 8

This *read-only* bit reflects the current state of the external input pin after application of the Input Polarity Select (IPS) bit.

14.7.2.9 Input Capture Mode (Capture Mode)—Bits 7–6

These bits specify the operation of the Capture register as well as the operation of the input edge flag. The input capture mode bits seven to six must be 01, 10 or 11 in order for input edge flag interrupts to be asserted.

Table 14-3. Capture Register Operation

Capture Mode	IPS	Action
00	x	Capture Disabled
01	0	Load on Rising Edge
01	1	Load on Falling Edge
10	0	Load on Falling Edge
10	1	Load on Rising Edge
11	x	Load on Both Edges

14.7.2.10 Master Mode (MSTR)—Bit 5

When set, this bit enables the compare function's output to be broadcast to the other Counters/Timers in the module. This signal then can be used to re initialize the other counters and/or force their OFLAG signal outputs.

14.7.2.11 Enable External OFLAG Force (EEOF)—Bit 4

When set, this bit enables the compare from another Counter/Timer enabled as a master to force the state of this counters OFLAG output signal.

14.7.2.12 Forced OFLAG Value (VAL)—Bit 3

This bit determines the value of the OFLAG output signal when an software triggered FORCE command, or another Counter/Timer (set as a master) issues a FORCE command.

14.7.2.13 Force the OFLAG Output (FORCE)—Bit 2

This *write-only* bit forces the current value of the VAL bit to be written to the OFLAG output. This bit always reads as 0. The VAL and FORCE bits can be written simultaneously in a single write operation. Write to the FORCE bit only if the counter is disabled. Setting this bit while the counter is enabled may yield unpredictable results.

14.7.2.14 Output Polarity Select (OPS)—Bit 1

When set, this bit inverts the polarity of a signal driven by a timer group on to an external package pin. Other timer groups reading this pin will read the inverted signal.

- 0 = True polarity

- 1 = Inverted polarity

14.7.2.15 Output Enable (OEN)—Bit 0

When set, this bit enables the OFLAG output signal to be put on the external pin. Other timer groups using this external pin as their input will see the driven value. The polarity of the signal will be determined by the OPS bit.

- 0 = OFLAG output signal is disabled
- 1 = Enables the OFLAG output signal to be put on the external. Also connects a timer's output pin to its input

14.7.3 Compare Register 1 (CMP1)

TMRA0_CMP1 (Timer A, Channel 0 Compare #1)—Address: TMRA_BASE + $0
TMRA1_CMP1 (Timer A, Channel 1 Compare #1)—Address: TMRA_BASE + $8
TMRA2_CMP1 (Timer A, Channel 2 Compare #1)—Address: TMRA_BASE + $10
TMRA3_CMP1 (Timer A, Channel 3 Compare #1)—Address: TMRA_BASE + $18

TMRB0_CMP1 (Timer B, Channel 0 Compare #1)—Address: TMRB_BASE + $0
TMRB1_CMP1 (Timer B, Channel 1 Compare #1)—Address: TMRB_BASE + $8
TMRB2_CMP1 (Timer B, Channel 2 Compare #1)—Address: TMRB_BASE + $10
TMRB3_CMP1 (Timer B, Channel 3 Compare #1)—Address: TMRB_BASE + $18

TMRC0_CMP1 (Timer C, Channel 0 Compare #1)—Address: TMRC_BASE + $0
TMRC1_CMP1 (Timer C, Channel 1 Compare #1)—Address: TMRC_BASE + $8
TMRC2_CMP1 (Timer C, Channel 2 Compare #1)—Address: TMRC_BASE + $10
TMRC3_CMP1 (Timer C, Channel 3 Compare #1)—Address: TMRC_BASE + $18

TMRD0_CMP1 (Timer D, Channel 0 Compare #1)—Address: TMRD_BASE + $0
TMRD1_CMP1 (Timer D, Channel 1 Compare #1)—Address: TMRD_BASE + $8
TMRD2_CMP1 (Timer D, Channel 2 Compare #1)—Address: TMRD_BASE + $10
TMRD3_CMP1 (Timer D, Channel 3 Compare #1)—Address: TMRD_BASE + $18

TMR_BASE+$0, 8,10,18	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read / Write	COMPARISON VALUE															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 14-6. Compare Register One (CMP1)

See Table A-8, List of Programmer's Sheets

This read/write register stores the value used for comparison with the counter value.

14.7.4 Compare Register 2 (CMP2)

TMRA0_CMP2 (Timer A, Channel 0 Compare #2)—Address: TMRA_BASE + $1
TMRA1_CMP2 (Timer A, Channel 1 Compare #2)—Address: TMRA_BASE + $9
TMRA2_CMP2 (Timer A, Channel 2 Compare #2)—Address: TMRA_BASE + $11
TMRA3_CMP2 (Timer A, Channel 3 Compare #2)—Address: TMRA_BASE + $19

TMRB0_CMP2 (Timer B, Channel 0 Compare #2)—Address: TMRB_BASE + $1
TMRB1_CMP2 (Timer B, Channel 1 Compare #2)—Address: TMRB_BASE + $9
TMRB2_CMP2 (Timer B, Channel 2 Compare #2)—Address: TMRB_BASE + $11

TMRB3_CMP2 (Timer B, Channel 3 Compare #2)—Address: TMRB_BASE + $19

TMRC0_CMP2 (Timer C, Channel 0 Compare #2)—Address: TMRC_BASE + $1
TMRC1_CMP2 (Timer C, Channel 1 Compare #2)—Address: TMRC_BASE + $9
TMRC2_CMP2 (Timer C, Channel 2 Compare #2)—Address: TMRC_BASE + $11
TMRC3_CMP2 (Timer C, Channel 3 Compare #2)—Address: TMRC_BASE + $19

TMRD0_CMP2 (Timer D, Channel 0 Compare #2)—Address: TMRD_BASE + $1
TMRD1_CMP2 (Timer D, Channel 1 Compare #2)—Address: TMRD_BASE + $9
TMRD2_CMP2 (Timer D, Channel 2 Compare #2)—Address: TMRD_BASE + $11
TMRD3_CMP2 (Timer D, Channel 3 Compare #2)—Address: TMRD_BASE + $19

TMR_BASE+$1, 9,11,19	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	COMPARISON VALUE															
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 14-7. Compare Register Two (CMP2)

See Table A-8, List of Programmer's Sheets

This read/write register stores the value used for comparison with the counter value.

14.7.5 Capture Register (CAP)

TMRA0_CAP (Timer A, Channel 0 Capture)—Address: TMRA_BASE + $2
TMRA1_CAP (Timer A, Channel 1 Capture)—Address: TMRA_BASE + $A
TMRA2_CAP (Timer A, Channel 2 Capture)—Address: TMRA_BASE + $12
TMRA3_CAP (Timer A, Channel 3 Capture)—Address: TMRA_BASE + $1A

TMRB0_CAP (Timer B, Channel 0 Capture)—Address: TMRB_BASE + $2
TMRB1_CAP (Timer B, Channel 1 Capture)—Address: TMRB_BASE + $A
TMRB2_CAP (Timer B, Channel 2 Capture)—Address: TMRB_BASE + $12
TMRB3_CAP (Timer B, Channel 3 Capture)—Address: TMRB_BASE + $1A

TMRC0_CAP (Timer C, Channel 0 Capture)—Address: TMRC_BASE + $2
TMRC1_CAP (Timer C, Channel 1 Capture)—Address: TMRC_BASE + $A
TMRC2_CAP (Timer C, Channel 2 Capture)—Address: TMRC_BASE + $12
TMRC3_CAP (Timer C, Channel 3 Capture)—Address: TMRC_BASE + $1A

TMRD0_CAP (Timer D, Channel 0 Capture)—Address: TMRD_BASE + $2
TMRD1_CAP (Timer D, Channel 1 Capture)—Address: TMRD_BASE + $A
TMRD2_CAP (Timer D, Channel 2 Capture)—Address: TMRD_BASE + $12
TMRD3_CAP (Timer D, Channel 3 Capture)—Address: TMRD_BASE + $1A

TMR_BASE+$2,A, 12,1A	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	CAPTURE [15:0]															
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 14-8. Capture Register (CAP)

See Table A-8, List of Programmer's Sheets

This read/write register stores the value captured from the counter.

14.7.6 Load Register (LOAD)

TMRA0_LOAD (Timer A, Channel 0 Load)—Address: TMRA_BASE + $3
TMRA1_LOAD (Timer A, Channel 1 Load)—Address: TMRA_BASE + $B
TMRA2_LOAD (Timer A, Channel 2 Load)—Address: TMRA_BASE + $13
TMRA3_LOAD (Timer A, Channel 3 Load)—Address: TMRA_BASE + $1B

TMRB0_LOAD (Timer B, Channel 0 Load)—Address: TMRB_BASE + $3
TMRB1_LOAD (Timer B, Channel 1 Load)—Address: TMRB_BASE + $B
TMRB2_LOAD (Timer B, Channel 2 Load)—Address: TMRB_BASE + $13
TMRB3_LOAD (Timer B, Channel 3 Load)—Address: TMRB_BASE + $1B

TMRC0_LOAD (Timer C, Channel 0 Load)—Address: TMRC_BASE + $3
TMRC1_LOAD (Timer C, Channel 1 Load)—Address: TMRC_BASE + $B
TMRC2_LOAD (Timer C, Channel 2 Load)—Address: TMRC_BASE + $13
TMRC3_LOAD (Timer C, Channel 3 Load)—Address: TMRC_BASE + $1B

TMRD0_LOAD (Timer D, Channel 0 Load)—Address: TMRD_BASE + $3
TMRD1_LOAD (Timer D, Channel 1 Load)—Address: TMRD_BASE + $B
TMRD2_LOAD (Timer D, Channel 2 Load)—Address: TMRD_BASE + $13
TMRD3_LOAD (Timer D, Channel 3 Load)—Address: TMRD_BASE + $1B

TMR_BASE+$3, B,13,1B	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	LOAD															
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 14-9. Load Register (LOAD)

See Table A-8, List of Programmer's Sheets

This read/write register stores the value used to initialize the counter.

14.7.7 Hold Register (HOLD)

TMRA0_HOLD (Timer A, Channel 0 Hold)—Address: TMRA_BASE + $4
TMRA1_HOLD (Timer A, Channel 1 Hold)—Address: TMRA_BASE + $C
TMRA2_HOLD (Timer A, Channel 2 Hold)—Address: TMRA_BASE + $14
TMRA3_HOLD (Timer A, Channel 3 Hold)—Address: TMRA_BASE + $1C

TMRB0_HOLD (Timer B, Channel 0 Hold)—Address: TMRB_BASE + $4
TMRB1_HOLD (Timer B, Channel 1 Hold)—Address: TMRB_BASE + $C
TMRB2_HOLD (Timer B, Channel 2 Hold)—Address: TMRB_BASE + $14
TMRB3_HOLD (Timer B, Channel 3 Hold)—Address: TMRB_BASE + $1C

TMRC0_HOLD (Timer C, Channel 0 Hold)—Address: TMRC_BASE + $4
TMRC1_HOLD (Timer C, Channel 1 Hold)—Address: TMRC_BASE + $C
TMRC2_HOLD (Timer C, Channel 2 Hold)—Address: TMRC_BASE + $14
TMRC3_HOLD (Timer C, Channel 3 Hold)—Address: TMRC_BASE + $1C

TMRD0_HOLD (Timer D, Channel 0 Hold)—Address: TMRD_BASE + $4
TMRD1_HOLD (Timer D, Channel 1 Hold)—Address: TMRD_BASE + $C
TMRD2_HOLD (Timer D, Channel 2 Hold)—Address: TMRD_BASE + $14
TMRD3_HOLD (Timer D, Channel 3 Hold)—Address: TMRD_BASE + $1C

TMR_BASE+$4, C,14,1C	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	HOLD															
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 14-10. Hold Register (HOLD)

See Table A-8, List of Programmer's Sheets

This read/write register stores the counter's values of specific channels whenever any one of the four counters within a module is read.

14.7.8 Counter Register (CNTR)

TMRA0_CNTR (Timer A, Channel 0 Cntr)—Address: TMRA_BASE + $5
TMRA1_CNTR (Timer A, Channel 1 Cntr)—Address: TMRA_BASE + $D
TMRA2_CNTR (Timer A, Channel 2 Cntr)—Address: TMRA_BASE + $15
TMRA3_CNTR (Timer A, Channel 3 Cntr)—Address: TMRA_BASE + $1D

TMRB0_CNTR (Timer B, Channel 0 Cntr)—Address: TMRB_BASE + $5
TMRB1_CNTR (Timer B, Channel 1 Cntr)—Address: TMRB_BASE + $D
TMRB2_CNTR (Timer B, Channel 2 Cntr)—Address: TMRB_BASE + $15
TMRB3_CNTR (Timer B, Channel 3 Cntr)—Address: TMRB_BASE + $1D

TMRC0_CNTR (Timer C, Channel 0 Cntr)—Address: TMRC_BASE + $5
TMRC1_CNTR (Timer C, Channel 1 Cntr)—Address: TMRC_BASE + $D
TMRC2_CNTR (Timer C, Channel 2 Cntr)—Address: TMRC_BASE + $15
TMRC3_CNTR (Timer C, Channel 3 Cntr)—Address: TMRC_BASE + $1D

TMRD0_CNTR (Timer D, Channel 0 Cntr)—Address: TMRD_BASE + $5
TMRD1_CNTR (Timer D, Channel 1 Cntr)—Address: TMRD_BASE + $D
TMRD2_CNTR (Timer D, Channel 2 Cntr)—Address: TMRD_BASE + $15
TMRD3_CNTR (Timer D, Channel 3 Cntr)—Address: TMRD_BASE + $1D

TMR_BASE+$5, D,15,1D	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	COUNTER															
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 14-11. Counter (CNTR)

See Table A-8, List of Programmer's Sheets

This read/write register is the counter for the corresponding channel in a timer module.

14.8 Timer Group A, B, C, and D Functionality

Some of the input and output pins of the timers are shared with other peripheral modules on the 56F80x. This section identifies the input/ouput pins associated with each timer group. If a timer is not used for external input/output, it is available for internal use as well.

Note: Individual timers often use their own I/O pin but they may use any available pin within their group as an input. The timers are limited to their own I/O pin for use as an output pin.

14.8.1 Timer Group A (56F803, 56F805, and 56F807 Only)

Timer group A shares pins with quad decoder module zero. The decoder has primary ownership of the pins for inputs. The Decoder Control register for Quad Decoder Zero (DEC0_DECCR) controls a switch matrix that connects timer A inputs to the I/O pins, provides filtered input data, or a remapped data format. Timer group A outputs are directly connected to the I/O pins. If a timer's output is enabled, its output will drive the I/O pin. The Decoder module does not use the I/O pins for outputs.

Note: The Quad Decoder Zero module contains a quad test signal generator capable of being monitored by the Timer Module on Timer Zero (PHASEA0) input and the Timer One (PHASEB0) input.

Timer A, counter zero input/output pin is shared with Quad Decoder Zero PHASEA0 pin.

- Timer A, counter one I/O pin is shared with quad decoder one PHASEB0 pin.
- Timer A, counter two I/O pin is shared with quad decoder zero INDEX0 pin.
- Timer A, counter three I/O pin is shared with quad decoder zero HOME0 pin.

14.8.2 Timer Group B (56F805 and 56F807 Only)

Timer Group B shares pins with Decoder Module One. The decoder has primary ownership of the pins for inputs. The Decoder Control Register for Quad Decoder One, QD1_DECCR,

controls a switch matrix connecting Timer B inputs to the I/O pins, provides filtered input data, or a remapped data format. The Timer Group B outputs are directly connected to the I/O pins. If a timer's output is enabled, its output will drive the I/O pin. The Decoder module does not use the I/O pins for outputs.

- The Quad Decoder One module contains a quad test signal generator can be monitored by the Timer module on the Timer Zero (PHASEA0) input and the Timer One (PHASEB0) input.
- Timer B, counter zero I/O pin is shared with Quad Decoder One PHASE0 pin.
- Timer B, counter one I/O pin is shared with Quad Decoder One PHASE1 pin.
- Timer B, counter two I/O pin is shared with Quad Decoder One INDEX1 pin.
- Timer B, counter three I/O pin is shared with Quad Decoder One HOME1 pin.

14.8.3 Timer Group C

14.8.3.1 56F805 and 56F807 Only

- Timer Group C counters zero and one have dedicated I/O pins
- Timer Group C counter zero has its own dedicated input/output pin TC0
- Timer Group C counter one has its own dedicated input/output pin TC1

14.8.3.2 56F801, 56F802, 56F803, 56F805, and 56F807

Timers two and three do not have dedicated I/O pins. They can accept inputs from the PWM modules, and their outputs are connected to the ADC modules.

Note: Timers two and three can use the input pins allocated to Timers Zero and One. Their outputs can be used by Timers Zero and One. The intended purpose of timers two and three is to provide a user selectable delay, One-shot mode, between the PWM sync signal and the updating of the ADC values.

- Timer Group C, counter two input is connected to the PWM A sync output
- Timer Group C, counter two output is connected to the ADC A sync input
- Timer Group C, counter three input is connected to the PWM B sync output
- Timer Group C, counter three output is connected to the ADC B sync input

14.8.4 Timer Group D

Three of the four timers in group D have their own dedicated I/O pins, varying in number between the 56F80x configurations.

14.8.4.1 567F801 Only

- Timer Group D, counter zero has its own dedicated input/output pin TD0
- Timer Group D, counter one has its own dedicated input/output pin TD1
- Timer Group D, counter two has its own dedicated input/output pin TD2

Note: For the 56F801, the timer group D pins are multiplexed with GPIO pins so TD0 is muxed to GPIOA0, TD1 is muxed to GPIOA1, TD2 is muxed to GPIOA2. When the quad timer is not required, it offers additional GPIO.

14.8.4.2 567F802 Only

- Timer Group D, counter one has its own dedicated input/output pin TD1
- Timer Group D, counter two has its own dedicated input/output pin TD2

Note: For the 56F802, the timer group D pins are multiplexed with GPIO pins so TD1 is muxed to GPIOA1, and TD2 is muxed to GPIOA2. When the quad timer is not required, it offers additional GPIO.

14.8.4.3 56F803 Only

- Timer Group D, counter one has its own dedicated input/output pin TD1
- Timer Group D, counter two has its own dedicated input/output pin TD2

14.8.4.4 56F805 and 56F807 Only

- Timer Group D, counter zero has its own dedicated input/output pin TD0
- Timer Group D, counter one has its own dedicated input/output pin TD1
- Timer Group D, counter two has its own dedicated input/output pin TD2
- Timer Group D, counter three has its own dedicated input/output pin TD3

14.8.4.5 General Input Behavior

Time inputs are sampled at the peripheral clock rate. There is a delay of one peripheral clock period before the input can affect the behavior of a counter. This is typical of synchronous counters systems. There is one exception to this statement: when the counter is gated.

Table 14-4. Document Revision History for Chapter 14

Version History	Description of Change
Rev. 8	Formatting, layout, spelling, and grammar corrections. Added revision history table. In Section 14.7.1.4 (CTRL[ONCE] bit description), replaced the text "...the timer is stopped by changing the timer's Count Mode to Stop Mode (CM=0)" with "...the timer module changes its Count Mode to Stop Mode (Count Mode = 0)". In **Figure 14-8**, changed field name (was "Capture value", is CAPTURE). In **Figure 14-9**, changed field name (was "LOAD VALUE", is LOAD).

Chapter 15
On-Chip Clock Synthesis (OCCS)

15.1 Introduction

The On-Chip Clock Synthesis (OCCS) allows product design using inexpensive 8MHz crystals to run the device at any user selectable multiple from one to 80Mhz.

All peripherals on 56F80x devices, except the COP/Watchdog Timer, run off the IPBus clock frequency. It is the chip operating frequency (ZCLK) divided by two. The maximum frequency of operation is 80MHz correlating to a 40MHz IPBus clock frequency.

15.2 Features

The On-Chip Clock Synthesis (OCCS) module interfaces to the oscillator and PLL with an on-chip prescaler. The OCCS module features are:

- 2-bit prescaler
- 2-bit postscaler
- Ability to power-down the internal PLL
- Selectable PLL source clock
- Selectable System Clock (ZCLK) from three sources

The Clock Generation module provides the programming interface for both the PLL and on- and off-chip oscillators.

15.3 Pin Descriptions

15.3.1 Oscillator Inputs (XTAL, EXTAL)

The oscillator inputs can be used to connect an external crystal or to directly drive the chip with an external clock source, thus bypassing the internal oscillator circuit. Design considerations for the External Clock mode of operation are discussed in Section 15.3.2, “External Crystal Design Considerations”.

15.3.2 External Crystal Design Considerations

Either an external crystal oscillator or an external frequency source can be used to provide a reference clock to the 56F80x. The 56F802 device can only be clocked from the onchip relaxation oscillator.

15.3.2.1 Crystal Oscillator

The Internal Crystal Oscillator circuit is designed to interface with a parallel-resonant crystal resonator in the frequency range, specified for the external crystal, is 4 to 8MHz. **Figure 15-1** illustrates a typical crystal oscillator circuit. Follow the crystal supplier's recommendations when selecting a crystal, since crystal parameters determine the component values required to provide maximum stability and reliable start-up. The load capacitance values used in the oscillator circuit design should include all stray layout capacitances. The crystal and associated components should be mounted as close as possible to the EXTAL and XTAL pins to minimize output distortion and start-up stabilization time.

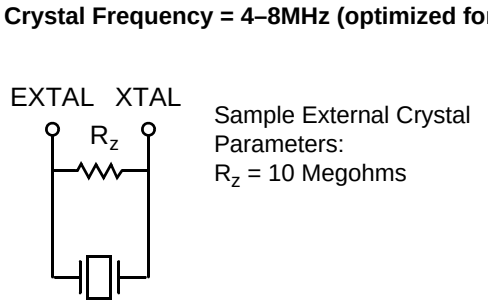


Figure 15-1. External Crystal Oscillator Circuit

15.3.2.2 External Clock Source

The recommended method of connecting an external clock is provided in **Figure 15-2.** The External Clock Source is connected to XTAL and the EXTAL pin is grounded.

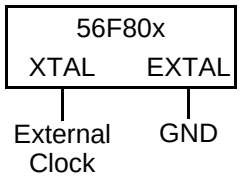


Figure 15-2. Connecting an External Clock Signal Using XTAL

It is possible to drive EXTAL with an external clock, though *this is not the recommended method.* To drive EXTAL with an External Clock Source the following conditions must be met:

- XTAL must be completely unloaded

- Maximum frequency of the applied clock must be less than 8MHz

Figure 15-3 illustrates how to connect an external clock circuit with an external clock source using EXTAL as the input.

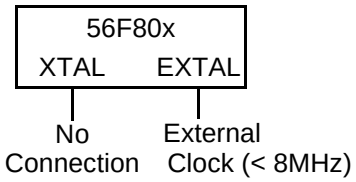
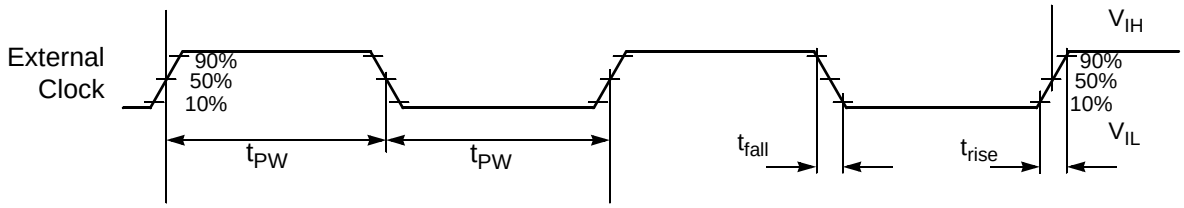


Figure 15-3. Connecting an External Clock Signal Using EXTAL



Note: The midpoint is $V_{IL} + (V_{IH} – V_{IL})/2$.

Figure 15-4. External Clock Timing

Table 15-1. External Clock Operation Timing Requirements

Operating Conditions: $V_{SS} = V_{SSA} = 0$ V, $V_{DD} = V_{DDA} = 3.0–3.6$ V, $T_A = – 40 \times$ to $+ 85 \times$ C

Characteristic	Symbol	Min	Typ	Max	Unit
Frequency of Operation (External Clock Driver)[1]	f_{osc}	4	8	8	MHz
Clock Pulse Width[2]	t_{PW}	6.25	—	—	ns
External Clock Input Rise Time[3]	t_{rise}	—	—	3	ns
External Clock Input Fall Time[4]	t_{fall}	—	—	3	ns

1. See specific product datasheet for details on using the external clock driver.
2. The high or low pulse width must be no smaller than 6.25ns or the chip will not function.
3. External clock input rise time is measured from 10% to 90%.
4. External clock input fall time is measured from 90% to 10%

15.4 Functional Description

This section describes the clocking methods available on the 56F80x devices.

Figure 15-5 illustrates the types of acceptable reference clocks. These include, from left to right:

- Crystal oscillator, uses EXTAL and XTAL pins
- External clock source, uses EXTAL pin
- Internal Relaxation Oscillator, available only on the 56F801 and 56F802 devices

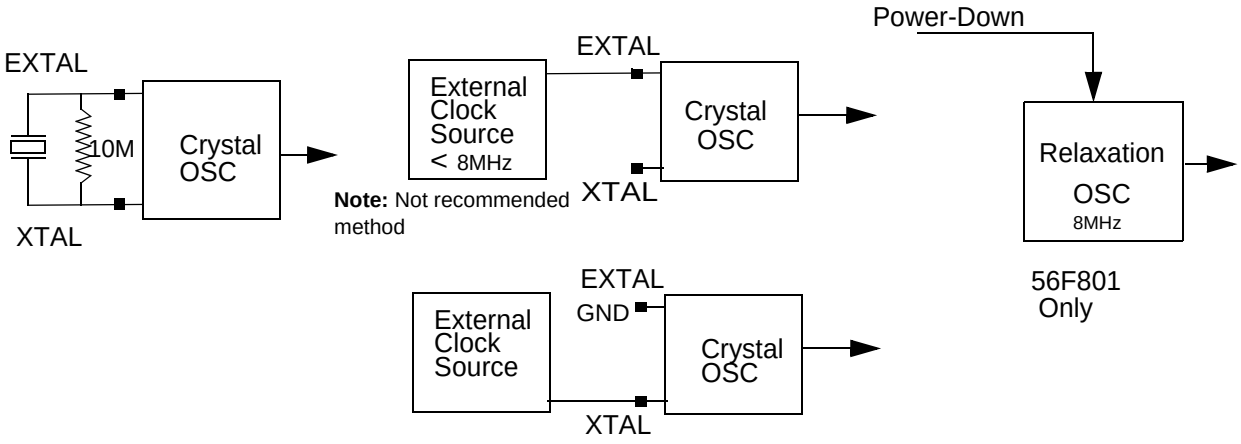


Figure 15-5. Reference Clock Sources

The 56F803/805/807 chips allow the use of the external crystal oscillator or external clock source options only. The 56F801 accepts these same options, as well as use of the Internal Chip Relaxation Oscillator.

The Internal Oscillator available on the 56F801/802 devices have very little variability with temperature and voltage, but it does vary as a function of wafer fabrication process. This On-Chip Relaxation Oscillator reaches a stable frequency very quickly, well under 1µs. During the 56F801/802 reset sequence, the Internal Oscillator is activated by default. Application code can be used to switch from the On-Chip Relaxation Oscillator to the Crystal Oscillator or External Source, powering down the Internal Oscillator if desired. A block diagram of the OCCS module is shown in **Figure 15-6.**

.

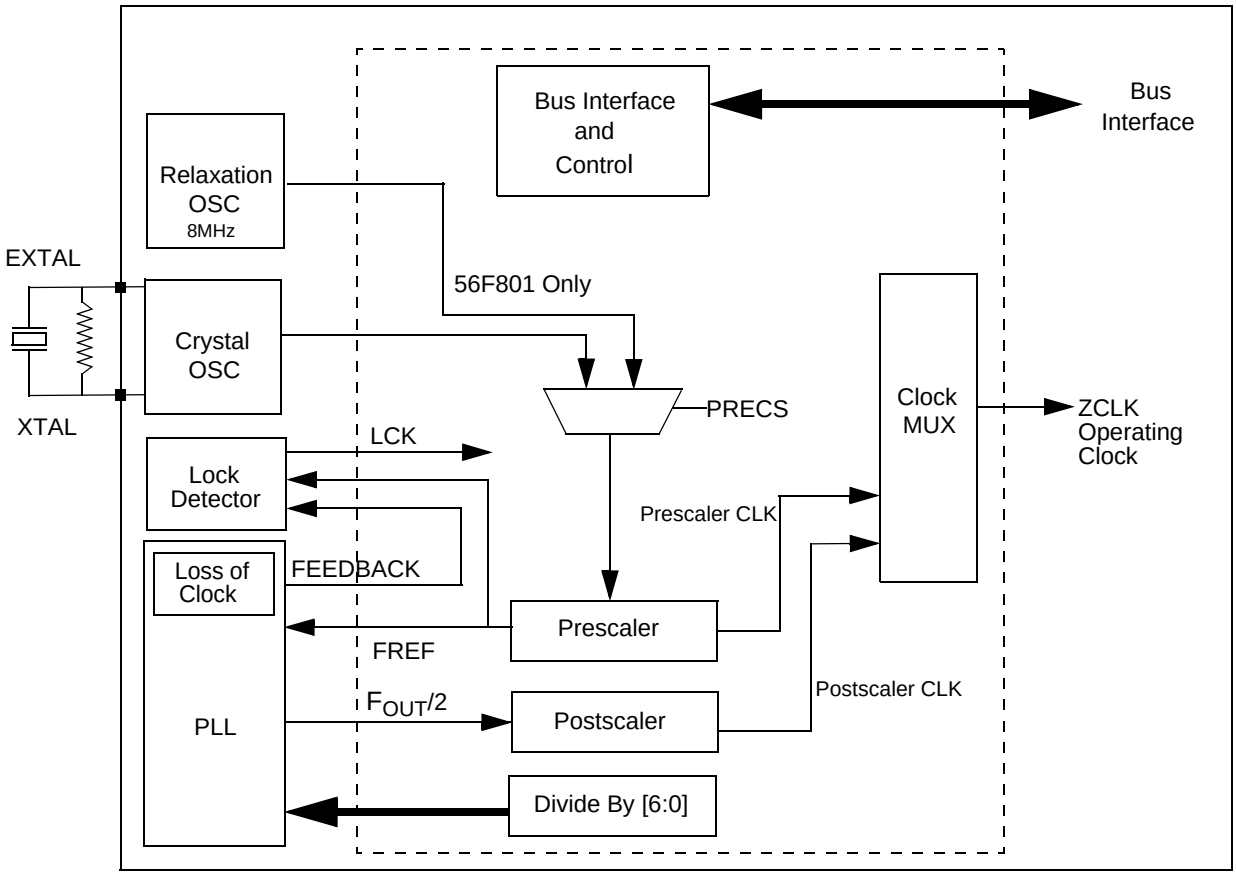


Figure 15-6. OCCS Block Diagram

The Clock Multiplexer (MUX) selects the OSC clock on power-up. A different clock source can be selected by writing to the PLL Control Register (PLLCR). Once a new clock source is selected, the new clock will be activated within four clock periods of the new clock after the clock selection request is re-clocked by the current IPBus clock.

Possible clock source choices are:

- Crystal oscillator clock, derived from either a clock fed into the EXTAL pin, or an external crystal connected between the EXTAL and XTAL pins
- Crystal oscillator clock adjusted by the prescaler
- PLL clock

The PLL uses the crystal oscillator clock or divided version from the prescaler for deriving its clock.

Frequencies going into and out of the PLL are controlled by the prescaler, postscaler, and the divide-by ratio within the PLL. For proper operation of the PLL, the user must keep the VCO, within the PLL, in its operational range of 80 - 240MHz, the output of the VCO is depicted as

F_{OUT} in **Figure 15-14.** The input frequency divided by the prescaler ratio, multiplied by the divide-by ratio, is the frequency at which the PLL is running.

The PLL lock time is 10ms or less when coming from a powered down state to a power-up state. It is recommended when changing the prescaler ratio, or the divide-by ratio, powering down or powering up, the PLL be deselected as the clocking source. Only after lock is achieved should the PLL be used as a valid clocking source.

15.4.1 Timing

A timing diagram for changing clock source from the PLL clock to the prescaler clock is shown in **Figure 15-7.**

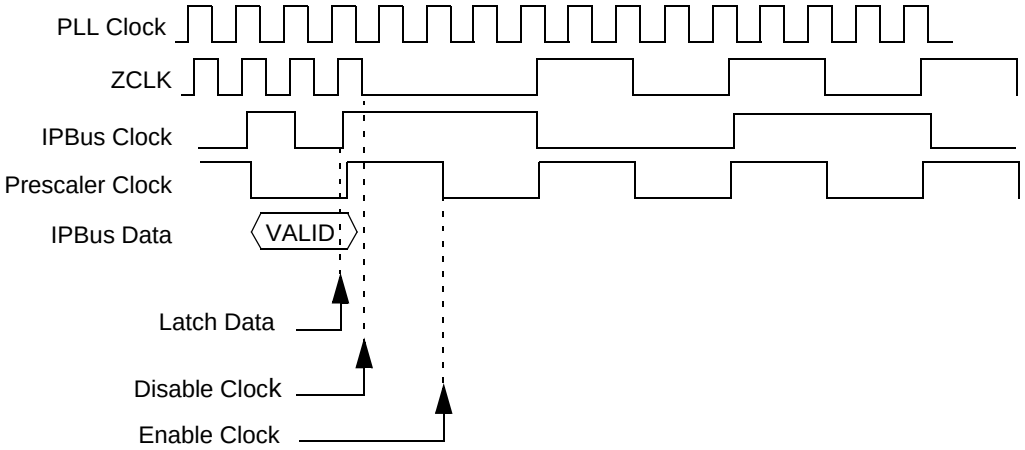


Figure 15-7. Changing Clock Sources

The clock from the PLL to the core is selected by writing to the PLL Control Register (PLLCR) ZSRC [1:0] bit. The clock to the core is selected by writing to the ZCLOCK (ZSRC) source bits in the PLLCR. Before changing the divide-by value for the PLL, it is recommended the core clock first be switched to the prescaler clock. After the PLL has locked, the core clock can be switched back to the PLL by writing to the ZSRC bits in the PLLCR.

When a new core clock is selected, the Clock Generation module will synchronize the request and select the new clock. The PLL Status Register (PLLSR) shows the status of the core clock source. Because the synchronizing circuit changes modes to avoid any glitches, the PLLSR ZCLOCK (ZSRC) source will show overlapping modes as an intermediate step.

15.5 Register Definitions

Table 15-2. OCCS Memory Map

Device	Peripheral	Address
801/802/ 803/805	CLKGEN_BASE	$0FA0
807	CLKGEN_BASE	$13A0

The address of a register is the sum of a base address and an address offset. The base address is defined at the MCU level and the address offset is defined at the module level. Please refer to **Table 3-11** for CLKGEN_BASE definition.

Table 15-3. OCCS Register Summary

Address Offset	Register Acronym	Register Name	Access Type	Register Location
Base + $0	PLLCR	Control Register	Read/Write	Section 15.5.1
Base + $1	PLLDB	Divide-By Register	Read/Write	Section 15.5.2
Base + $2	PLLSR	Status Register	Read/Write	Section 15.5.3
		Reserved		
Base + $4	CLKOSR	CLKO Select Register	Read/Write	Section 15.5.4
Base + $5	IOSCTL	Internal Oscillator Control Reg.	Read/Write	Section 15.5.5

Bit fields of the five registers are summarized in **Figure 15-8.** Details of each follow.

Add. Offset	Register Name		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$0	PLLCR	R	PLLIE1		PLLIE0		LOCIE	0	0	0	LCKON	CHPMPTRI	0	PLLPD	0	PRECS	ZSRC	
		W																
$1	PLLDB	R	LORTP				PLLCOD		PLLCID		0	PLLDB						
		W																
$2	PLLSR*	R	LOLI1	LOLI0	LOCI	0	0	0	0	0	0	LCK1	LOK0	PLLPDN	0	PRECSS	ZSRC	
		W																
$2	PLLSR**	R	LOLI1	LOLI0	LOCI	0	0	0	0	0	0	LCK1	LOK0	PLLPDN	0	0	ZSRC	
		W																
RESERVED																		
$4	CLKOSR	R	0	0	0	0	0	0	0	0	0	0	0	CLKOSEL				
		W																
$5	IOSCTL	R	0	0	0	0	0	0	0	0	0	TRIM						
		W																

*801/802 only. **56F803/805/807 is reserved.

R 0 Read as 0
W Reserved

Figure 15-8. OCCS Register Map Summary

15.5.1 PLL Control Register (PLLCR)

CLKGEN_BASE+$0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	PLLIE1		PLLIE0		LOCIE	0	0	0	LCKON	CHPMPTRI	0	PLLPD	0	PRECS	ZSRC	
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1

Figure 15-9. PLL Control Register (PLLCR)

See Table A-8, List of Programmer's Sheets

The loss of clock circuit monitors the output of the OCCS. In the event of loss of clock, an optional interrupt can be generated.

15.5.1.1 PLL Interrupt Enable 1 (PLLIE1)—Bits 15–14

An optional interrupt can be generated when the PLL Lock Status (LCK1) bit in the PLL Status Register (PLLSR) changes:

- 00 = Disable interrupt
- 01 = Enable interrupt on any rising edge of LCK1
- 10 = Enable interrupt on falling edge of LCK1
- 11 = Enable interrupt on any edge change of LCK1

15.5.1.2 PLL Interrupt Enable 0 (PLLIE0)—Bits 13–12

An optional interrupt can be generated if the PLL Lock Status (LCK0) bit in the PLL Status Register (PLLSR) changes:

- 00 = Disable interrupt
- 01 = Enable interrupt on any rising edge of LCK0
- 10 = Enable interrupt on falling edge of LCK0
- 11 = Enable interrupt on any edge change of LCK0

15.5.1.3 Loss of Clock Interrupt Enable (LOCIE)—Bit 11

An optional interrupt can be generated if the oscillator circuit output clock is lost:

- 0 = Interrupt disabled
- 1 = Interrupt enabled

15.5.1.4 Reserved—Bits 10–8

This bit field is reserved or not implemented.The bits may be read/written with 0s.

15.5.1.5 Lock Detector On (LCKON)—Bit 7

- 0 = Lock detector disabled
- 1 = Lock detector enabled

15.5.1.6 Charge Pump Tri-state (CHPMPTRI)—Bit 6

During normal chip operation the CHPMTRI bit should be set to a value of zero. In the event of loss of clock reference the CHPMTRI bit must be set to a value of one. A value of one isolates the charge pump from the loop filter allowing the PLL output to slowly drift providing enough time to shut down the chip. Activating this bit will render the PLL inoperable and should not be completed during standard operation of the chip.

Additionally, this bit is used for isolating the charge pump from the loop filter so the filter voltage can be driven from an external source during bench test evaluations.

15.5.1.7 Reserved—Bit 5

This bit is reserved or not implemented. It is read as 0 and cannot be modified by writing.

15.5.1.8 PLL Power-Down (PLLPD)—Bit 4

The PLL can be turned off by setting the PLLPD bit. There is a four IPBus clock delay from changing the bit to signaling the PLL. When the PLL is powered down, the gear shifting logic automatically switches to ZSRC[1:0] = 1b in order to prevent loss of clock to the core.

- 0 = PLL enabled
- 1 = PLL powered down

15.5.1.9 Reserved—Bit-3

This bit is reserved or not implemented. It is read as 0 and cannot be modified by writing.

15.5.1.10 Prescaler Clock Select (PRECS)—Bit 2

This bit is reserved (value 0) for the 56F803/802/805/807. The following description is applicable only to the 801/802 devices.

This bit will not be allowed to be set unless the crystal oscillator is enabled in the GPIO. The clock source to the prescaler can be selected to be either the internal relaxation oscillator or the crystal oscillator. The PRECS bit is automatically set to 0b at reset.

- 0 = Relaxation oscillator selected
- 1 = Crystal oscillator selected

15.5.1.11 ZCLOCK Source (ZSRC)—Bits 1–0

The ZCLOCK source determines the clock source to the core. The ZCLOCK is also the source of the IPBus clock. ZSRC is automatically set to 01b during STOP_MODE, or if PLLPD is set in order to prevent loss of clock to the core. The 56F80x ZSRC may have the following values:

- 01 = Prescaler output
- 10 = Postscaler output

15.5.2 PLL Divide-By Register (PLLDB)

CLKGEN_BASE+$1	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	LORTP				PLLCOD		PLLCID		0	PLLDB						
Write																
Reset	0	0	1	0	0	0	0	0	0	0	0	1	0	0	1	1

Figure 15-10. PLL Divide-By Register (PLLDB)

See Table A-8, List of Programmer's Sheets

15.5.2.1 Loss of Reference Timer Period (LORTP)—Bits 15-12

These bits control the amount of time required for the loss of reference interrupt to be generated. This failure detection time is LORTP × 10 × PLL-clock-time-period.

15.5.2.2 PLL Clock-Out-Divide (PLLCOD)—Bits 11–10

The PLL output clock can be divided down by a 2-bit postscaler as shown in **Figure 15-14.**

- 00 = divide by one
- 01 = divide by two
- 10 = divide by four
- 11 = divide by eight

15.5.2.3 PLL Clock-In-Divide (PLLCID)—Bits 9–8

The PLL input clock, EXTAL_CLK, **Figure 15-14,** can be divided down by a 2-bit prescaler. The output of the prescaler is a selectable clock source for the core as determined by the ZSRC[1:0] in the PLLCR.

- 00 = Divide by one
- 01 = Divide by two
- 10 = Divide by four
- 11 = Divide by eight

15.5.2.4 Reserved—Bit 7

This bit is reserved or not implemented. It is read as 0 and cannot be modified by writing.

15.5.2.5 PLL Divide-By (PLLDB)—Bits 6–0

The output frequency of the PLL is controlled, in part, by the PLLDB[6:0] register. The value written to this register, plus one, is used by the PLL to directly multiply the input frequency and present it at its output. For example, if the input frequency is 8MHz and the PLLDB[6:0] register is set to 14, then the PLL output frequency is 120MHz.

Using the default bits illustrated in **Figure 15-9,** the value is (19 +1), or 20. For an 8MHz EXTAL_CLK, seen in **Figure 15-14,** this default value sets the output of the PLL, F_{OUT}, to 160MHz, resulting in an $F_{OUT}/2$ of 80MHz. Before changing the divide-by value, it is recommended the core clock be first switched to the prescaler clock.

Note: Upon writing to the PLLDB register, the lock detect circuit is reset.

15.5.3 PLL Status Register (PLLSR)

CLKGEN_BASE+$2	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	LOLI1	LOLI0	LOCI	0	0	0	0	0	0	LCK1	LCK0	PLLPDN	0	PRECSS*	ZSRC	
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1

Figure 15-11. PLL Status Register (PLLSR)

See Table A-8, List of Programmer's Sheets

* Applicable only to 56F801. This bit is Reserved in 56F802/803/805/807.

The Loss of Lock Interrupt is cleared by writing 1 to the respective LOCI bit in the PLLSR. A PLL interrupt is generated if any of the LOLI or LOCIE bits are set and the respective Interrupt Enable is set in the PLLCR.

15.5.3.1 PLL Loss of Lock Interrupt 1 (LOLI1)—Bit 15

This bit is cleared by writing one to the LOLI1 bit.

- 0 = No interrupt pending
- 1 = Interrupt pending

15.5.3.2 PLL Loss of Lock Interrupt 0 (LOLI0)—Bit 14

- 0 = No interrupt pending
- 1 = Interrupt pending

15.5.3.3 Loss of Clock (LOCI)—Bit 13

The Loss of Clock Interrupt is cleared by writing 1 to the respective LOCI bit in the PLLSR.

- 0 = Oscillator clock normal
- 1 = Lost oscillator clock

15.5.3.4 Reserved—Bits 12–7

This bit field is reserved or not implemented. It is read as 0 and cannot be modified by writing.

15.5.3.5 Loss of Lock 1 (LCK1)—Bit 6

- 0 = PLL is unlocked (fine)
- 1 = PLL is locked (fine)

15.5.3.6 Loss of Lock 0 (LCK0)—Bit 5

- 0 = PLL is unlocked (course)
- 1 = PLL is locked (course)

15.5.3.7 PLL Power-Down (PLLPDN)—Bit 4

PLL power-down status is delayed by four IPBus clocks from the PLLPD bit in the PLLCR.

- 0 = PLL not powered down
- 1 = PLL powered down

15.5.3.8 Reserved—Bit 3

This bit is reserved or not implemented. It is read as 0 and cannot be modified by writing.

15.5.3.9 Prescaler Clock Status Source Register (PRECSS)—Bit 2

This is a reserved bit for only 801/802.

- 0 = Relaxation oscillator selected
- 1 = Crystal oscillator selected

15.5.3.10 ZCLOCK Source (ZSRC)—Bits 1–0

This bit is reserved with a zero value for the 803/805/807. This description is applicable only to the 801/802 devices.

ZSRC indicates the current ZCLOCK source. Since the synchronizing circuit switches the ZCLOCK source, ZSRC takes more than one IPBus clock to indicate the new selection.

- 00 = Synchronizing in progress
- 01 = Prescaler output
- 10 = Postscaler output
- 11 = Synchronizing in progress

15.5.4 CLKO Select Register (CLKOSR)

The CLKO Select Register can be used to multiplex out any one of the clocks generated inside the Clock Generation module. See **Figure 15-12.** The default value is ZCLK. The ZCLK is the

processor clock and should be used for any external memory accesses requiring a clock. This path has been optimized in order to minimize any delay and clock duty cycle distortion. All other clocks possibly muxed out are for test purposes only. This register is applicable to all devices.

CLKGEN_BASE+$4	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	0	0	0	0	0	0	CLKOSEL				
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 15-12. CLKO Select Register (CLKOSR)

See Table A-8, List of Programmer's Sheets

15.5.4.1 Reserved—Bits 15–5

This bit field is reserved or not implemented. It is read as 0 and cannot be modified by writing.

15.5.4.2 CLKO Select (CLKOSEL)—Bits 4–0

Selects clock to be muxed out on the CLKO pin.

- 10000 = No clock
- 00000 = ZCLK (DEFAULT)
- 00001 = reserved for factory test—t0
- 00010 = reserved for factory test—t1
- 00011 = reserved for factory test—t2
- 00100 = reserved for factory test—t3
- 00101 = reserved for factory test—phi0
- 00110 = reserved for factory test—phi1
- 00111 = reserved for factory test—ctzn
- 01000 = reserved for factory test—ct301en
- 01001 = reserved for factory test—1PB clock
- 01010= reserved for factory test—Feedback
- 01011 = reserved for factory test—Prescaler Clk
- 01100 = reserved for factory test—F_{OUT}
- 01101 = reserved for factory test—$F_{OUT}/2$

- 01110 = reserved for factory test—Postscaler Clk
- 01111 = reserved for factory test—Postscaler Clk

15.5.5 56F801/802 Internal Oscillator Control Register (IOSCTL)

The TRIM[7:0] read/write bits of this register change the size of the internal capacitor used by the Internal Relaxation Oscillator. By testing the frequency of the internal clock and incrementing or decrementing this factor accordingly, the accuracy of the internal clock can be improved to two percent. **Figure 15-13** illustrates an Internal Oscillator Control (IOSCTL) register. This register is applicable to 56F801/802 only.

56F801 Internal Oscillator Control Register (IOSCTL) CLKGEN_BASE+$5	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	0	0	0	0	TRIM							
	Write																
	Reset	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

Figure 15-13. Internal Oscillator Control Register (IOSCTL)

See Table A-8, List of Programmer's Sheets

15.5.6 56F801 Clock Switch Over Procedure

Reset or power-up configures the 56F801 chip to use the Internal Relaxation Oscillator. To switch the clock from Internal Relaxation Oscillator to an External Crystal Oscillator clock, exercise care to satisfy the 56F801clock switch-over requirement.

This is the recommended clock switch-over procedure:

- Disable the pull-up resistors for the EXTAL pin and XTAL pin
- Wait through the period for the crystal oscillator to stabilize
- Switch-over to the external crystal oscillator
- Enable PLL lock detect circuit and enable PLL
- Wait for PLL to lock
- Switch over to PLL postscaler clock

15.5.7 56F801 Disabling EXTAL and XTAL Pull Up Resistors

The EXTAL and XTAL pull resistors are controlled by GPIOB Pull-Up Enable Register (PUR) bits two and three. GPIOB pull resistors are enabled by default after reset or power-up. This is applicable to 56F801 only.

Warning: Do not enable GPIOB pull-up resistors for bits two and three (EXTAL

and XTAL pins). Ensure these bits are not configured or changed when external crystal oscillator is being used as a clock source for the 56F801.

15.5.8 External Crystal Oscillator Signal Generation

The correct operation of the 56F801 chips initialization depends on the reset input signal and the stability of the External Clock Oscillator. The Crystal Clock Oscillator may take up to 20ms or more before reaching the correct operating frequency. Different crystal oscillators have different stabilization requirements. Please consult the manufacturer's recommended specifications. The 56F801 reset signal can be held during the External Crystal Oscillator stabilization time, and before gating it to the 56F801.

Warning: Clock switching over from External Crystal Oscillator to the Internal Crystal Oscillator is not supported.

15.5.9 TRIM[7:0] – Internal Relaxation Oscillator TRIM Bits

The TRIM bits can be calibrated to the Internal Crystal Oscillator frequency. Incrementing these bits by one decreases the frequency by 0.23 percent of the unadjusted value. Decremented these bits by one increases the frequency by 0.23 percent of the unadjusted value. Reset these bits to $80, centering the range of possible adjustment.

Additionally, because the Internal Relaxation Oscillator frequency is dependent on manufacturing, the value may not be exactly 8MHz. Therefore, a TRIM value (calculated at probe) will be stored in the information Block of the Data Flash. This value will be calculated for room temperature (approximately 25°C). It is extremely important NOT to erase this value.

To gain access to the TRIM value:

1. Set only the IFREN bit of the DFIU_CNTL register to enable the Information Block, bit six of address $0F60.
2. Read TRIM value from data address $0F60.
3. Write the whole TRIM value to the IOSCTL register at $0F15.
4. Clear the IFREN bit of the DFIU_CNTL register to disable the Information Block.

This is the ONLY place the TRIM value is stored. The DFIU IFREN bit should only be set when reading the TRIM value. The IFREN bit should not be set when erasing the data FLAH or the TRIM value will be lost.

By storing the value in the information block, it is separated from other FLASH data and can be kept intact while erasing the main 2K portion of the data FLASH.

Note: Only the 56F801 has an External Relaxation Oscillator of 8MHz. Please refer to **Figure 15-14.**

15.5.10 Clock Operation in the Power-Down Modes

The 56F80x operate in one of three Power-Down modes:

1. Run
2. Wait
3. Stop

In the Run mode, all clocks are active. In Wait mode, all of the chip's clocks are active except those internal to the 5680x core itself. All peripheral devices are still active and able to produce interrupt requests. When an interrupt is not masked off, assertion will cause the core to come out of its suspended state, resuming normal operation. *Wait is typically used for power conscious applications.*

Stop mode causes ZCLK to shift-down to the OSC_CLK frequency, keeping the COP Timer alive, but typically running much slower, and the IPBUS_CLOCK to turn-off. Like Wait, the Stop mode causes all clocks internal to the 5680x core to shutdown, thereby suspending processing. The only possible recoveries from the Stop mode are:

1. COP Timeout Interrupt
2. CAN Interrupt
3. Hardware Reset
4. Power-On Reset

A typical application example is demonstrated in **Figure 15-14.** Usually an inexpensive 8MHz crystal is selected to provide a stable time reference for the system. Determine the PLL by selecting a prescaler value to be written to the PLLCID register. A recommended prescaler value is one. The output frequency of the prescaler is the input frequency to the PLL. This output frequency to the PLL is multiplied by the (PLLDB + 1) value. The result is written to the PLLDB[6:0] register. The resulting frequency is depicted as F_{OUT} in **Figure 15-14.** Divide the frequency by two through a fixed divider at the output of the PLL prior to being presented to the postscaler divider. The postscaler divider further apportions the frequency by its user-defined value before presenting it as ZCLK through the sync mux. Setting the frequency using the programmable divider is discussed in **Section 15.5.2.5, "PLL Divide-By (PLLDB)—Bits 6–0".**

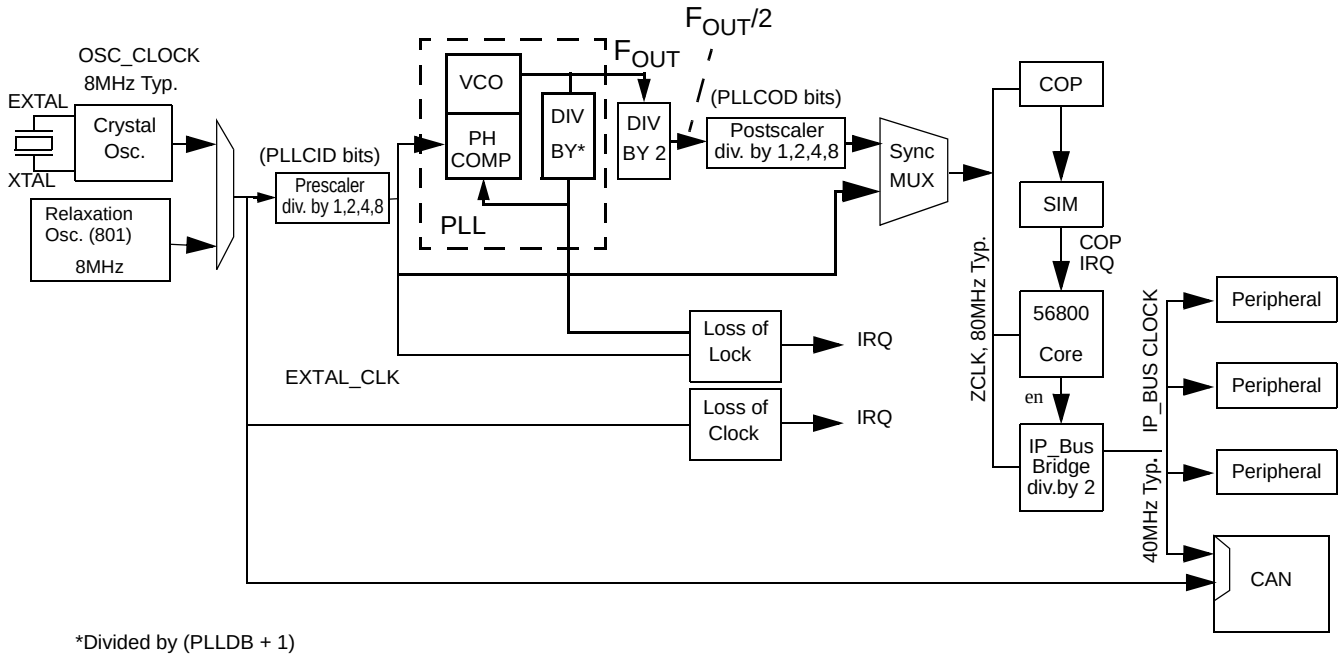


Figure 15-14. Relationship of IPBus Clock and ZCLK

Note: The maximum frequency of operation is 80MHz correlating to a 40MHz IPBus clock frequency.

The CAN module can elect to use the OSC_CLOCK frequency directly for CAN applications requiring extraordinary jitter constraints, or for CAN operation during the Stop mode.

The following table summarizes the state of the on-chip clocks in typical operating frequencies during various power-down modes.

Table 15-4. On-Chip Clock States

State	ZCLK	IPBUS_CLK
Run	80MHz	40MHz
Wait	80MHz	40MHz
Stop	8MHz	Off

Note: All peripherals, except the COP/Watchdog Timer, run off of the IPBus clock frequency. That frequency is the chip operating frequency divided-by two. The maximum frequency of operation is 80MHz, correlates to a 40MHz IPBus clock frequency.

15.5.11 PLL Recommended Range of Operation

The PLL's Voltage Controlled Oscillator (VCO) within the PLL has a characterized operating range extending from 80MHz to 240MHz. The output of the PLL, F_{OUT} in **Figure 15-14** is followed by a hard wired divide-by two circuit, yielding a 40MHz to 120 MHz clock at the input of the postscaler. The PLL is programmable via a divide by *n+1* register, capable of taking on values varying between one and 128. PLLDB bits determine this value, referenced in **Section 15.5.3, "PLL Status Register (PLLSR)".** For higher values of *n*, PLL lock time becomes an issue. It is recommended to avoid values of *n* resulting in the VCO frequency greater than 240MHz or less than 80MHz. **Figure 15-15** illustrates the recommended range of available output frequencies from the PLL verses the input frequency. Generally speaking, the lower the value of *n*, the quicker the PLL will be able to lock.

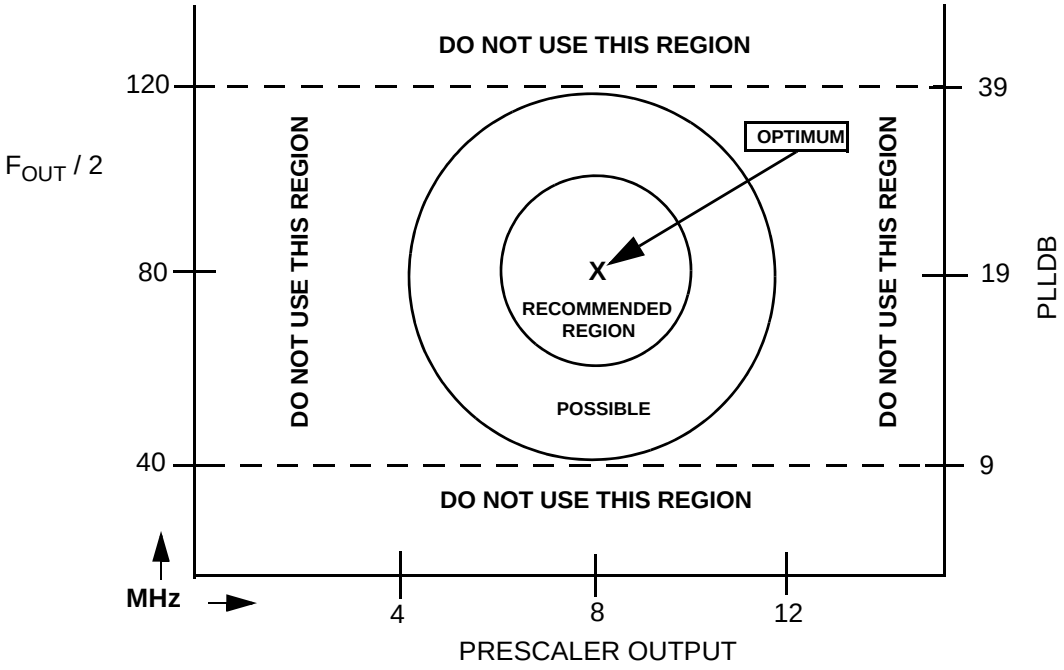


Figure 15-15. Recommended Design Regions of OCCS PLL Operation

15.6 PLL Lock Time Specification

In many applications, the Lock Time of the PLL is the most critical PLL design parameters. Proper design and use of the PLL ensures the highest stability and lowest lock time.

15.6.1 Lock Time Definition

Typical control systems refer to the Lock Time as the reaction time within specified tolerances of the system to a step input. In a PLL, the step input occurs when the PLL is turned on, or when it

suffers a noise hit. The tolerance is usually specified as a percent of the step input, or when the output settles to the desired value plus, or minus a percent of the frequency change. Therefore, the reaction time is constant in this definition, regardless of the size of the step input.

When the PLL is coming from a powered down state, PLL_PDN high, to a powered up condition, PLL_PDN low, the maximum Lock Time, with a divide-by count of 16 or less, is 10ms. Other systems refer to Lock Time as the time the system takes to reduce the error between the actual output and the desired output to within specified tolerances. Therefore, the Lock Time varies according to the original error in the output. Minor errors may be shorter or longer in many cases.

15.6.2 Parametric Influences on Reaction Time

Lock Time is designed to be as short as possible while still providing the highest possible stability. The reaction time is not constant, however. Many factors directly and indirectly affect the Lock Time.

The most critical parameter affecting the reaction time of the PLL is the Reference Frequency (FREF) illustrated in **Figure 15-6.** This frequency is the input to the phase detector and controls how often the PLL makes corrections. For stability, it is desirable for the corrections to be small and frequent. Therefore, a higher reference frequency provides optimal performance; 8MHz is preferred. This parameter is under user control through the choice of OSC_CLK frequency FOSC and the value programmed in the reference divider.

15.7 PLL Frequency Lock Detector Block

This digital block monitors the VCO output clock and sets the LCK[1:0] bits in the PLL Status Register (PLLSR) based on its frequency accuracy. The lock detector is enabled with the LCKON bit of the PLL Control Register (PLLCR), as well as the PLL_PDN bit of PLLCR. Once enabled, the detector starts two counters whose outputs are periodically compared. The input clocks to these counters are the VCO output clock divided by the value in the $\overline{N}$ register, called feedback, and the PLL input clock, illustrated as FREF in **Figure 15-6.** The period of the pulses being compared cover one whole period of each clock. This is due to the feedback clock not guaranteeing a 50 percentage duty cycle. The design of this block was accomplished with the assumption the feedback clock transitions high on the rising edge of FREF.

Feedback and FREF clocks are compared after 16, 32, and 64 cycles. If, after 32 cycles, the clocks match, the LCK0 bit is set to one. If, after 64 cycles of FREF, there is the same number of FREF clocks as feedback clocks, the LCK1 bit is also set. The LCK bit stay set until:

- Clocks Fail to Match
- When a New Value is Written to the N-factor
- On Reset Caused By LCKON, PLL_PDN

- Chip_Level Reset

When the circuit sets the LCK1, the two counters are reset and start the count again. The lock detector is designed so if LCK1 is reset to 0 because clocks did not match, LCK0 can stay high. This provides the processor the accuracy of the two clocks with respect to each other.

Table 15-5. Document Revision History for Chapter 15

Version History	Description of Change
Rev. 8	Formatting, layout, spelling, and grammar corrections. Added revision history table. In **Figure 15-8**, changed the name of bit 7 (was LOKON, is LCKON).

Chapter 16
Reset, Low Voltage, Stop and Wait Operations

16.1 Introduction

This chapter is devoted to the description and sources of three different types of reset methods and their effects on the system used with this controller product. Those three reset methods are:

1. External
2. COP timeout
3. Low voltage

Additionally, the chapter describes control of the Stop and Wait modes.

16.2 Sources of Reset

Three sources of reset present in this system are:

1. External RESET
2. Computer Operating Properly (COP) reset
3. Power-On Reset (POR)

Figure 16-1 illustrates how these Reset sources are used within the chip.

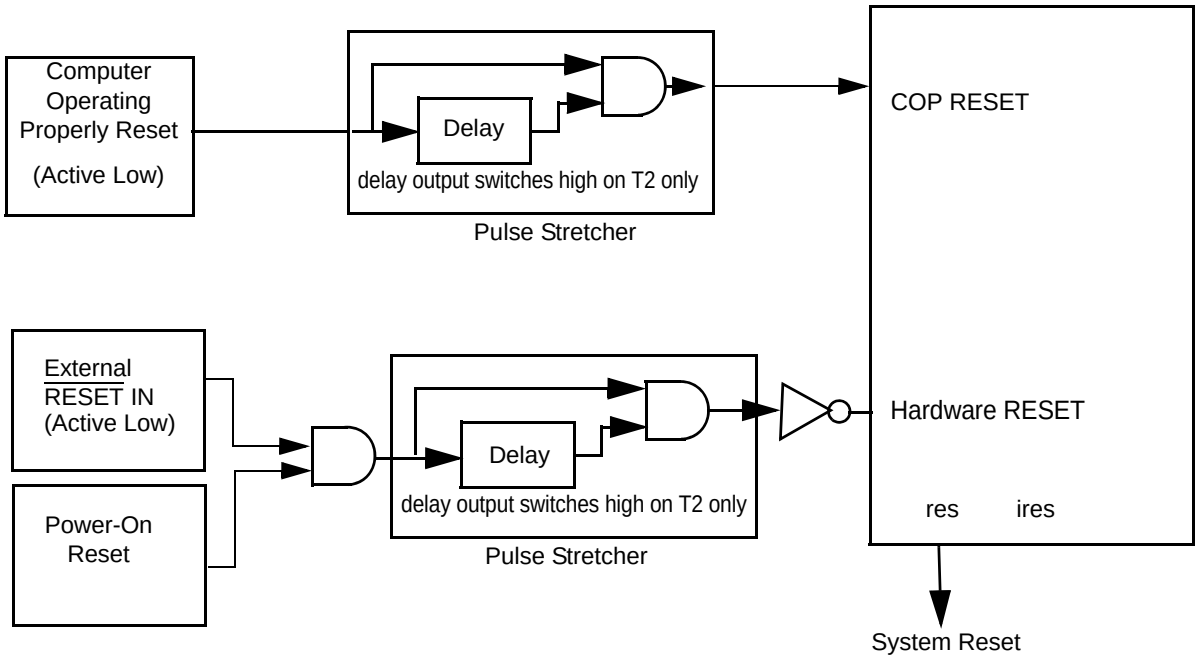


Figure 16-1. Sources of RESET

By default, the Pulse Stretcher function forces Internal Reset Signals to be asserted a *minimum* of 63 oscillator clock cycles after the release of the external reset input.

If the external reset input is held high for three external clock cycles after the Power-On Reset (POR) circuit has detected V_{DD} greater than 1.8V, the pulse stretcher will force the internal reset signals for a minimum of 2,097, 151 external clock cycles. This mode of operation is called Internal Reset Generation (IRG). If the external clock crystal is 8MHz, the internal reset period is approximately 250ms. IRG mode ensures the clock oscillator has plenty of time to stabilize prior to system operation.

If an external reset circuit is connected to the external reset input, the external reset input can be held low during the POR process resulting in the IRG mode not being enabled.

The RSTO output signal mirrors the internal reset operation.

In addition to the reset sources above, two low voltage detect signals initiate a controlled shut down of the chip when voltage supply drops below acceptable levels. These low voltage detect circuits are assigned as high priority interrupts. They can be masked if desired. Please see **Section 16.9.1, “System Control Register (SYS_CNTL)”.**

The following list develops details of each system.

16.3 Register Summary

The Reset, Low Voltage, and the Stop and Wait operations registers of the 56F80x COP Control (COPCTL) register

- COP Time Out (COPTO) register
- COP Service (COPSRV) register
- System Control (SYS_CNTL) register
- System Status (SYS_STS) register
- Most Significant Half of JTAG ID (MSH_ID)
- Least Significant Half of JTAG_ID (LSH_ID)

Please refer to **Table 3-29** for additional information about COP registers. Further data concerning the System Control register may be found in **Section 16.9.1, "System Control Register (SYS_CNTL)".**

16.4 Power-On Reset and Low Voltage Interrupt

The basic POR and Low Voltage Detect module is shown in **Figure 16-2.** The POR circuit is designed to assert the internal reset from V_{DD} = 0V to V_{DD} = 1.8V. POR switching high indicates there is enough voltage for on-chip logic to operate at the oscillator frequency. High speed operation is not guaranteed until both the 2.7V and 2.2V interrupt sources are inactive. Please see the SYS_STS register, **Figure 16-10.**

1. If external $\overline{\text{RESET}}$ IN is asserted, Delay = 63 clock cycles else Delay = 2^{21} clock cycles.

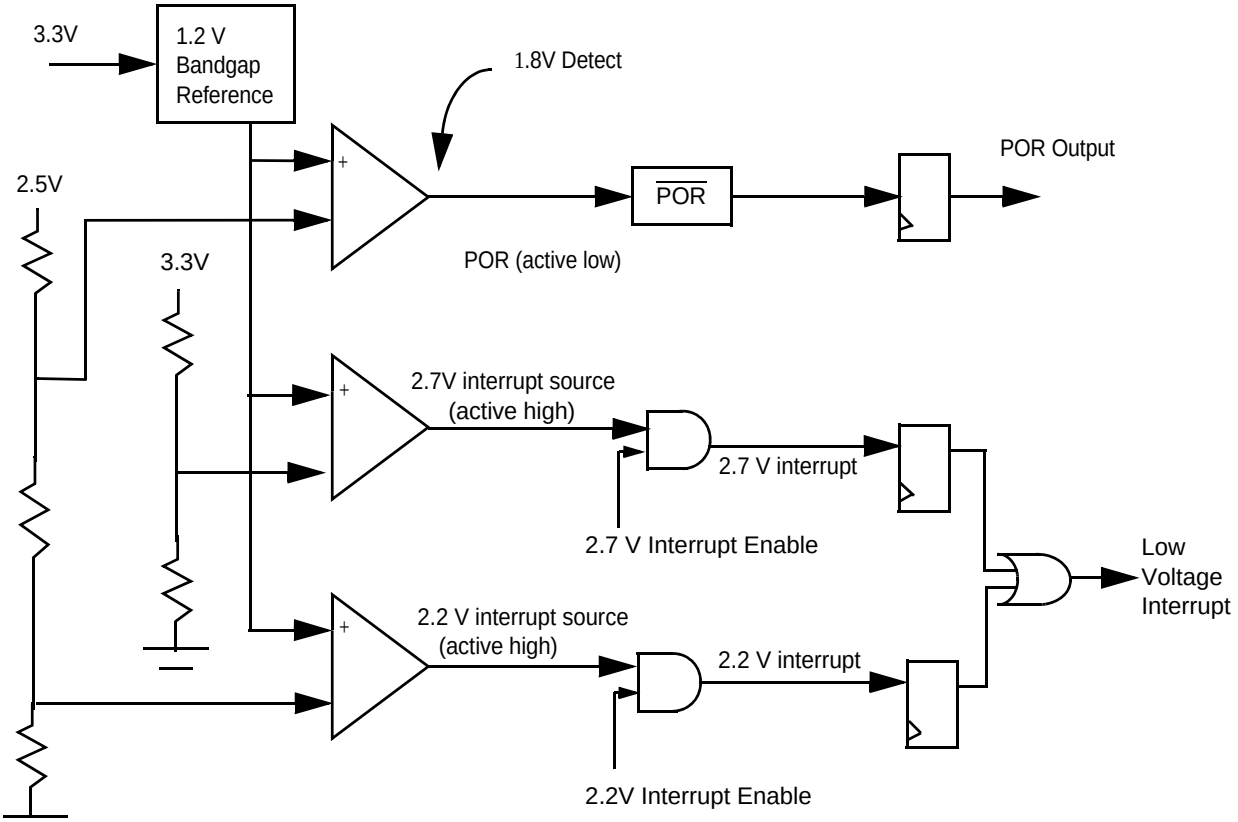


Figure 16-2. $\overline{\text{POR}}$ and Low Voltage Detection

The Low Voltage Interrupts should be explicitly clear and disabled by the interrupt service routine responsible for shutting down the part when a low voltage is detected.

Figure 16-3 illustrates one of the 2.2V and 2.7V interrupts. The 2.2V Low Voltage Interrupts is generated based on the internal logic supply voltage and the 2.7V Low Voltage Interrupt is generated based on the externally supplied V_{DD}.

The POR enable signal is used for test purposes only. Only the POR enable is low by default. Low Voltage Interrupts are masked upon POR. They must be explicitly enabled and disabled thereafter. Low Voltage Interrupts include roughly 50mV of hysteresis.

16.5 External Reset

The External Reset is active low, causing the part to be immediately placed in a reset condition. Some internal portions of the chip may be synchronously reset. The internal waveform shaper ensures the internal reset remains low long enough for all portions of the chip to achieve their reset value.

When the $\overline{\text{RESET}}$ pin is deasserted, the Initial Chip Operating mode is latched into the OMR, based on the value present on the EXTBOOT pin. Additional details are found in **Section 3.3.2, "Operating Mode Register (OMR)".**

16.6 Computer Operating Properly (COP) Module

The Computer Operating Properly (COP) module is used to help software recover from runaway code. The COP register is a free-running counter. Once enabled, it is designed to generate a reset on overflow. Software must periodically service the COP module in order to clear the counter and prevent a reset.

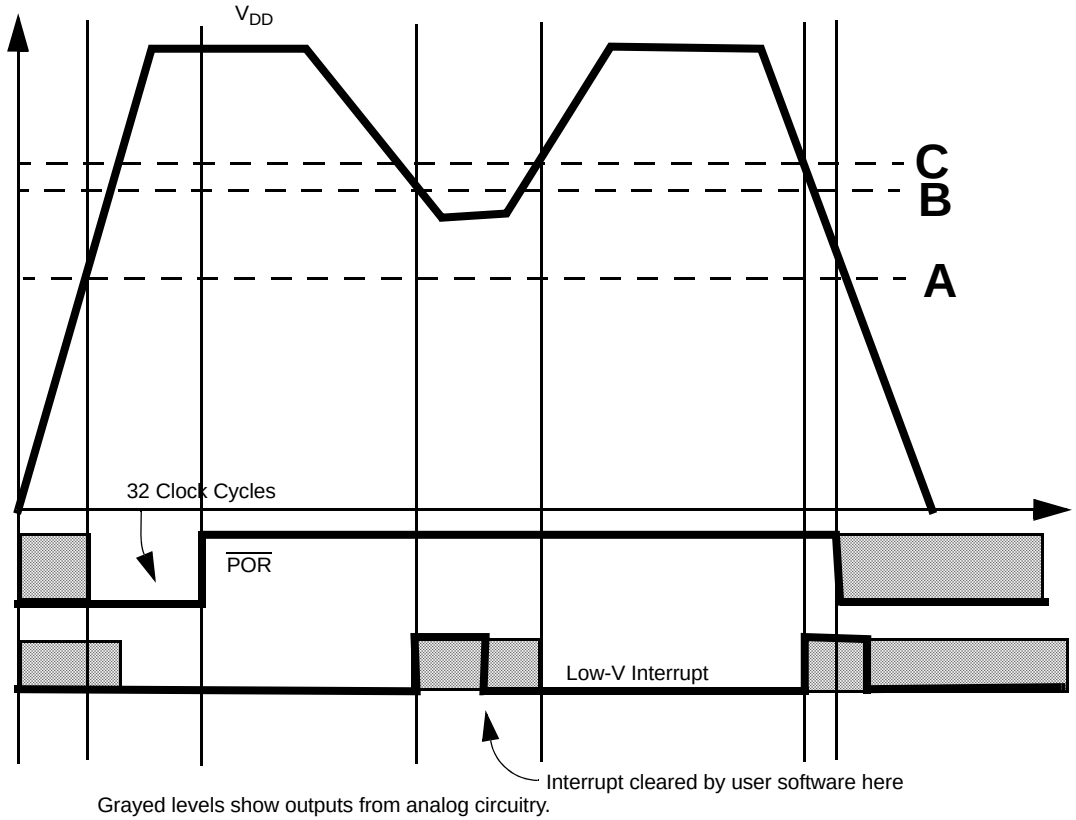


Figure 16-3. $\overline{\text{POR}}$ Versus Low Voltage Interrupts

16.7 COP Functional Description

16.7.1 Timeout Specifications

The COP uses a 12-bit counter with a prescaler to provide 4096 different timeout periods. The prescaler is a divide-by-16384 version of the CPU clock. At 80 MHz the CPU clock will give the COP a minimum timeout period of 250μs and a maximum of 839ms and a resolution of 205μs. The timeout period can be selected by writing to the COP Timeout (COPTO) bits (CT[11:0]) in the COPTO Register.

16.7.2 COP After Reset

When the COPCTL is cleared and out of reset, COP is disabled. Further, the COPTO is set to $FFF and the COPSRV is set to $000 during reset. The COP register can be enabled by setting the CEN bit in the COPCTL register.

The COP can be disabled by clearing the CEN bit in the COPCTL register. This action resets the COP counter to the value in the Timeout register. Resetting the COP module also loads the COP counter from the Timeout register whose value is $FFF. Both COPCTL and COPTO may be modified as long as the CWP bit in the COPCTL register is clear. Once the CWP bit is set, COPCTL and COPTO is write-protected until a Reset occurs.

16.7.3 COP in the Wait Mode

The COP module can run or be terminated in the Wait mode. If the COP Wait Enable (CWEN) bit in the COPCTL register is set, the COP counter runs in the Wait mode. However, when the CWEN bit is cleared, the COP counter is disabled in the Wait mode.

16.7.4 COP in the Stop Mode

The COP module can run or be disabled in Stop mode. When the COP Stop Enable (CSEN) bit in COPCTL is set, the COP counter runs in Stop mode. However, when the CSEN bit is cleared, the COP counter is disabled in Stop mode.

16.8 Register Definitions

Table 16-1. COP/SIM Memory Map

Device	Peripheral	Address
801/802/ 803/805	COP_BASE	$0F30
	SYS_BASE	$0C00
807	COP_BASE	$1330
	SYS_BASE	$1000

The address of a register is the sum of a base address and an address offset. The base address is defined at the core unit level and the address offset is defined at the module level.

Table 16-2. COP/SIM Register Summary

Address Offset	Register Acronym	Register Name	Access Type	Chapter Location
COP_Base + $0	COPCTL	Control Register	Read/Write	Section 16.8.1
COP_Base + $1	COPTO	Timeout Register	Read/Write	Section 16.8.2
COP_Base + $2	COPSRV	Service Register	Read/Write	Section 16.8.3
SYS_Base + $0	SYS_CNTL	System Control Register	Read/Write	Section 16.9.1
SYS_Base + $1	SYS_STS	Ssytem Status Register	Read/Write	Section 16.9.2
SYS_Base + $6	MSH_ID	Most Significant Half ID Register	*Read-Only*	Section 16.9.3
SYS_Base + $7	LSH_ID	Least Significant Half ID Register	*Read-Only*	Section 16.9.4

Bit fields of the above seven registers are summarized in **Figure 16-4**. Details of each follow.

Add. Offset	Register Name		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
COP_Base + $0	COPCTL	R	0	0	0	0	0	0	0	0	0	0	0	0	CSEN	CWEN	CEN	CWP
		W																
COP_Base + $1	COPTO	R	0	0	0	0	CT											
		W																
COP_Base + $2	COPSRV	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W	COP SERVICE REGISTER															
SYS_Base + $0	SYS_CNTL	R	0	0	0	0	TMRPD	CTRLPD	ADRPD	DATAPD	0	0	0	BOOTMAP_B	LVIE27	LVIE22	PD	RPD
		W																
SYS_Base + $1	SYS_STS	R	0	0	0	0	0	0	0	0	0	0	0	COPR	EXTR	POR	LVIS27	LVIS22
		W																
SYS_Base + $6	MSH_ID	R	0	0	0	1	0	0	1	1	1	1	1	1	0	0	1	0
		W																
SYS_Base + $7	LSH_ID	R	0	0	0	1	0	0	1	1	1	1	1	1	0	0	1	0
		W																

Both MSH and LSH register values are hard coded and cannot be reset.

R	0	Read as 0
W		Reserved

Figure 16-4. COP/SIM Registers Map Summary

16.8.1 COP Control Register (COPCTL)

COP_BASE+$0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	0	0	0	0	0	0	0	CSEN	CWEN	CEN	CWP
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 16-5. COP Control Register (COPCTL)

See Table A-8, List of Programmer's Sheets

16.8.1.1 Reserved—Bits 15–4

This bit field is reserved or not implemented. It is read as 0 and cannot be modified by writing.

16.8.1.2 Stop Enable (CSEN)—Bit 3

This bit controls the state of the COP counter in the Stop mode. This bit can only be modified when CWP is set to zero. For the COP to run in Stop mode, CEN must also be set.

- 0 = COP counter will stop in Stop mode
- 1 = COP counter will run in Stop mode

16.8.1.3 COP Wait Enable (CWEN)—Bit 2

This bit controls the state of the COP counter in the Wait mode. This bit can only be modified when CWP is set to zero. For the COP to run in the Wait mode, CEN must also be set.

- 0 = COP counter will stop in Wait mode
- 1 = COP counter will run in Wait mode

16.8.1.4 COP Enable (CEN)—Bit 1

This bit determines if the COP counter is enabled or disabled. This bit can only be modified when CWP is set to zero.

- 0 = COP is disabled
- 1 = COP is enabled

16.8.1.5 COP Write Protect (CWP)—Bit 0

This bit is used to control the write protection feature of the COP Control (COPCTL) register and COP Timeout (COPTO) register. Once set, this bit can only be cleared by system reset.

- 0 = COPCTL, COPTO may be modified

- 1 = COPCTL, COPTO are *read-only*

16.8.2 COP Timeout Register (COPTO)

COP_BASE+$2	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	CT											
Write																
Reset	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1

Figure 16-6. COP Timeout Register (COPTO)

See Table A-8, List of Programmer's Sheets

16.8.2.1 Reserved—Bits 15–12

This bit field is reserved or not implemented. It is read as 0 and cannot be modified by writing.

16.8.2.2 COP Timeout (CT)—Bits 11–0

COP timeout bits are used to control the length of the timeout period. COP timeout period equals [16384 × (CT[11:0]) + 1]) clock cycles.

The value in this register determines the timeout period, as shown in the above formula. The formula, CT[11:0], should be written before the COP is enabled. Once the COP has been enabled, the recommended procedure for changing CT[11:0] is to disable the COP, write to COPTO, and reactivate the COP. This ensures the new timeout value is loaded into the counter. Alternatively, the CPU can write to COPTO and then write the proper patterns to COPSRV, causing the counter to reload with the new CT[11:0] value. The COP counter is not reset by a write to COPTO.

Note: Changing CT[11:0] while the COP is enabled will result in a timeout period differing from the expected value given by the above formula. These bits can only be changed when CWP bit is set to zero.

16.8.3 COP Service Register (COPSRV)

COP_BASE+$4	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Write	COP SERVICE REGISTER															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 16-7. COP Service Register (COPSRV)

See Table A-8, List of Programmer's Sheets

When enabled, the COP requires a service sequence to be performed periodically, clearing the COP counter to prevent a reset. This routine consists of writing a $5555 to the COPSRV followed by writing a $AAAA to the COPSRV before the expiration of the selected timeout period. The writes to COPSRV must be performed in the correct order before the counter times out, but any number of instructions may be executed between the two writes.

16.9 Stop and Wait Mode Disable Function

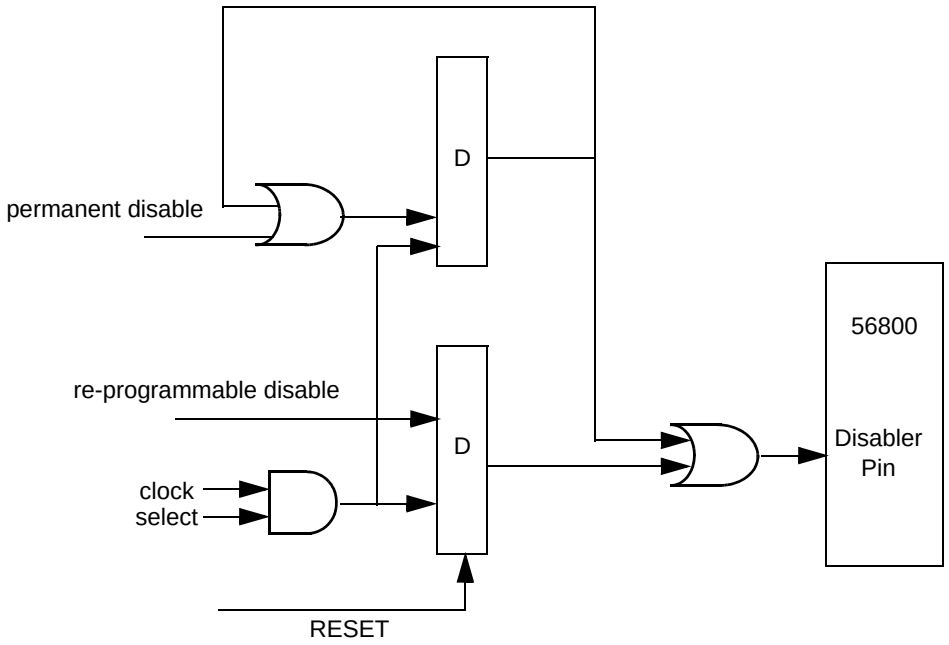


Figure 16-8. Stop/Wait Disable Circuit

The 56F800 core contains both Stop and Wait instructions. Both instructions put the CPU to sleep. The PLL and peripheral bus continue to run in Wait mode, but not in Stop mode. The ADC

will be placed in low power mode in both. In Stop mode, the core clock system (ZCLK) is set equal to the oscillator output.

Some applications require the core Stop/Wait instructions to be disabled. This is accomplished on either a permanent or temporary basis by writing to the System Control (SYS_CNTL) register, as described in the next section. Permanently assigned applications are last only until their next reset.

16.9.1 System Control Register (SYS_CNTL)

SYS_BASE +$0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	TMRPD	CTRLPD	ADRPD	DATAPD	0	0	0	BOOT MAP_B	LVIE 27	LVIE 22	PD	RPD
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 16-9. System Control Register (SYS_CNTL)

See Table A-8, List of Programmer's Sheets

16.9.1.1 Reserved—Bits 15–12

This bit field is reserved or not implemented. It is read as 0 and cannot be modified by writing.

16.9.1.2 Timer I/O Pull-Up Disable (TMRPD)—Bit 11

If this bit is set, the pull-ups for the Timer I/O pins are disabled. Normally the pull-ups are enabled.

16.9.1.3 Control Signal Pull-Up Disable (CTRL PD)—Bit 10

If this bit is set, the pull-ups for the $\overline{DS}$, $\overline{PS}$, $\overline{RD}$, and the $\overline{WR}$ I/O pins are disabled. Normally, the pull-ups are enabled.

16.9.1.4 Address Bus Pull-Up Disable (ADRPD)—Bit 9

If this bit is set, the pull-ups for the lower address I/O pins are disabled. Normally, the pull-ups are enabled.

16.9.1.5 Data Bus I/O Pull-Up Disable (DATA PD)—Bit 8

If this bit is set, the pull-ups for the Data Bus I/O pins are disabled. Normally, the pull-ups are enabled.

16.9.1.6 Reserved—Bits 7–5

This bit field is reserved or not implemented. It is read as 0 and cannot be modified by writing.

16.9.1.7 Bootmap ($\overline{\text{BOOTMAP}}$)—Bit 4

This bit determines the memory map when the Operating Mode Register (OMR) MA and MB bits are set to enable internal program memory.

Table 16-3. Memory Map Controls

Mode Number	MA	MB	BOOTMAP1	Description
0A	0	0	0	Boot Mode
0B	0	0	1	Half Internal & Half External PMEM
0	0	1	X	Reserved
0	1	0	X	Reserved
3	1	1	X	Development

16.9.1.8 2.7V Low Voltage Interrupt Enable (LVIE27)—Bit 3

When 1, this bit enables the 2.7V low voltage interrupt. When 0, this interrupt is disabled.

16.9.1.9 2.2V Low Voltage Interrupt Enable (LVIE22)—Bit 2

When 1, this bit enables the 2.2V low voltage interrupt. When 0, this interrupt is disabled.

16.9.1.10 Permanent Stop/Wait Disable (PD)—Bit 1

This bit can only be cleared by system reset. Once set, Stop and Wait instructions essentially act as loops.

16.9.1.11 Re-Programmable Stop/Wait Disable (RPD)—Bit 0

This bit can be set *and* cleared under software control. While set, Stop and Wait instructions act as loops.

16.9.2 System Status Register (SYS_STS)

This register is reset upon any system reset. It is initialized only by a Power-On Reset (POR).

SYS_BASE +$1	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	0	0	0	0	0	0	COPR	EXTR	POR	LVIS 27	LVIS 22
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	—	—	—	0	0

Figure 16-10. System Status Register (SYS_STS)

See Table A-8, List of Programmer's Sheets

16.9.2.1 Reserved—Bits 15–5

This bit field is reserved or not implemented. It is read as 0 and cannot be modified by writing.

16.9.2.2 COP Reset (COPR)—Bit 4

When 1, the COPR bit indicates the Computer COP Timer generated reset occurred. This bit is cleared by a Power-On Reset or by software. Writing 0 to this bit position will set the bit, while writing 1 to the bit will clear it.

16.9.2.3 External Reset (EXTR)—Bit 3

When 1, the EXTR bit indicates an external system reset occurred. This bit is cleared by a Power-On Reset or by software. Writing 0 to this bit position sets the bit, while writing 1 to the bit position clears it. Setting this bit *will not* reset system.

16.9.2.4 Power-On Reset (POR)—Bit 2

When 1, the POR bit indicates a Power-On Reset occurred some time previously. This bit can only be cleared by software. Writing 0 to this bit sets the bit, while writing 1 to the bit position clears the bit. Setting this bit *will not* reset system.

16.9.2.5 2.7V Low Voltage Interrupt Source (LVIS27)—Bit 1

When 1, the LVIS27 bit indicates a 2.7V low voltage interrupt is active. Writing 0 to this bit position sets this bit. Writing 1 to this bit position clears this bit. Setting this bit causes an interrupt if the corresponding interrupt enable bit is set. When 1, this bit indicates teh chip voltage dropped below 2.7V.

16.9.2.6 2.2 Low Voltage Interrupt Source (LVIS 22)—Bit 0

When 1, the LVIS22 bit indicates a 2.2V low voltage interrupt is active. Writing 0 to this bit position sets this bit. Writing 1 to this bit position clears this bit. Setting this bit will cause an interrupt when the corresponding interrupt enable bit is set. When 1, this bit indicates the chip voltage dropped below 2.2V.

16.9.3 Most Significant Half of JTAG ID (MSH_ID)

This read-only register displays the Most Significant Half of the JTAG ID for the chip. For the 56F80x chip, read as $01F2.

SYS_BASE +$6	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	0	0	0	0	0	0	1	1	1	1	1	0	0	1	0
Write																
Reset	0	0	0	0	0	0	0	1	1	1	1	1	0	0	1	0

Figure 16-11. Most Significant Half of JTAG_ID (MSH_ID)

See Table A-8, List of Programmer's Sheets

16.9.4 Least Significant Half of JTAG ID (LSH_ID)

This *read-only* register displays the Least Significant Half of the JTAG ID for the chip. For the 56F80x chip, read as $701D.

SYS_BASE +$7	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	0	1	0	1	0	0	0	0	0	0	0	1	1	1	0	1
Write																
Reset	0	1	0	1	0	0	0	0	0	0	0	1	1	1	0	1

Figure 16-12. Least Significant Half of JTAG_ID (LSH_ID)

See Table A-8, List of Programmer's Sheets

Table 16-4. Document Revision History for Chapter 16

Version History	Description of Change
Rev. 8	Formatting, layout, spelling, and grammar corrections. Added revision history table. Updated COPSRV figure to show the COP service register field properly. Updated Section 16.8 with proper address offsets (including SYS_Base and COP_Base references.)

Chapter 17
OnCE Module

17.1 Introduction

This chapter describes the 56F800 core-based family devices providing board and chip-level testing capability through two on-chip modules. Both modules are accessed through the JTAG/OnCE™ port. The modules are:

- On-Chip Emulation (OnCE) module
- Test Access Port (TAP) and 16-state controller, known also as the Joint Test Action Group (JTAG) port

The presence of the JTAG/OnCE port permits users to insert the digital signal controller into a target system while retaining debug control. This capability is especially important for devices without an external bus thereby eliminating the need for an expensive cable bringing out the chip footprint required by a traditional emulator system. Additionally, on the 56F80x devices, the JTAG/OnCE port can be used to program the internal Flash Memory OnCE module.

17.2 Features

The OnCE module is a module used in DSC devices to debug application software used with the chip. The module is a separate on-chip block allowing non-intrusive interaction with the controller, with the accessibility through the pins of the JTAG interface. The OnCE module makes it possible to examine registers, memory, or on-chip peripherals' contents in a special debug environment. This avoids sacrificing any user-accessible on-chip resources to perform debugging.

The capabilities of the OnCE module include the ability to:

- Interrupt or break into Debug mode on a program memory address: fetch, read, write, or access
- Interrupt or break into Debug mode on a data memory address: read, write, or access
- Interrupt or break into Debug mode on an on-chip peripheral register access: read, write, or access
- Enter Debug mode using a controller microprocessor instruction

- Display or modify the contents of any controller core register
- Display or modify the contents of peripheral memory-mapped registers
- Display or modify any desired sections of program or data memory
- Trace one, single stepping, or as many as 256 instructions
- Save or restore the current state of the controller's pipeline
- Display the contents of the real-time instruction trace buffer, whether in Debug mode or not
- Return to user mode from Debug mode
- Set up breakpoints without being in Debug mode
- Set hardware breakpoints, software breakpoints, and trace occurrences (OnCE events) possibly forcing the chip into Debug mode, force a vectored interrupt, force the real-time instruction buffer to halt, or toggle a pin, based on the user's needs

Please refer to **Section 17.6, "OnCE Module Architecture"** for a detailed description of this port.

Joint Test Action Group (JTAG) Port

The JTAG port is a dedicated user-accessible Test Access Port (TAP) compatible with the *IEEE 1149.1a-1993 Standard Test Access Port and Boundary Scan Architecture*. Problems associated with testing high-density circuit boards have led to the development of this proposed standard under the sponsorship of the Test Technology Committee of IEEE and the JTAG. The DSP56800 family supports circuit board test strategies based on this standard.

Five dedicated pins interface to a TAP containing a 16-state controller. The TAP uses a boundary scan technique to test the interconnections between integrated circuits after they are assembled onto a printed circuit board. Boundary scan allows a tester to observe and control signal levels at each component pin through a shift register placed next to each pin. This is important for testing continuity and determining if pins are stuck at a one or zero level.

The TAP port has the following capabilities:

- Perform boundary scan operations to test circuit board electrical continuity
- Bypass the controller for a given circuit board test by replacing the boundary scan register (BSR) with a single-bit register
- Provide a means of accessing the OnCE module controller and circuits to control a target system
- Disable the output drive to pins during circuit board testing

- Sample the controller system pins during operation, and shift out the result in the BSR; preload values outputing pins prior to invoking the EXTEST instruction
- Query identification information, manufacturer, part number, and version from a controller
- Force test data onto the outputs of a controller IC, while replacing its BSR in the serial data path with a single-bit register
- Enable a weak pull-up current device on all input signals of a controller IC, ensuring determined test results in the presence of a continuity fault during interconnect testing

17.3 Combined JTAG/OnCE Interface Overview

The JTAG and OnCE blocks are tightly coupled. **Figure 17-1** shows the block diagram of the JTAG/OnCE port with its two distinctive parts. The JTAG port is the master. It must enable the OnCE module before the OnCE module can be accessed.

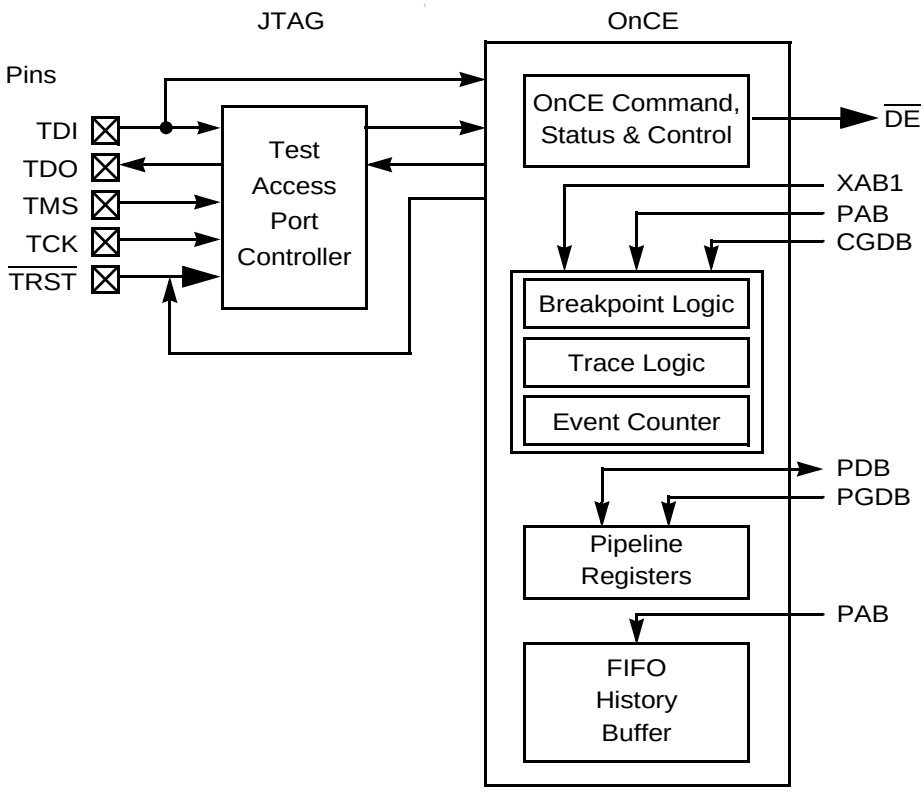


Figure 17-1. JTAG/OnCE Port Block Diagram

There are three different programming models to consider when using the JTAG/OnCE interface:

- OnCE programming model—accessible through the JTAG port
- OnCE programming model—accessible from the controller core

- JTAG programming model—accessible through the JTAG port

The programming models are discussed in more detail in **Section 17.6** and **Section 18.6.**

17.4 JTAG/OnCE Port Pin Descriptions

As described in the IEEE 1149.1a-1993 specification, the JTAG port requires a minimum of four pins to support TDI, TDO, TCK, and TMS signals. The 56F80x also uses the optional Test Reset ($\overline{\text{TRST}}$) input signal and a $\overline{\text{DE}}$ pin used for debug event monitoring. The pin functions are described in **Table 17-1.**

Table 17-1. JTAG/OnCE Pin Descriptions

Pin Name	Pin Description
TDI	**Test Data Input**—This input provides a serial data stream to the JTAG and the OnCE module. It is sampled on the rising edge of TCK and has an on-chip pull-up resistor.
TDO	**Test Data Output**—This tri-stateable output provides a serial data stream from the JTAG and the OnCE module. It is driven in the Shift-IR and Shift-DR controller states of the JTAG state machine and changes on the falling edge of TCK.
TCK	**Test Clock Input**—This input proves a gated clock to synchronize the test logic and shift serial data through the JTAG/OnCE port. The maximum frequency for TCK is 1/8 the maximum frequency of the 56F80x devices (i.e., 5MHz for TCK if the maximum CLK input is 40MHz). The TCK pin has an on-chip pull-down resistor.
TMS	**Test Mode Select Input**—This input sequences the TAP controller's state machine. It is sampled on the rising edge of TCK and has an on-chip pull-up resistor.
$\overline{\text{TRST}}$	**Test Reset**—This input provides a reset signal to the TAP controller. This pin has an on-chip pull-up resistor.
$\overline{\text{DE}}$	**Debug Event**—This output signals OnCE events detected on a trigger condition. (Not available on the 802 device.)

The $\overline{\text{DE}}$ pin provides a useful event acknowledge feature. This selection is performed through appropriate control bits in the OCR.

- $\overline{\text{DE}}$ When enabled, this output provides a signal indicating an event has occurred in the OnCE debug logic. This event can be any of the following occurrences:
  - — Hardware breakpoint
  - — Software breakpoint
  - — Trace or entry into Debug mode caused by a DEBUG_REQUEST instruction being decoded in the JTAG port Instruction Register (IR)

The $\overline{\text{DE}}$ output indicates an OnCE event has occurred. For example, a trace or breakpoint occurrence causes the pin to go low for a minimum of eight Phi clocks. This pin is further described in **Section 17.7.4.6, "Event Modifier (EM[1:0])—Bits 6–5".**

When the JTAGIR does not contain an ENABLE_ONCE instruction, the OCR control bits are reset only by assertion of $\overline{\text{RESET}}$ or COP timer reset. When the ENABLE_ONCE instruction is in the JTAGIR at the time of reset, the OCR bits are not modified.

17.5 Register Summary

The OnCE module of the 56F80x have the following registers:

- OnCE Breakpoint Address Register (OBAR)
- OnCE Breakpoint Address Register 2 (OBAR2)
- OnCE Breakpoint Control Register 2 (OBCTL2)
- OnCE Command Register (OCMDR)
- OnCE Breakpoint Mask Register 2 (OBMSK2)
- OnCE Count Register (OCNTR)
- OnCE Control Register (OCR)
- OnCE Decoder (ODEC) (*not memory mapped)*
- OnCE Memory Address Comparator (OMAC) (*not memory mapped)*
- OnCE Memory Address Comparator 2 (OMAC2) (*not memory mapped)*
- OnCE Memory Address Latch (OMAL) (*not memory mapped)*
- OnCE Memory Address Latch 2 (OMAL2) (*not memory mapped)*
- OnCE PAB (Program Address Bus) Decode Register (OPABDR)
- OnCE PAB (Program Address Bus) Execute Register (OPABER)
- OnCE PAB (Program Address Bus) Fetch Register (OPABFR)
- OnCE PDB (Program Data Bus) Register (OPDBR)
- OnCE PAB Change-of-Flow (OPFIFO) (*not memory mapped)*
- OnCE PDGB (Program Global Data Bus) Register (OPGDBR)
- OnCE Shift Register (OSHR) (*not memory mapped)*
- OnCE Status Register (OSR)

17.6 OnCE Module Architecture

While the JTAG port described in **Section 18.6, “JTAG Port Architecture”** provides board test capability, the OnCE module provides emulation and debug capability to the user. The OnCE module permits full-speed, non-intrusive emulation on a user’s target system or on an evaluation module (EVM) board.

A typical debug environment consists of a target system where the controller resides in the user-defined hardware. The controller's JTAG port interfaces to a command converter board over a seven-wire link, consisting of the five JTAG serial lines, a ground, and reset wire. The reset wire is optional and is only used to reset the controller and its associated circuitry.

The OnCE module is composed of four different sub-blocks, each performing a different task:

- Command, status, and control
- Breakpoint and trace
- Pipeline registers
- FIFO history buffer

A block diagram of the OnCE module is shown in **Figure 17-2.** The registers serially shift information from TDI to TDO through the OnCE Shift Register (OSHR). **Figure 17-3** displays the OnCE module registers accessible through the JTAG port. **Figure 17-4** exhibits the OnCE module registers accessible through the controller core and its corresponding OnCE interrupt vector.

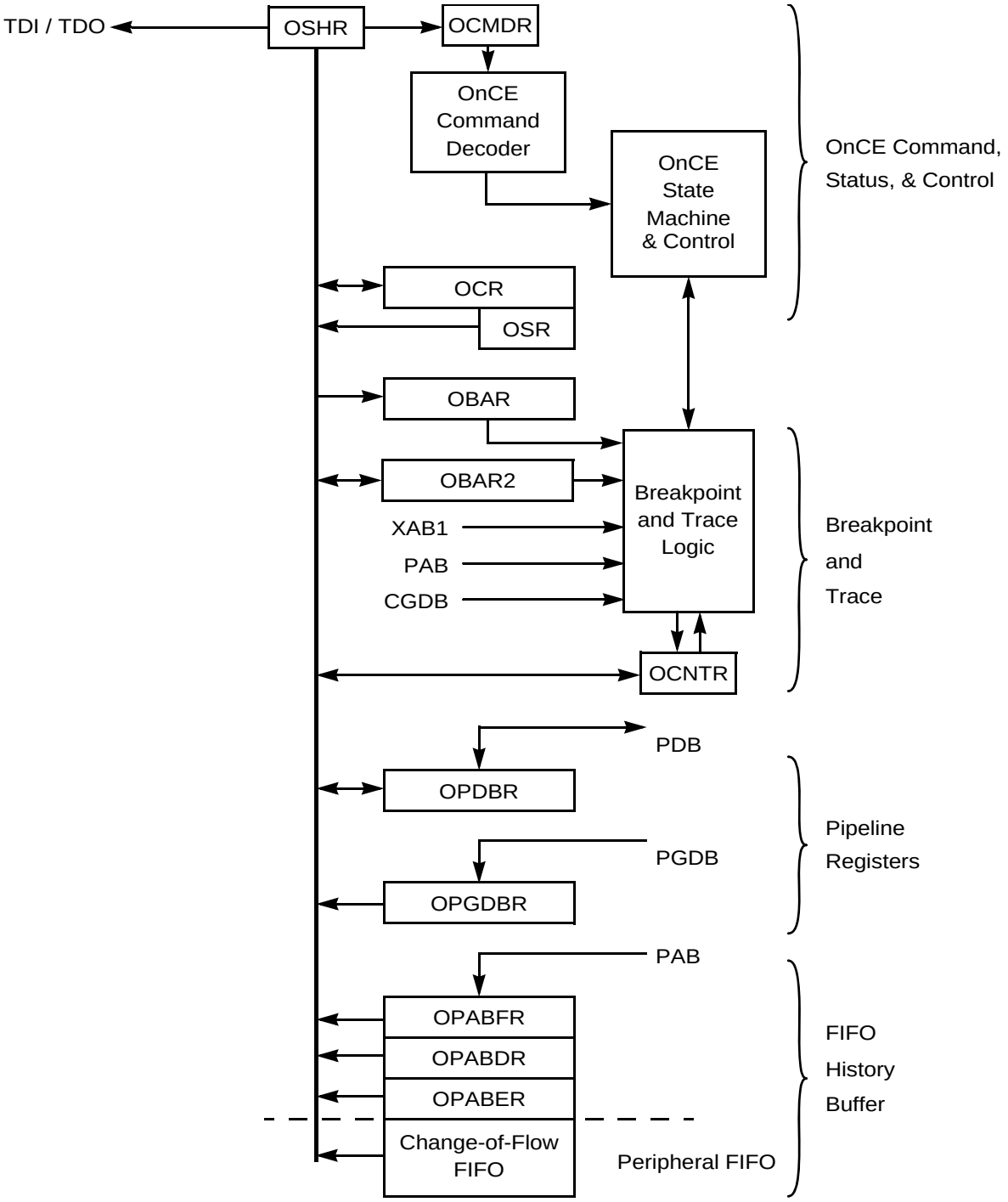
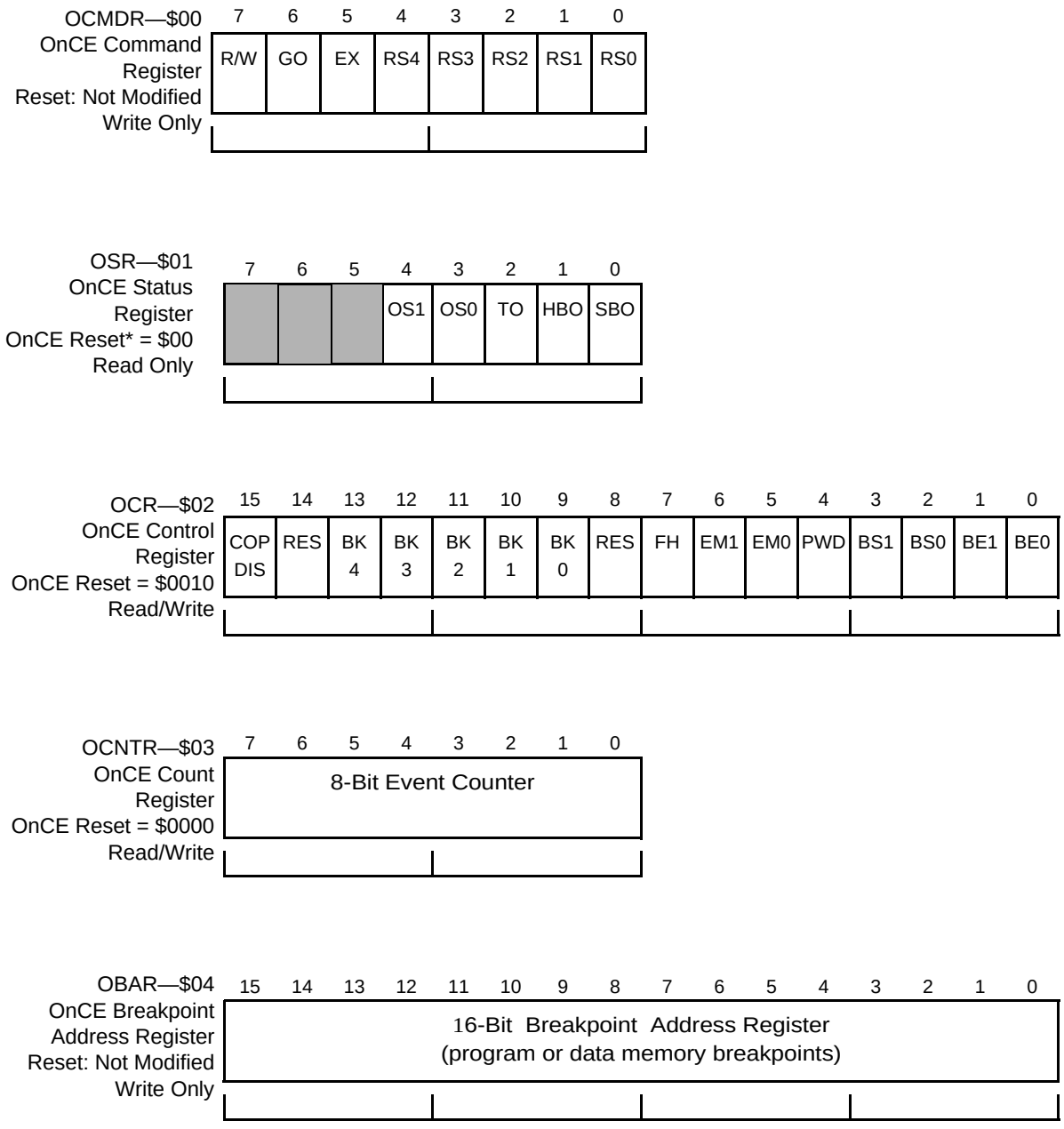


Figure 17-2. 56F80x OnCE Block Diagram



Indicates reserved bits, written as zero for future compatibility.

Note: OnCE reset occurs when hardware or COP reset occurs, and an ENABLE_ONCE instruction is not latched into the JTAG Instruction Register (IR).

Figure 17-3. OnCE Module Registers Accessed Through JTAG

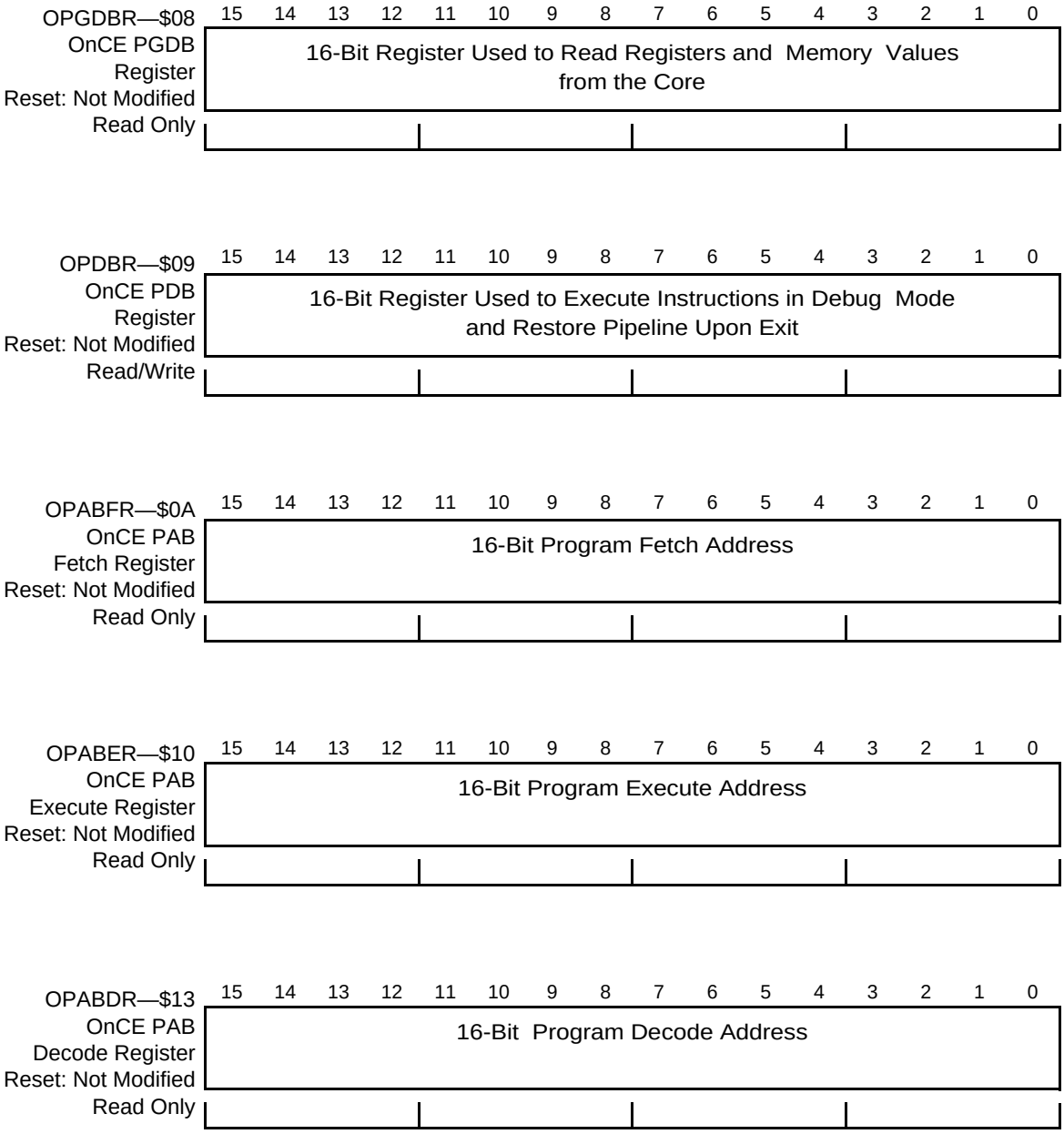
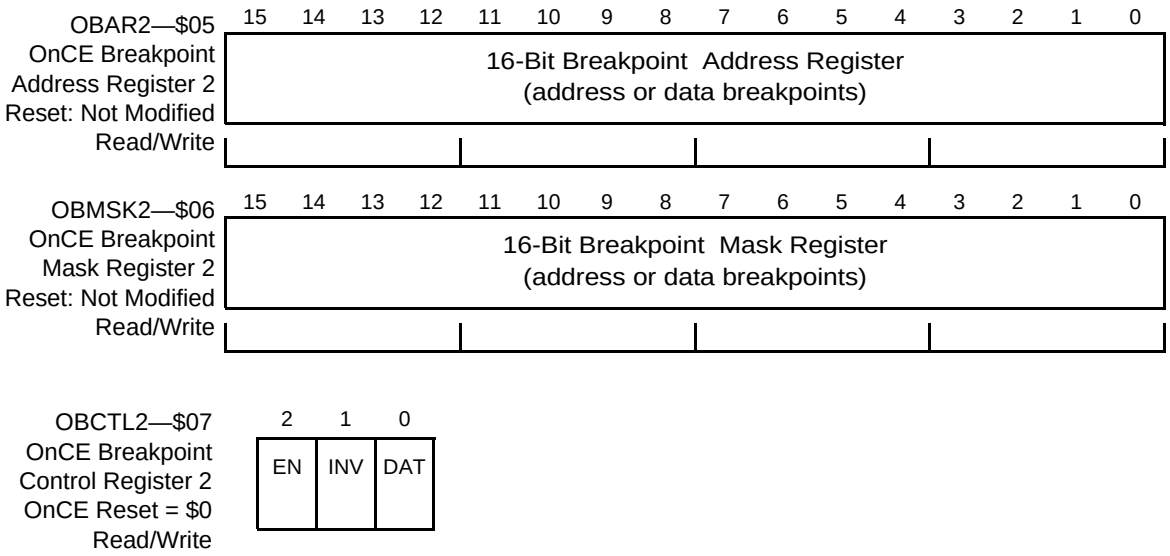
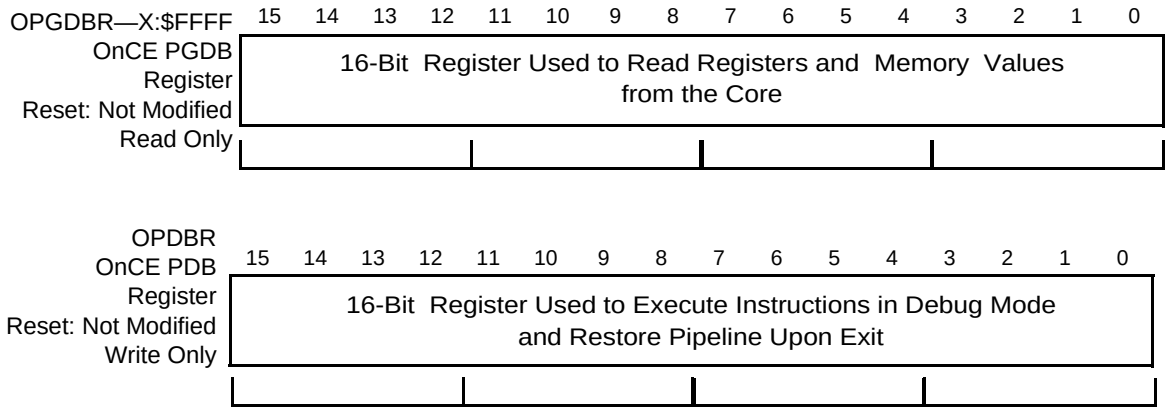


Figure 17-3. OnCE Module Registers Accessed Through JTAG (Continued)



OnCE reset occurs when hardware or Computer Operating Properly (COP) reset occurs, and an ENABL is not latched into the JTAG port Instruction Register (IR).

Figure 17-3. OnCE Module Registers Accessed Through JTAG (Continued)



OnCE Port interrupt vector:

OnCE TRAP P:$000C

OnCE TRAPs are Level one interrupts and not programmable in the IPR

Notes: 1. OPGDBR and OPDBR are not dedicated OnCE module registers. They share functionality with the core. If used incorrectly, they can give unexpected results.

Figure 17-4. OnCE Module Registers Accessed From the Core

The OnCE module has an associated interrupt vector $000C. The user can configure the Event Modifier (EM) bits of the OCR so an OnCE event, other than trace, generates a level one non-maskable interrupt. This interrupt ability is described in **Section 17.7.4.6, "Event Modifier (EM[1:0])—Bits 6–5".**

17.7 Command, Status, and Control Registers

The OnCE command, status, and control registers are described in the following subsections. They include these registers:

- OnCE Breakpoint 2 Control (OBCTL2) register
- OnCE Command Register (OCMDR)
- OnCE Control Register (OCR)
- OnCE Decoder (ODEC) register (*not memory mapped)*
- OnCE Status register (OSR)
- OnCE shift Register (OSHR) (*not memory mapped)*

17.7.1 OnCE Shift Register (OSHR)

The OnCE Shift Register (OSHR) is a JTAG Shift Register sampling the TDI pin on the rising edge of TCK in Shift-DR while providing output to the TDO pin on the falling edge of TCK. The last bit of TDI will be sampled upon exiting Shift-DR on the rising edge of TCK. During OCMDR/OSR transfers, this register is eight bits wide. During all other transfers, this register is 16 bits wide. The input from TDI must be presented to the OSHR least Significant Bit (LSB) first. The TDO pin receives data from the LSB of the OSHR. This register is not memory mapped. It is not possible to modify this register.

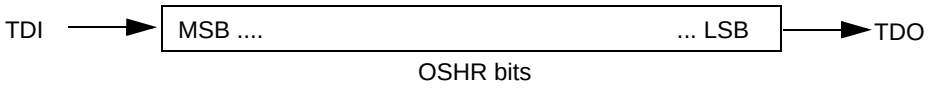


Figure 17-5. OnCE Shift Register (OSHR)

Note: OnCE instructions are shifted on the data side (select-DR-scan) of the TAP controller not the instruction side (select-IR-scan).

17.7.2 OnCE Command Register (OCMDR)

The OnCE module has its own instruction register and instruction decoder, the OnCE Command Register (OCMDR). After a command is latched into the OCMDR, the command decoder implements the instruction through the OnCE state machine and control block. There are two types of commands:

1. Read commands, causing the chip to deliver required data.
2. Write commands, transfering data into the chip, then write it in one of the on-chip resources.

The commands are eight bits long and have the format displayed in **Figure 17-6.** The lowest five bits (RS[4:0]) identify the source for the operation, defined in **Table 17-2.** Bits 5, 6, and 7 delineate the Exit (EX) command bit is located in **Table 17-3,** while the Execute (GO) bit is found in **Table 17-4.** The Read/Write (R/$\overline{W}$) bit is illustrated in **Table 17-5.**

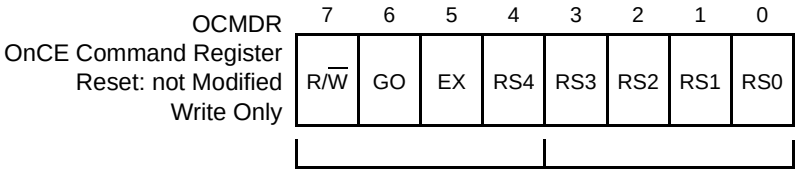


Figure 17-6. OnCE Command Format

Table 17-2. Register Select (RS) Encoding

RS[4:0]	Register/Action Selected	Mode	Read/$\overline{\text{Write}}$
00000	No register selected	All	N/A
00001	OnCE Breakpoint and Trace Counter (OCNTR)	All	Read/Write
00010	OnCE Debug Control Register (OCR)	All	Read/Write
00011	Reserved	All	N/A
00100	OnCE Breakpoint Address Register (OBAR)	All	*Write-only*
00101	(Reserved)	All	N/A
00110	(Reserved)	All	N/A
00111	(Reserved)	All	N/A
01000	OnCE PGDB Bus Transfer Register (OPGDBR)	Debug	*Read-only*
01001	OnCE Program Data Bus Register (OPDBR)	Debug	Read/Write
01010	OnCE Program Address Register—Fetch cycle (OPABFR)	FIFO halted	*Read-only*
01011	(Reserved)	N/A	N/A
01100	Clear OCNTR	All	N/A
01101	(Reserved)	N/A	N/A
01110	(Reserved)	N/A	N/A
01111	(Reserved)	N/A	N/A

Table 17-2. Register Select (RS) Encoding (Continued)

RS[4:0]	Register/Action Selected	Mode	Read/$\overline{\text{Write}}$
10000	OnCE Program Address Register—Execute cycle (OPABER)	FIFO halted	*Read-only*
10001	OnCE Program address FIFO (OPFIFO)	FIFO halted	*Read-only*
10010	(Reserved)	N/A	N/A
10011	OnCE Program Address Register—Decode cycle (OPABDR)	FIFO halted	*Read-only*
101xx	(Reserved)	N/A	N/A
11xxx	(Reserved)	N/A	N/A

Table 17-3. EX Bit Definition

EX	Action
0	Remain in the Debug Processing State
1	Leave the Debug Processing State

Table 17-4. GO Bit Definition

GO	Action
0	Inactive—No Action Taken
1	Execute Controller Instruction

Note: In the OnCE command word, bit five is the exit command. To leave the Debug mode and re-enter the Normal mode, both the EX and GO bits must be asserted in the OnCE Input Command register.

Table 17-5. R/$\overline{\text{W}}$ Bit Definition

R/$\overline{\text{W}}$	Action
0	Write To the Register Specified by the RS[4:0] Bits
1	ReadFrom the Register Specified by the RS[4:0] Bits

17.7.3 OnCE Decoder (ODEC)

The OnCE Decoder (ODEC) decodes all OnCE instructions received in the OCMDR. The ODEC generates all the strobes required for reading and writing the selected OnCE registers. This block prohibits access to the OnCE Program Data Bus Register (OPDBR) and the OnCE PGDB Bus Transfer Register (OPGDBR) if the chip is not in the Debug mode. Accessing the Program Address Bus (PAB) pipeline registers from User mode when the FIFO is not halted gives

indeterminate results. The ODEC works closely with the OnCE state machine on register reads and writes. This register is not memory mapped and cannot be modified.

17.7.4 OnCE Control Register (OCR)

The 16-bit OnCE Control Register (OCR) contains bit fields determining how breakpoints are triggered, what action occurs when a OnCE event occurs, as well as controlling other miscellaneous OnCE features. **Figure 17-7** illustrates the OCR and its fields.

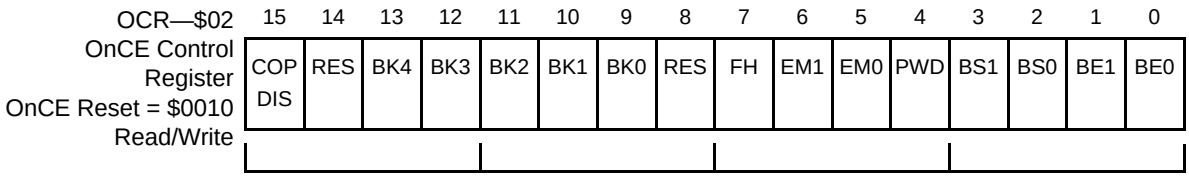


Figure 17-7. OCR Programming Model

17.7.4.1 COP Timer Disable (COPDIS)—Bit 15

The COP Timer Disable (COPDIS) bit is used to prevent the COP timer from resetting the controller when it times out. When COPDIS is cleared, the COP timer is enabled. When COPDIS is set, the COP timer is disabled.

Note: When the COP Enable (CPE) bit in the COP/RTI Control (COPCTL) register is cleared, the COP timer is not enabled. In this case, the COPDIS bit has no effect on the deactivated COP timer. That is, the COP reset can not be generated. However, the COPDIS bit overrides the CPE bit when both are set. See **Section 16.8.1.4, "COP Enable (CEN)—Bit 1"** for more information.

17.7.4.2 Reserved—Bit 14

This is a reserved bit required always to be zero.

17.7.4.3 Breakpoint Configuration (BK[4:0])—Bits 13–9

The Breakpoint Configuration BK[4:0]) bits are used to configure the operation of the OnCE module when it enters the debug processing state. In addition, these bits can also be used to setup a breakpoint on one address and an interrupt on another address. **Table 17-6** lists the different breakpoint combinations available.

Table 17-6. Breakpoint Configuration Bits Encoding—Two Breakpoints

BK[4:0]	Final Trigger Combination and Actions
00000	Breakpoint 1 occurs the number of times specified in the OCNTR. Then the trigger is generated.
00001	Breakpoint 1 or Breakpoint 2 occur the number of times specified in the OCNTR. Then the trigger is generated.
00010	Breakpoint 1 and Breakpoint 2 must occur simultaneously the number of times specified in the OCNTR. Then the trigger is generated.
00100	Breakpoint 1 generates trigger; Breakpoint 2 generates a OnCE Interrupt.
01011	Breakpoint 2 occurs once, followed by Breakpoint 1 occurring the number of times specified in the OCNTR. Then the trigger is generated.
01111	Breakpoint 2 occurs the number of times specified in the OCNTR, followed by Breakpoint 1 occurring once. Then the trigger is generated.
10000	Breakpoint 1 occurs once, followed Trace Mode using the instruction count specified in the OCNTR. Then the trigger is generated.
10001	Breakpoint 1 or Breakpoint 2 occurs once, followed Trace Mode using the instruction count specified in the OCNTR. Then the trigger is generated.
10010	Breakpoint 1 and Breakpoint 2 occur simultaneously, followed Trace Mode using the instruction count specified in the OCNTR. Then the trigger is generated.
10100	Breakpoint 1 occurs once, followed Trace Mode using the instruction count specified in the OCNTR. Then the trigger is generated; Breakpoint 2 generates a OnCE Interrupt.
10111	Trace Mode using the instruction count specified in the OCNTR.
11011	Breakpoint 2 occurs once, followed by Breakpoint 1 occurring once, followed Trace Mode using the instruction count specified in the OCNTR. Then the trigger is generated.
All other encodings of BK[4:0] are illegal and may cause unpredictable results.	

Breakpoint two is a simple address compare. It is unaffected by BS/BE bits, except BE = 00 disables it. BE = 00 disables all of the above BK settings except pure Trace mode (BK[4:0] = 10111). See **Section 17.7.5, “OnCE Breakpoint 2 Control Register (OBCTL2)”** for more information.

17.7.4.4 Reserved—Bit 8

This bit is reserved or not implemented. It must always be zero.

17.7.4.5 FIFO Halt (FH)—Bit 7

The FIFO Halt (FH) bit permits a halt address capture in the OnCE Change-of-Flow FIFO (OPFIFO) register, the OnCE PAB Fetch register (OPABFR), the OnCE PAB Decode register (OPABDR), and the OnCE PAB Execute Register (OPABER) when the bit is set. The FH bit is set only by writes to the OCR, never automatically by on-chip circuitry; for example, it is a control bit, never a status bit. This provides a simple method to halt the PAB history capture without setting up event conditions using the EM bits and breakpoint circuitry.

Note: The FIFO is halted immediately after the FH bit is set. This means the FIFO can be halted in the middle of instruction execution, leading to incoherent OPFIFO register contents.

17.7.4.6 Event Modifier (EM[1:0])—Bits 6–5

The Event Modifier (EM[1:0]) bits allow different actions to occur when a OnCE event develops. OnCE events are defined to be occurrences of hardware breakpoints, software breakpoints, and traces. Each event occurrence sets the respective occurrence bit, HBO, SBO, or TO in the OnCE Status Register (OSR), regardless of EM encoding. In addition, when the $\overline{DE}$ pin is enabled, each event occurrence drives $\overline{DE}$ low until the event is rearmed, again regardless of EM encoding.

The first trigger condition of a breakpoint sequence does not set a bit in the OSR. Only completion of the final trigger condition sets the respective bit in the OSR Hardware Breakpoint Occurrence (HBO) for all but one BK encoding.

When EM[1:0] = 00, an OnCE event halts the core and the chip enters Debug mode. The core is halted on instruction boundaries only. When the core is halted, the user has access to core registers as well as data and program memory locations including peripheral memory mapped registers. The user also has the ability to execute core instructions forced into the instruction pipeline through OnCE module transfers.

If EM[1:0] = 01, the core does not halt when an event occurs. For example: the Debug mode is not entered but the OPFIFO, the OPABFR, the OPABDR, and the OPABER registers stop capturing. This allows access to the PAB history information while the application continues to execute.

If EM[1:0] = 10, the core does not halt when an event occurs, but a level one interrupt occurs with a vector at location P:$000C. This permits diagnostic subroutines execution upon event occurrences, or even to patch program memory by setting a breakpoint at the beginning of the code to be patched.

Note: Trace occurrences do not trigger vectored interrupts. Only hardware and software breakpoints are allowed OnCE events for this EM encoding.

If EM[1:0] = 11, the core does not halt and no other action is taken other than the pulsing low of $\overline{DE}$ when enabled. This encoding serves to produce an external trigger without changing OnCE module or core operation.

EM encodings 11 and 10 enable automatic event rearming. This means, eight Phi clock cycles after the event occurrence flag HBO, TO, or SBO is set, it is reset, thus rearming the event. If $\overline{DE}$ is enabled, it is asserted, or driven low for eight Phi clock cycles, then released. If another event occurs within those eight Phi clock cycles or immediately after the cycles, the occurrence flag is promptly set and $\overline{DE}$ remains low.

To rearm an event in EM encoding 00, Debug mode must be exited, typically by executing a core instruction when setting EX and GO in the OCMDR, thereby clearing the status bits and releasing $\overline{DE}$.

To rearm an event in EM encoding 10, the OCR must be written. If the OCR value is to remain constant, writing OCR with its current value successfully rearms the event. When $\overline{DE}$ is activated and released rearming occurs.

Enabling trace for EM = 11 or EM = 10 is not particularly useful. Since a trace event occurs on every instruction execution once OCNTR reaches zero, the event is continuously set, meaning $\overline{DE}$ stays low after the first event. For EM = 10, vectoring is disabled on trace occurrences, though $\overline{DE}$ goes low and stays low after the first trace occurrence. The appropriate event occurrence bit is not reset in this case, tracing and OCNTR = $0000, until the Trace mode is disabled and an event-clearing action takes place, such as exiting Debug mode or writing the OCR while in User mode.

Note: Any OCR write in User mode resets the event flags, while OCR writes in Debug mode, do not reset the event flags.

Table 17-7 summarizes the different EM encodings.

Table 17-7. Event Modifier Selection

EM[1:0]	Function	Action on Occurrence of an Event
00	Enter Debug Mode	The core halts and Debug mode is entered. FIFO capture is automatically halted. The event is rearmed by exiting Debug mode.
01	FIFO Halt	Capture by the OPABFR, the OPABDR, the OPABER, and FIFO is halted. The user program is unaffected. The event is rearmed by writing to the OCR.
10	Vector Enable	The user program is interrupted by the OnCE event. Program execution goes to P:$000C, FIFO capturing continues, the event is automatically rearmed, and the user program continues to run. Trace occurrences do not cause vectoring, though the TO bit is set and $\overline{DE}$ is asserted.
11	Rearm	The event is automatically rearmed. FIFO capture continues, and the user program continues to run.

Note: When events are rearmed, OCNTR is not reloaded with its original value. It remains at zero, and the next triggering condition generates an event.

Use care when changing EM bits. It is recommended a particular event being changed, such as trace, hardware, or software breakpoint, is disabled first. On the next OCR write, the EM bits can be modified and the event re-enabled. This is only required when the chip is not in Debug mode. Improper operation can occur if this is not followed. For example, if the FIFO has halted due to an event occurrence with EM[1:0] = 01 and the next OCR write changes EM[1:0] to 00, the chip enters Debug mode immediately. Automatic rearming is desirable if the 10 encoding is being used for a ROM patch or the 11 encoding is used for profiling code. The EM[1:0] bits add some powerful debug techniques to the OnCE module. Users can profile code more easily with the 01 encoding or perform special tasks when events occur with the 10 encoding.

The most attractive feature of the 10 encoding is the ability to patch the ROM. If a section of code in ROM is incorrect, the user can set a breakpoint at the starting address of the bad code and vector off to a program RAM location where the patch resides. There are also BK encodings available for this purpose.

The 11 encoding is useful for toggling the $\overline{DE}$ pin output. The user can count events on the $\overline{DE}$ output and determine how much time is being spent in a certain subroutine or other useful things. $\overline{DE}$ is held low for two instruction cycles to avoid transmission line problems at the board level at high internal clock speeds. This restricts event recognition to no more than one event every three instruction cycles, limiting its usefulness during tracing.

17.7.4.7 Power Down Mode (PWD)—Bit 4

The Power Down Mode (PWD) bit is a power-saving option designed to reduce running current for applications not using the OnCE module. The PWD bit can be set or reset by writing to the OCR. On hardware reset, deassertion of the $\overline{RESET}$ signal, this bit is set at Low Power mode if the JTAG TAP controller is not decoding an ENABLE_ONCE command. If the ENABLE_ONCE command is being decoded, the bit can be set or cleared only through a OnCE

module write command to the OCR. Breakpoints should be completely disabled before setting PWD, assuring proper operation.

When the OnCE module is powered down (PWD = 1), much of the OnCE module is shut down, although the following two things can still occur:

- JTAG DEBUG_REQUEST instruction still halts the core
- The OnCE module state machine is still accessible so that the user can write to the OCR

Note: DEBUG instructions executed by the core are ignored if PWD is set, and no event occurs.

17.7.4.8 Breakpoint Selection (BS[1:0])—Bits 3–2

Breakpoint selection (BS[1:0]) control bits select whether the breakpoints are recognized on program memory fetch, program memory access, or first X memory access. These bits are cleared on hardware reset, as described in **Table 17-8.** These bits are used only in determining triggering conditions for breakpoint one, not for additional future breakpoint comparators.

The BS and BE bits apply only to the breakpoint one mechanism. Breakpoint two and future breakpoint mechanisms are unaffected by BS or BE bit encodings except when all breakpoint mechanisms are disabled at BE[1:0] = 00.

Table 17-8. BS[1:0] Bit Definition

BS[1:0]	Action on Occurrence of an Event
00	Breakpoint on program memory fetch (fetch of the first word of instructions that are actually executed, not of those that are killed, not of those that are the second word of two-word instructions, and not of jumps that are not taken)
01	Breakpoint on any program memory access (any MOVEM instructions, fetches of instructions that are executed and of instructions that are killed, fetches of second word of two-word instructions, and fetches of jumps that are not taken)
10	Breakpoint on the first X memory access—XAB1/CGDB access
11	(Reserved)

Note: It is not possible to set a breakpoint on the XAB2 bus when it is used in the second access of a dual read instruction.

The BS[1:0] bits work in conjunction with the BE[1:0] bits determining how the address breakpoint hardware is setup. The decoding scheme for BS[1:0] and BE[1:0] is shown in **Table 17-9.**

Table 17-9. Breakpoint Programming with the BS[1:0] and BE[1:0] Bits

Function	BS[1:0]	BE[1:0]
Disable all breakpoints *	All combinations	00
(Reserved)	00	01
Program Instruction Fetch	00	10
(Reserved)	00	11
Any program write or fetch	01	01
Any program read or fetch	01	10
Any program access or fetch	01	11
XAB1 write	10	01
XAB1 read	10	10
XAB1 access	10	11
(Reserved)	11	01
(Reserved)	11	10
(Reserved)	11	11

* When all breakpoints are disabled with the BE[1:0] bits set to 00, the full-speed instruction tracing capability is not affected. See **Section 17.12.2.**

17.7.4.9 Breakpoint Enable (BE[1:0])—Bits 1–0

The Breakpoint Enable (BE[1:0]) control bits enable or disable the breakpoint logic selecting the type of memory operations: read, write, or access, upon operation of the breakpoint logic. *Access* means either a read/write can be taking place. These bits are cleared on hardware reset. **Table 17-10** describes the bit functions.

Table 17-10. BE[1:0] Bit Definition

BE[1:0]	Selection
00	Breakpoint Disabled
01	Breakpoint Enabled on Memory Write
10	Breakpoint Enabled on Memory Read
11	Breakpoint Enabled on Memory Access

The BE[1:0] bits work in conjunction with the BS[1:0] bits determining how the address breakpoint hardware is setup. The decoding scheme for BS[1:0] and BE[1:0] is shown in **Table 17-9.** Breakpoints should remain disabled until after the OBAR is loaded. See **Section 17.8.5, "OnCE Breakpoint and Trace Section"** and **Section 17.12.2, "Entering Debug Mode"** for a more complete description of tracing and breakpoints. Breakpoints can be disabled or enabled for one memory space.

17.7.5 OnCE Breakpoint 2 Control Register (OBCTL2)

OnCE Breakpoint Two Control (OBCTL2) register is a 3-bit register used to program breakpoint two. Please refer to **Figure 17-8.** The register can be read or written by the OnCE unit. It is used to set up the second breakpoint for breakpoint operation. This register is accessed as the lowest three bits of a 16-bit word. The upper bits are reserved and should be written with a zero to ensure future compatibility.

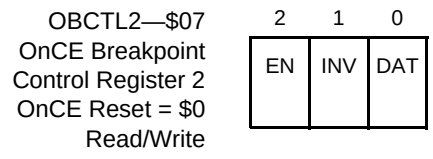


Figure 17-8. OnCE Breakpoint Control Register 2 (OBCTL2)

17.7.5.1 Reserved—Bits 15–3

This bit field is reserved. They are read as zero during read operations. These bits should be written with zero assuring future compatibility.

17.7.5.2 Enable (EN)—Bit 2

The Enable (EN) bit is used to activate the second breakpoint unit. When EN is set, the second breakpoint unit is enabled. When EN is cleared, the second breakpoint unit is disabled.

17.7.5.3 Invert (INV)—Bit 1

The Invert (INV) bit is used to specify whether to invert the result of the comparison before sending it to the breakpoint and trace units. When INV is set, the second breakpoint unit inverts the result of the comparison. Inversion is not performed when INV is cleared.

17.7.5.4 Data/Address Select (DAT)—Bit 0

The Data/address select (DAT) bit determines which bus is selected by the second breakpoint unit. When DAT is set, the second breakpoint unit examines the Core Global Data Bus (CGDB). When DAT is cleared, the Program Address Bus (PAB) is examined.

17.7.6 OnCE Status Register (OSR)

The OnCE Status Register (OSR) is shown in **Figure 17-9.** By observing the values of the five status bits in the OSR, the user can determine if the core has halted, what caused it to halt, or why the core has not halted in response to a debug request. The user can see the OSR value when shifting in a new OnCE command, writing to the OCMDR, and allowing for efficient status polling. The OSR and all other OnCE registers are inaccessible in Stop mode.

Bits	7	6	5	4	3	2	1	0
Read				OS1	OS0	TO	HBO	SBO
Write								
Reset	0	0	0	0	0	0	0	0

Figure 17-9. Once Status Register (OSR)

17.7.6.1 Reserved—Bits 7–5

This bit field is reserved future expansion. They are read as zero during controller read operations. Never write a one to these bits.

17.7.6.2 OnCE Core Status (OS[1:0])—Bits 4–3

The OnCE Core Status (OS[1:0]) bits describe the operating status of the controller core. It is recommended JTAGIR for OS[1:0] information be read, because OSR is unreadable in Stop

mode. **Table 17-11** summarizes the OS[1:0] descriptions. On transitions from 00 to 11 and from 11 to 00, there is a small chance intermediate states (01 or 10) may be captured.

Table 17-11. Core Status Bit Description

OS[1:0]	Instruction	Description
00	Normal	Controller Core Executing Instructions or in Reset
01	Stop/Wait	Controller Core in Stop or Wait Mode
10	Busy	Controller is Performing External or Peripheral Access (Wait States)
11	Debug	Controller Core Halted and in Debug Mode

Note: The OS bits are also captured by the JTAG Instruction Register (IR). See **Section 17.13.3, "Loading the JTAG Instruction Register"** for details.

17.7.6.3 Trace Occurrence (TO)—Bit 2

The read only Trace Occurrence (TO) status bit is set when a trace event occurs. This bit is cleared by hardware reset if ENABLE_ONCE is not decoded in the JTAGIR and also by event rearm conditions described in **Section 17.7.4.6, "Event Modifier (EM[1:0])—Bits 6–5".**

17.7.6.4 Hardware Breakpoint Occurrence (HBO)—Bit 1

The read only Hardware Breakpoint Occurrence (HBO) status bit is set when a OnCE hardware breakpoint event occurs. This bit is cleared by hardware reset if ENABLE_ONCE is not decoded in the JTAGIR and also by the event rearm conditions described in **Section 17.7.4.6, "Event Modifier (EM[1:0])—Bits 6–5".** Also see **Section 17.7.4.3, "Breakpoint Configuration (BK[4:0])—Bits 13–9"** to determine which encodings are defined to generate hardware breakpoint events.

17.7.6.5 Software Breakpoint Occurrence (SBO)—Bit 0

The read only Software Breakpoint Occurrence (SBO) status bit is set when a controller debug instruction is executed, a software breakpoint event, for example, except when PWD = 1 in the OCR. The SBO bit is cleared by hardware reset, provided ENABLE_ONCE is not decoded in the JTAGIR. It is also cleared by the event rearm conditions described in **Section 17.7.4.6, "Event Modifier (EM[1:0])—Bits 6–5".** The EM[1:0] bits determine if the core or the FIFO is halted.

17.8 Breakpoint and Trace Registers

The following subsections describe these OnCE breakpoint and trace registers:

- OnCE Breakpoint/Trace Counter Register (OCNTR)
- OnCE Memory Address Latch (OMAL) *(not memory mapped)*
- OnCE Breakpoint Address Register (OBAR)
- OnCE Memory Address Comparator (OMAC) *(not memory mapped)*

17.8.1 OnCE Breakpoint/Trace Counter Register (OCNTR)

The OnCE Breakpoint/Trace Counter Register (OCNTR) is an 8-bit counter permitting as many as 256 valid address comparisons or instruction executions. The process depends on whether it is configured as a breakpoint or trace counter and occurs before a OnCE event. In its most common use, OCNTR is $00 and the first valid address compare or instruction execution halts the core. If the user prefers to generate a OnCE event on *n* valid address compares or *n* instructions, OCNTR is loaded with *n* – 1. Again, if the Trace mode is selected and EM[1:0] is not cleared, only *n* – 1 instructions must execute before an event occurs.

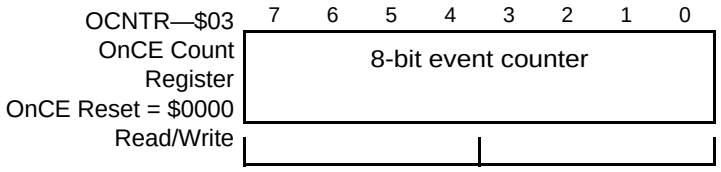


Figure 17-10. OnCE Breakpoint/Trace Counter (OCNTR)

When used as a breakpoint counter, the OCNTR becomes a powerful tool to debug real-time interrupt sequences such as servicing an A/D or D/A converter or stopping after a specific number of transfers from a peripheral have occurred. OCNTR is cleared by hardware reset provided ENABLE_ONCE is not decoded in the JTAGIR.

When used as a Trace mode counter, the OCNTR permits single-step through code. Meaning after each controller instruction is executed, Debug mode is reentered, allowing for display of the processor state after each instruction. By placing larger values in OCNTR, multiple instructions can be executed at full core speed before reentering Debug mode. Trace mode is most useful when the EM bits are set for Debug mode entry, but there is an ability to halt the FIFO or auto rearm the events with a $\overline{DE}$ toggle. The trace feature helps the software developer debug sections of code without a normal flow, or are getting hung up in infinite loops. Using the trace counter also permits time critical areas of code to be debugged.

It is important to note the breakpoint/trace logic is only enabled for instructions executed outside of Debug mode. Instructions forced into the pipeline via the OnCE module do not cause trace or breakpoint events.

17.8.2 OnCE Memory Address Latch Register (OMAL)

The OnCE Memory Address Latch (OMAL) register is a 16-bit register latching the PAB or XAB1 on every cycle. This latching is disabled if the OnCE module is powered down with the OCR's PWD bit. This register is not memory mapped. This is not a read/write register.

Note: The OMAL register does not latch the XAB2 bus. As a result, it is not possible to set an address breakpoint on any access done on the XAB2/XDB2 bus pair used for the second read in any dual-read instruction.

17.8.3 OnCE Breakpoint Address Register (OBAR)

The OnCE Breakpoint Address Register (OBAR) is a 16-bit OnCE register storing the memory breakpoint address. OBAR is available for write operations only through the JTAG/OnCE serial interface. Before enabling breakpoints, by writing to OCR, OBAR should be written with its proper value. OBAR is for breakpoint one only and has no effect on breakpoint two.

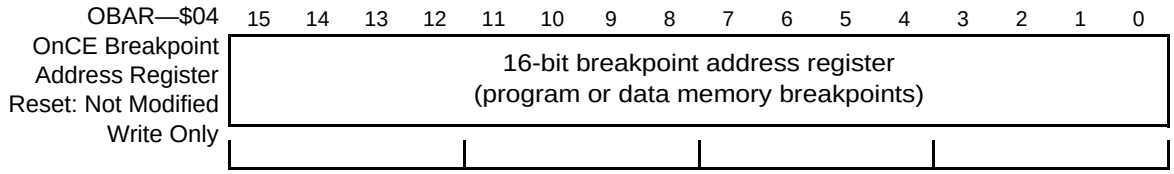


Figure 17-11. OnCE Breakpoint Address Register (OBAR)

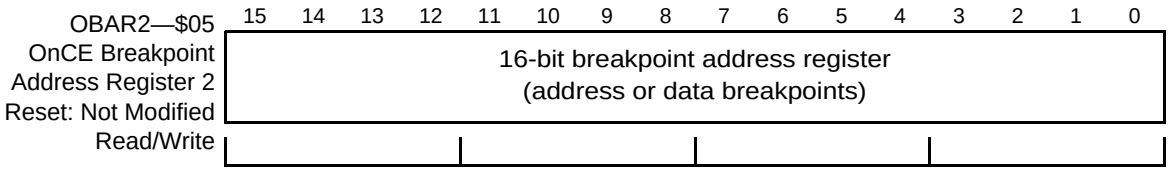


Figure 17-12. OnCE Breakpoint Address Register 2 (OBAR2)

17.8.4 OnCE Memory Address Comparator (OMAC)

The OnCE Memory Address Comparator (OMAC) is a 16-bit comparator and compares the current memory address (stored by OMAL) with memory address register (OBAR). If OMAC is

equal to OMAL, then the comparator delivers a signal indicating the breakpoint address has been reached. This register is not memory mapped. It is not a read/write register.

17.8.5 OnCE Breakpoint and Trace Section

Two capabilities useful for real-time debugging of embedded control applications are address breakpoints and full-speed instruction tracing. Traditionally, processors had set a breakpoint in program memory by replacing the instruction at the breakpoint address with an illegal instruction causing a breakpoint exception. This technique is limiting because breakpoints can only be set in RAM at the beginning of an opcode and not on an operand. Additionally, this technique does not permit breakpoints to be set on data memory locations. The 56F80x instead provides on-chip address comparison hardware for setting breakpoints on program or data memory accesses. This grants breakpoints to be set on program ROM as well as program RAM locations. Breakpoints can be programmed for reads, writes, program fetches, or memory accesses using the OCR's BS and BE bits. Please see **Section 17.7.4.8, "Breakpoint Selection (BS[1:0])—Bits 3–2".**

The breakpoint logic can be enabled for the following:

- Program instruction fetches
- Program memory accesses via the MOVE (M) instruction (read, write, or access)
- X data memory accesses (read, write, or access)
- On-chip peripheral register accesses (read, write, or access)
- On either of two program memory breakpoints (i.e., on either of two instructions)
- On a single bit or field of bits in a data value at a particular address in data memory
- On a program memory or data memory location (on XAB1)
- On a sequence of two breakpoints

Breakpoints are also possible during on-chip peripheral register accesses because these are implemented as memory-mapped registers in the X data space.

In addition, the 56F80x OnCE module provides a full-speed tracing capability, the capability to execute as many as 256 instructions at full speed before reentering the Debug processing state, or simply generate a OnCE event. This permits a single-step program in the simplest case, when the counter is set for one occurrence or to execute many instructions at full speed before returning to Debug mode.

Breakpoint logic and trace logic have been designed to work together making it possible to set up more sophisticated trigger conditions and combining as many as two breakpoints and trace logic. Individual events and conditions leading to triggering can also be modified.

While debugging, sometimes not enough breakpoints are available. In this case, the core-based debug instruction can be substituted for the desired breakpoint location. The JTAG instruction DEBUG_REQUEST (0111) can be used to force Debug mode.

17.9 Pipeline Registers

The OnCE module provides a halt capability of the controller core on any instruction boundary. Upon halting the core, instructions can be executed from Debug mode, giving access to on-chip memory and registers. These register values can be brought out through the OnCE module by executing a sequence of controller instructions moving values to Peripheral Global Data Bus (PGDB) and followed by OnCE commands to read the OPGDBR. This register is detailed in **Section 17.9.6.**

Executing instructions from Debug mode destroys the pipeline information. The OnCE module provides a means to preserve the pipeline when entering Debug mode and restoring it while leaving the Debug mode.

A restricted set of one- and two-word instructions can be executed from Debug mode. Three-word instructions cannot be forced into the pipeline. But the pipeline can be restored regardless whether the next instruction to be executed is one, two, or three words long.

The OnCE module provides the following pipeline registers:

- OnCE PAB Fetch Register (OPABFR)
- OnCE PAB Decode Register (OPABDR)
- OnCE PAB Execute Register (OPABER)
- OnCE PGDB Register (OPDBR)
- OnCE PGDB Register (OPGDBR)
- OnCE PAB Change-of-Flow FIFO Register (OPFIFO) *(not memory mapped)*

17.9.1 OnCE PAB Fetch Register (OPABFR)

OnCE PAB Fetch Register (OPABFR) is a read only. The 16-bit latch stores the address of the last fetched instruction before entering Debug mode. It holds both opcode and operand addresses. OPABFR is available for read operations only through the serial interface. This register is not affected by the operations performed during Debug mode.

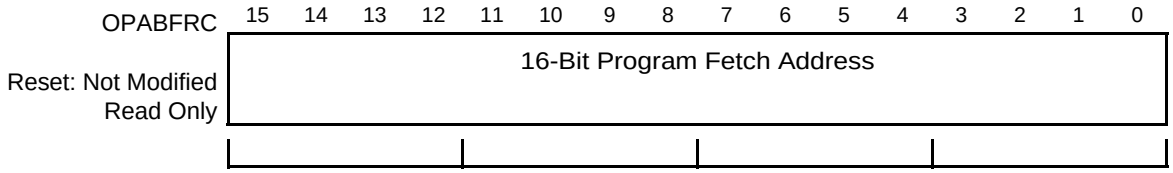


Figure 17-13. Once PAB Fetch Register (OPABFR)

17.9.2 OnCE PAB Decode Register (OPABDR)

The 16-bit OnCE PAB Decode Register (OPABDR) stores the opcode address of the instruction currently in the instruction latch, the Program Counter (PC) value. This instruction would have been decoded if the chip had not entered Debug mode. OPABDR is available for read operations only through the JTAG/OnCE port. This register is not affected by the operations performed during Debug mode.

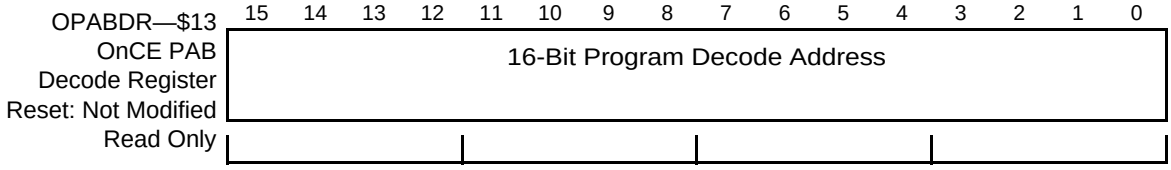


Figure 17-14. OnCE PAB Decode Register (OPABDR)

17.9.3 OnCE PAB Execute Register (OPABER)

The 16-bit OnCE PAB Execute Register (OPABER) stores the opcode address of the last instruction executed before entering Debug mode. OPABER is available for read operations only through the JTAG/OnCE port. This register is not affected by operations performed during Debug mode.

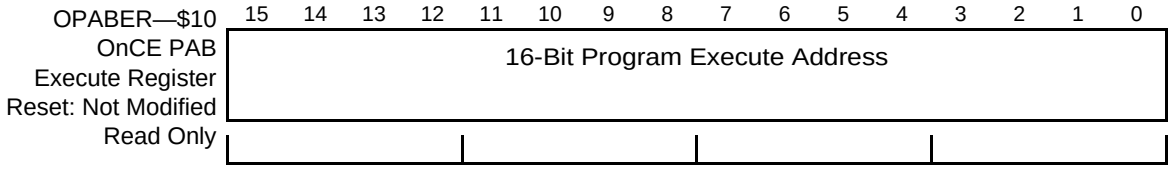


Figure 17-15. OnCE PAB Execute Register (OPABER)

17.9.4 OnCE PAB Change-of-Flow FIFO (OPFIFO)

The OnCE PAB Change-of-Flow FIFO (OPFIFO) register consists of multiple 16-bit register locations, though all locations are accessed through the same address as RS[4:0] in the OCMDR. The registers are serially available for read to the command controller through their common OPFIFO address. The OPFIFO is not affected by the operations performed during Debug mode, except for the shifting performed after reading an OPFIFO value. This register is not memory mapped and bits cannot be written to or read.

17.9.5 OnCE PDB Register (OPDBR)

The OnCE PDB (OPDBR) is a read/write register, 16-bit latch capable of storing the value of the Program Data Bus (PDB). The PDB is generated by the last program memory access of the controller before the Debug mode is entered. OPDBR is available only for read/write operations through the JTAG/OnCE serial interface and only when the chip is in Debug mode. Any attempted read of OPDBR when the chip is not in Debug mode results in the JTAG shifter capturing and shifting unspecified data. Similarly, any attempted write has no effect.

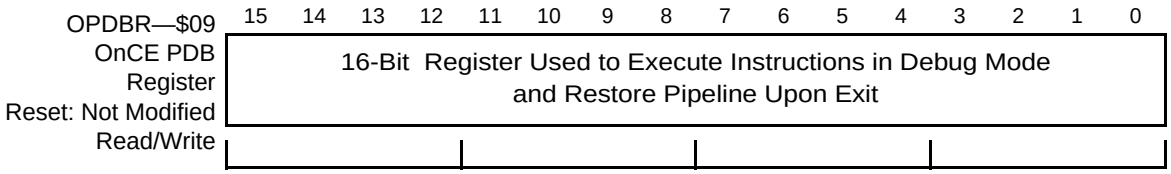


Figure 17-16. OnCE PDB Register (OPDBR)

Immediately upon entering Debug mode, OPDBR should be read out via a OnCE command by selecting OPDBR for read. This value *must then be saved externally* if the pipeline is to be restored, as is typically the case. Any OPGDBR access corrupts PDB, so if OPDBR is to be saved, it must be saved before OPGDBR read/writes.

To restore the pipeline, regardless where Debug mode was entered in the instruction flow, the same sequence must take place.

- First, the value read out of OPBDR upon entry to the Debug mode is written back to OPDBR with GO = EX = 0
- Next, that same value is written again to OPDBR, but this time with GO = EX = 1

The first write is necessary to restore PAB. The old PAB value is saved in the FIFO and restored on the first write. It is not restored directly by the user. The second write actually restores OPDBR and the GO = EX = 1 restarts the core.

To force execution of a one-word instruction from Debug mode, write the OPDBR with the opcode of the instruction to be executed and set GO = 1 and EX = 0. The instruction then implements while executing, OS[1:0] = 00. Upon completion, OS[1:0] = 11, Debug mode. Poll JTAGIR to determine if the instruction has completed. The period of time OS[1:0] = 00 is typically unnoticeably small. By the time JTAGIR polls status and is read, OS[1:0] = 11. The only time this is not true is on chips with mechanisms extending wait states infinitely, for example: transfer acknowledge pins. In that case, polling is necessary. Only a restricted set of one-word instructions can be executed from Debug mode.

To force execution of a two-word instruction from Debug mode, write the OPDBR with the opcode of the instruction to be executed and set GO = EX = 0. Next, write OPDBR with the operand with GO = 1 and EX = 0. The instruction then executes. As in the one-word case, JTAGIR should be polled for status. Only a restricted set of two-word instructions can be executed from Debug mode.

The set of supported instructions for execution from Debug mode, GO but not EX are:

- JMP #xxxx

- MOVE #xxxx,register
- MOVE register,x:$ffff
- MOVE register,register
- MOVE register,x:(r0)+
- MOVE x:(r0)+,register
- MOVE register,p:(r0)+
- MOVE p:(r0)+,register

Note: *r0* can be any of the *r* registers. Execution of other controller instructions is possible, but only the above are specified and supported. Three-word instructions cannot be executed from Debug mode.

17.9.6 OnCE PGDB Register (OPGDBR)

The OnCE PGDB Register (OPGDBR) is a *read-only*, 16-bit latch stowing the value of the global data bus (GDB) upon entry into Debug mode. The OPGDBR is available for read operations only through the serial interface. The OPGDBR is required as a means of passing information between the chip and the command controller. It is typically used by executing a move reg,X:$FFFF from Debug mode. The value in *reg* is read out onto PGDB. The value can then be accessed via a OnCE command selecting the OPGDBR for read.

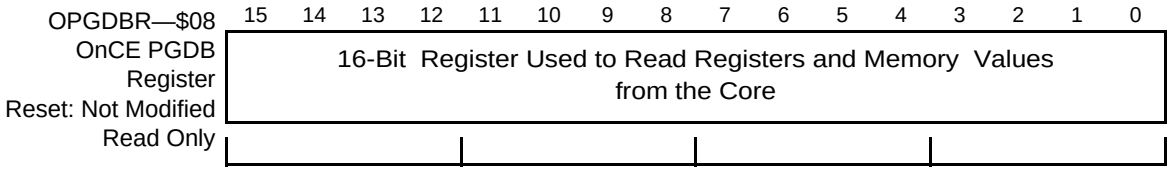


Figure 17-17. OnCE PDGB Register (OPGDBR)

The OPGDBR is a temporary storage register where the last value written to the PGDB is found. Executing the move reg,X:$FFFF loads the OPGDBR with the correct value. When the next controller instruction is executed, modifying the PGDB after it has executed the move reg,X:$FFFF instruction, the OPGDBR holds the newly modified PGDB value. An example of a modifying instruction of the PGDB bus is instruction writing to any of the peripheral memory-mapped registers on a controller.

The OPGDBR is available for read operations only through the JTAG/OnCE serial interface, but only when the chip is in the Debug mode. Any attempted read of the OPGDBR when the chip is not in the Debug mode results in the JTAG shifter capturing and shifting unspecified data.

Note: The OPGDBR accesses corrupt PDB. Therefore, if the user needs to save the value on PDB, an OPDBR read should be executed before the first OPGDBR access in any debug session.

17.9.7 OnCE FIFO History Buffer

To aid debugging activity and keep track of the program flow, a read only FIFO buffer is provided. The FIFO stores PAB values from the instruction flow. The FIFO consists of Fetch, Decode, and Execute registers as well as an Optional, or Peripheral, Change-of-Flow FIFO. **Figure 17-18** illustrates a block diagram of the OnCE FIFO history buffer.

The following instructions are considered to be change-of-flow:

- BRA
- JMP
- BCC (with condition true)
- JCC (with condition true)
- BRSET
- BRCLR
- RTS
- RTI
- JSR

Note: Addresses of JSR instructions at interrupt vector locations are not stored in the change-of-flow FIFO.

When one of the listed instructions is executed, its opcode address is immediately placed in the top location of the change-of-flow FIFO, as well as being placed in OPABER. Previous addresses placed in the OPFIFO are shifted down one location and the oldest address is overwritten. The OPABDR holds the PC value. If the core has been halted, the next opcode to be executed resides at the memory location pointed to by OPABDR.

Reads of the OPFIFO register return the oldest value in the OPFIFO first. The next read returns the next oldest. The *n*th read in an *n*-deep OPFIFO returns the latest change-of-flow address. It is recommended all OPFIFO locations are read, for example, *n* reads for an *n*-deep OPFIFO, so the oldest-to-newest ordering is maintained when address capture resumes.

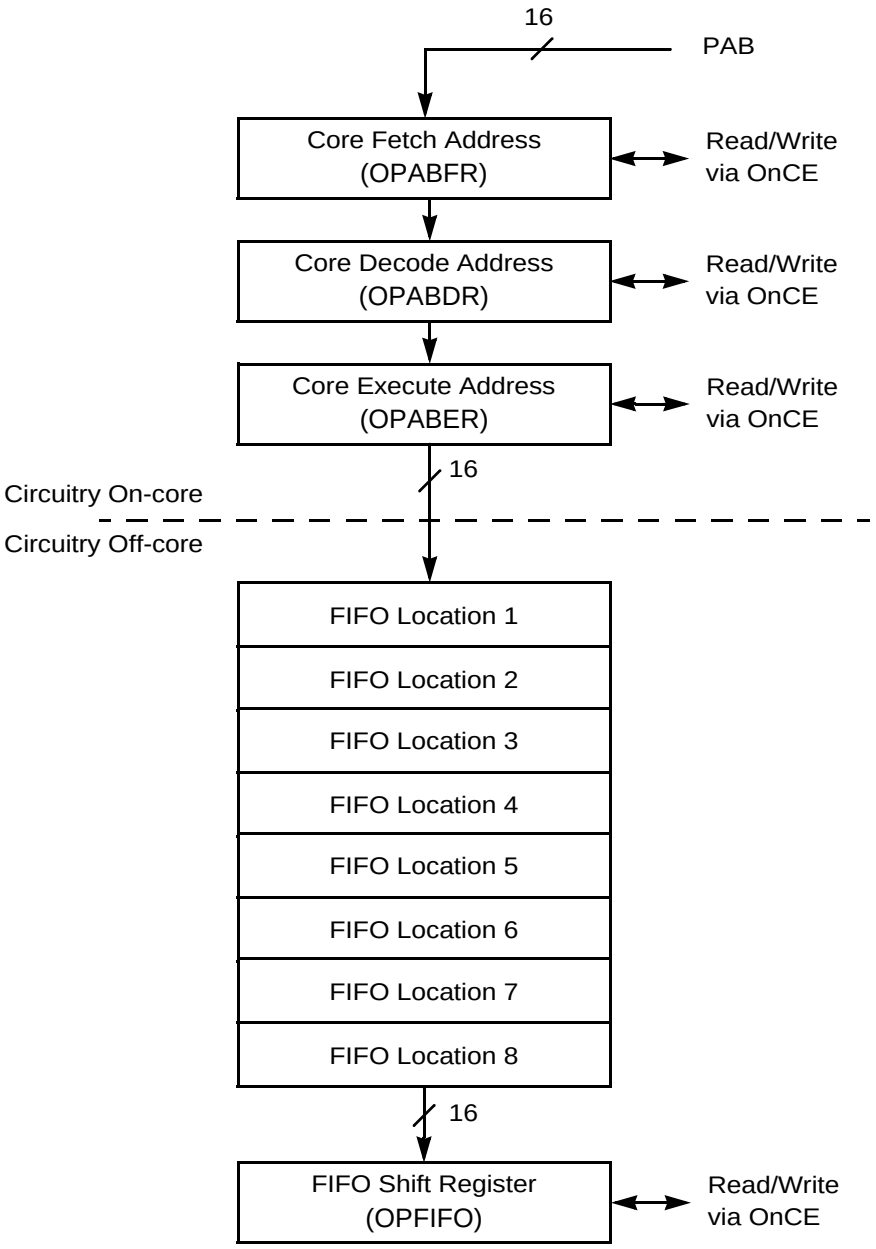


Figure 17-18. OnCE FIFO History Buffer

The change of flow nature of the FIFO begins *after* the OPABER and not the fetch, decode, or execute registers. Thus, changes of flow affect only the contents of the OPFIFO.

When the OPFIFO is halted in response to setting the FH bit, PAB capture halts immediately and transfers in progress can be interrupted. That means while determinate values are in the registers, these values may not provide entirely coherent information regarding the recent history of program flow.

Further, the state of the OPFIFO can be different when it is halted because of an event occurring when EM = 01 than when it is halted with the core due to an event occurring when EM = 00.

17.10 Breakpoint 2 Architecture

All DSP56800 devices contain a breakpoint one unit. The 56F80x provides a breakpoint two unit providing greater flexibility in setting breakpoints. Adding a second breakpoint greatly increases the debug capability of the device. It allows the following additional breakpoints to detect even more complex events:

- On either of two program memory breakpoints, such as on either of two instructions
- On a data value at a particular address in data memory
- On a bit or field of bits in a data value at a particular address in data memory
- On a program memory or data memory location (on XAB1)
- On a sequence of two breakpoints

Upon detecting a valid event, the OnCE module then performs one of the following four actions:

- Halt the controller core and enter the debug processing state
- Interrupt the controller core
- Halt the OnCE FIFO, but let the controller core continue operation
- Rearm the trigger mechanism and toggle the $\overline{\text{DE}}$ pin

The breakpoint one unit is used in conjunction with the breakpoint two unit for the detection of more complex trigger conditions. The breakpoint one unit is the same as found on other DSP56800 devices. The breakpoint two unit allows specifying more complex breakpoint conditions when used in conjunction with the first breakpoint. Additionally, it is possible to set up breakpoint two for interrupts while leaving breakpoint one available to the JTAG/OnCE port. When a breakpoint is determined the CGDB with the breakpoint two unit, the breakpoint condition should be qualified by an X memory access with the breakpoint one unit.

Figure 17-19 illustrates how the two breakpoint units are combined in the breakpoint and trace counter unit specifying more complex triggers to perform one of several actions upon detection of a breakpoint. In addition to simply detecting the breakpoint conditions, this unit allows the first of the two breakpoints to be qualified by the BS and BE bits found in the OCR. This allows a breakpoint to be qualified by a read/write or access condition. The second breakpoint is unaffected by these bits and merely detects the value on the appropriate bus. A counter is also available for detecting a specified occurrence of a breakpoint condition or for tracing a specified number of instructions.

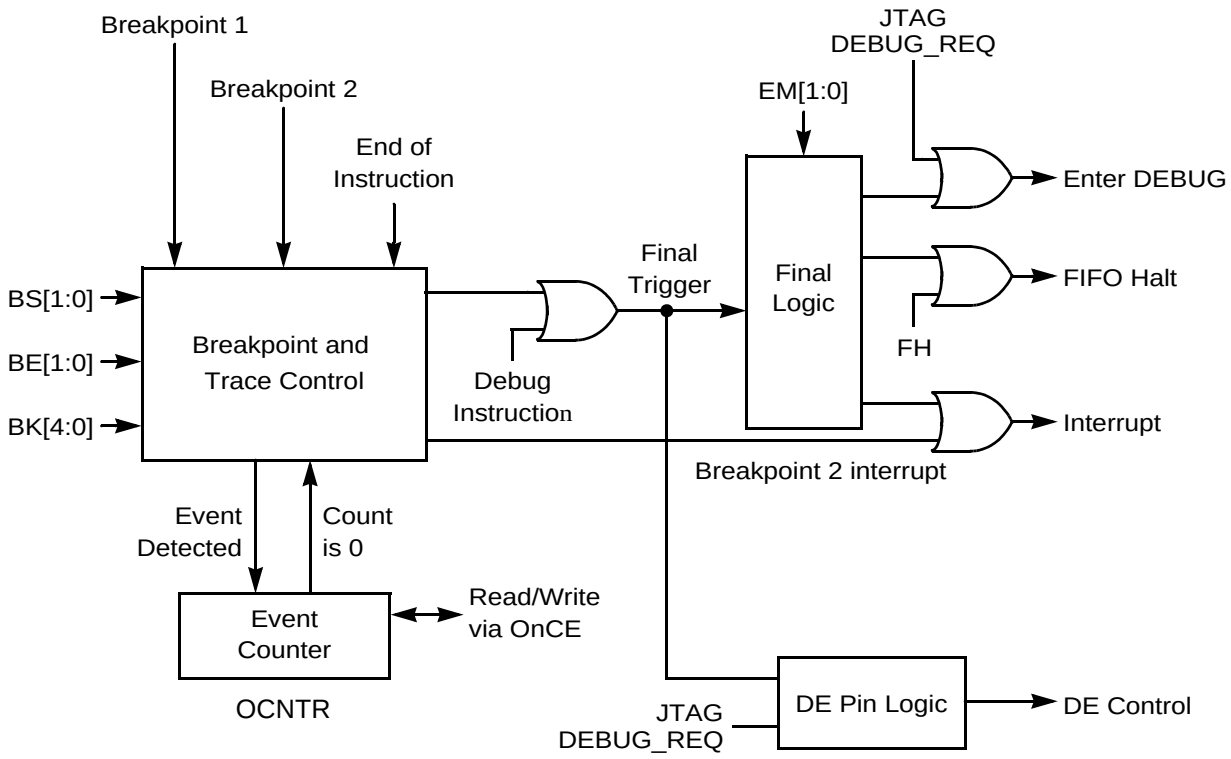


Figure 17-19. Breakpoint and Trace Counter Unit

17.11 Breakpoint Configuration

The breakpoint one unit is programmed in the OCR using the BS and BE bits. The breakpoint two unit is programmed by the OnCE Breakpoint Two Control (OBCTL2) register, located within the breakpoint two unit. BK bits in the OCR specify how the two breakpoints are configured to generate trigger and interrupt conditions. However, the action performed when a final trigger is detected is specified by the EM bits in the OCR. **Figure 17-20** illustrates the breakpoint programming model for the dual breakpoint system.

Note: Registers OMAC,OMAL, OMAC2, and OMAL2 are not memory mapped and their bits cannot modified or read.

OCR—$02
OnCE Control Register
OnCE Reset = $0010
Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
COP DIS	RES	BK 4	BK 3	BK 2	BK 1	BK 0	RES	FH	EM1	EM0	PWD	BS1	BS0	BE1	BE0

OBCTL2—$07
OnCE Breakpoint Control Register 2
OnCE Reset = $0
Read/Write

2	1	0
EN	INV	DAT

Figure 17-20. OnCE Breakpoint Programming Model

Breakpoint one unit's circuitry contains the OMAL, the OBAR, the OMAC, and the OCNTR. The OMAC, an address comparator, and the OBAR, its associated breakpoint address register, are useful in halting a program at a specific point to examine or change registers or memory. Using the OMAC to set breakpoints enables the user to set breakpoints in RAM or ROM while in any operating mode. The OBAR is dedicated to breakpoint one logic. **Figure 17-21** illustrates a block diagram of the breakpoint one unit.

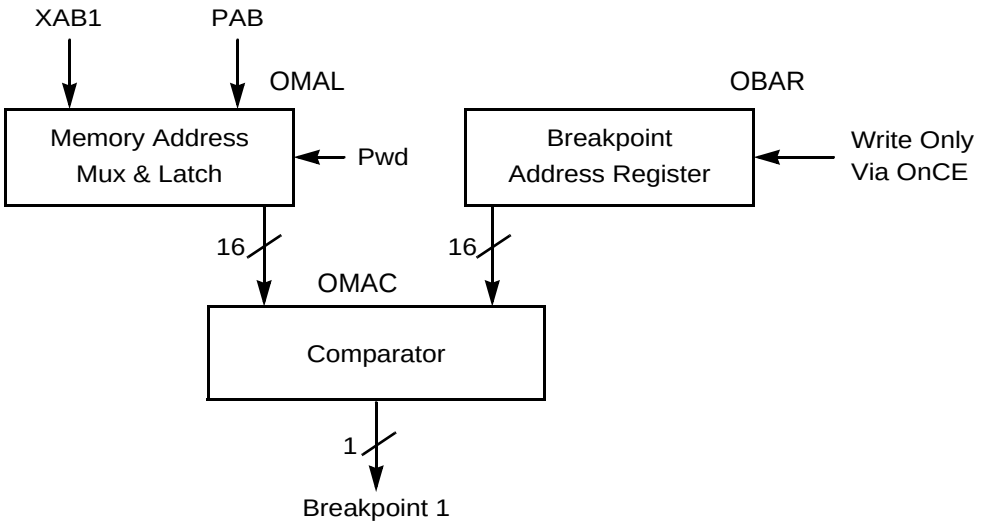


Figure 17-21. Breakpoint 1 Unit

For breakpoint one, a valid address compare is defined as follows: the value in OBAR matches the value on PAB or XAB1 while meeting the breakpoint conditions specified by the BE/BS bit combination. A valid address compare for breakpoint one can then do a few things based on BK encodings and the state of OCNTR. BK could dictate the first valid address comparison enables trace to decrement OCNTR. BK could also dictate a valid address compare directly decrements

OCNTR. If OCNTR = 0, because of previous decrements or a direct OCNTR write, one more valid address compare can be set to generate a hardware breakpoint event. In this case, HBO is set, the $\overline{DE}$ pin is asserted, when activated, and the EM bits determine whether to halt the core or only the FIFO.

Figure 17-22 provides a block diagram of the breakpoint two unit. This unit also has its own address register OBAR2, similar to breakpoint one's OBAR, and address comparator OMAC2, similar to breakpoint one's OMAC. If BE = 00, the breakpoint two unit is disabled other BE encodings and all BS encodings refer only to breakpoint one functionality. The BK encoding selects which, if any, breakpoint unit or combination of units, and/or, decrement OCNTR. The breakpoint two unit operates in a similar manner. A valid breakpoint two address comparison occurs when the value on the PAB or CGDB matches the value in the OBAR2 for the bits selected with the OnCE Breakpoint Mask Register 2 (OBMSK2).

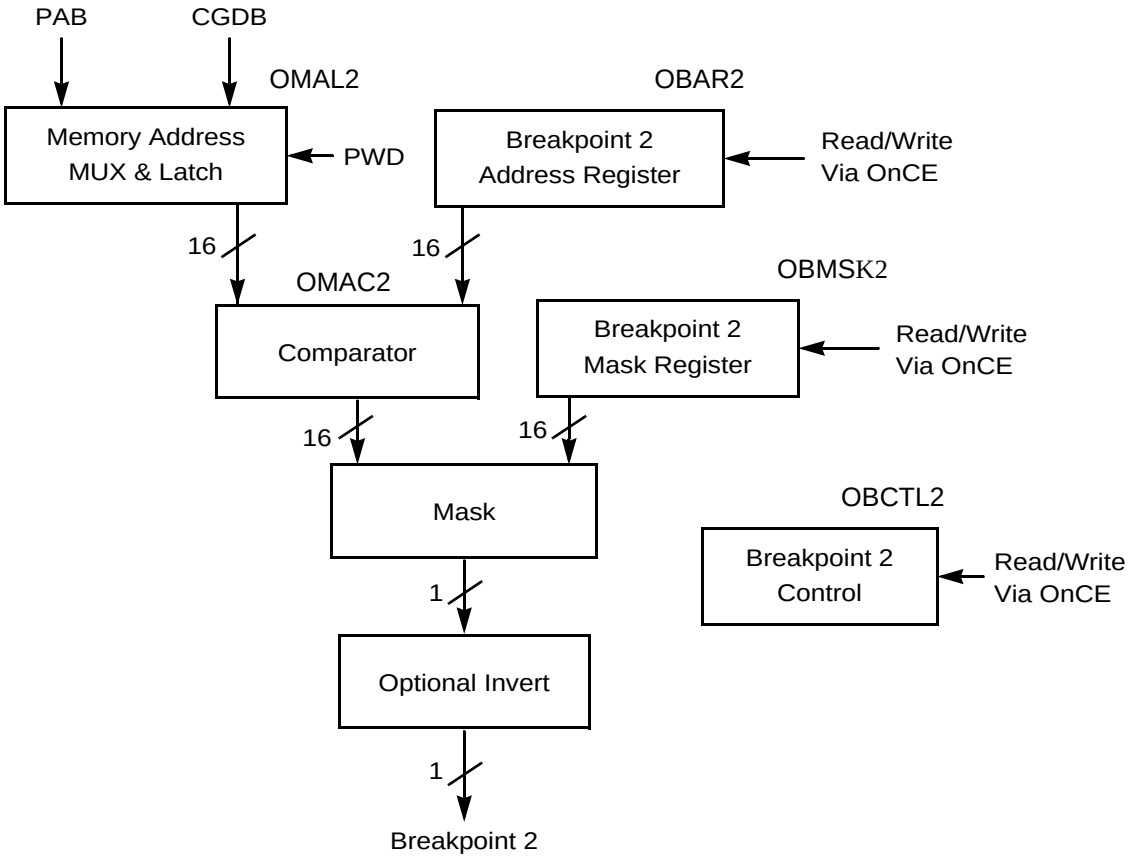


Figure 17-22. Breakpoint 2 Unit

A valid breakpoint two address comparison can do the following based on BK and the state of OCNTR:

- If OCNTR > 0, the OCNTR is decremented directly
- If OCNTR = 0, one more valid address compare can generate an HBO event

- The first valid address compare can trigger breakpoint one valid address compares to begin decrementing the OCNTR
- A valid address compare when OCNTR = 0 can trigger breakpoint one valid address compare to generate HBO event
- A valid address compare can generate a OnCE module interrupt. This is not considered an event, since no event flag is set. It is useful for Programmable ROM (PROM) code patching
- The first valid address compare can trigger trace to decrement the OCNTR
- The first valid address comparison can trigger the first valid address compare on breakpoint one to allow trace to decrement the OCNTR

Note: When a breakpoint is established on the CGDB bus with this unit, the breakpoint condition is qualified by an X memory access with the first breakpoint unit.

The BE/BS bits allow conditions be define determining a valid breakpoint. Using these bits, breakpoints could be restricted to occur on first data memory reads or only on fetched executed instructions. Again, these bits pertain only to breakpoint one. Please refer to **Section 17.7.4.8** and **Section 17.7.4.9** to understand various encodings.

Perhaps the most important breakpoint capabilities are to break up to two program memory locations. Those memory locations would be a sequence where first one breakpoint is found and it is followed sequentially by a second breakpoint. Both breakpoints would be on a specified data memory location when a programmed value is read/written as data to that location. Further, the breakpoints' capabilities would be on a specified data memory location where a programmed value is detected only at masked bits in a data value. Upon detecting a valid event, the OnCE module then performs one of four actions as is currently available in the OnCE module:

- Halt the controller core and enter the debug processing state
- Interrupt the controller core
- Halt the OnCE FIFO, but let the controller core continue operation
- Rearm the trigger mechanism (and toggle the $\overline{\text{DE}}$ pin)

If a breakpoint is set on the last instruction in a DO loop, even if BS = 00, BE = 10, a breakpoint match occurs during the execution of the DO instruction, as well as during the execution of the instruction at the end of the DO loop.

17.11.1 Programming the Breakpoints

Breakpoints and trace can be configured while the core is executing controller instructions or while the core is in reset. Complete access to the breakpoint logic, OCNTR, OBAR, and OCR, is provided during these operating conditions. The chip may be held in reset, set OCNTR, OBAR,

and OCR so Debug mode is entered on a specific condition, release reset, and debug the application. Similarly, the application can be running while the user configures breakpoints to toggle $\overline{DE}$ on each data memory access to a certain location, thereby allowing statistical information to be gathered.

In general, to set up a breakpoint, the following sequence must be performed:

1. JTAG must be decoding ENABLE_ONCE to allow OnCE module register read/writes.
2. The PWD bit in the OCR must be cleared to power up the OnCE module, and the BE[1:0] bits in the OCR should be set to 00.
3. Write the breakpoint address into the OBAR.
4. Write $n - 1$ into the OCNTR, where n is the number of valid address compares required to take place before generating a OnCE event.
5. Write the OCR to set the BE, BS, and BK bits for the desired breakpoint conditions.

When the above steps are conducted while in Debug mode, exit the Debug mode to restart the core. However, when the above steps are completed in the User mode, the breakpoint is set immediately.

Note: OnCE events can occur even if ENABLE_ONCE is not latched in the JTAGIR. This is useful in multiprocessor applications.

The first breakpoint unit is programmed in the OCR using the BS and BE bits. The second breakpoint unit is programmed by the OnCE Breakpoint Two Control (OBCTL2) register, located within the second breakpoint unit. The manner in which the two breakpoints are set up for generating triggers and interrupt conditions is specified by the BK bits in the OCR. The EM bits in the OCR specifies the action performed when a final trigger is detected.

17.11.2 OnCE Trace Logic Operation

The trace and breakpoint logics are tightly coupled, sharing resources where necessary. When BK[4:0] = 10111, OCNTR is decremented each time an instruction is executed from Normal mode. Instructions executed from Debug mode do not decrement the OCNTR. The event occurrence mechanism is slightly different for trace than for breakpoints.

For breakpoints, the event occurs when OCNTR = 0 and another valid address are compared. For trace, the event occurs, TO is set, when OCNTR first reaches zero. If the EM bits are set for entry of Debug mode, one more instruction is executed after TO is set. Therefore, if the user wants to halt the controller after executing n instructions, $n - 1$ should be placed in OCNTR, much like the breakpoint case. But if the user would like to halt only the FIFO after n instructions, n should be placed in OCNTR. This is different from the breakpoint case and occurs because the TO flag

is set when OCNTR first reaches zero. Trace events cannot cause OnCE interrupts, although TO is set and $\overline{DE}$ is asserted, pulled low, for this EM such as EM = 10 acts just like EM = 11 for trace.

Since trace events occur when OCNTR reaches zero and the Trace mode is enabled by one of the BK settings, rearming trace events responds differently than rearming breakpoint events. For example, EM = 10 and EM = 11 encodings attempt to rearm the trace event, but since the conditions are still valid for trace, TO remains set and $\overline{DE}$ remains low. Similarly, for EM = 01, FIFO halt, an OCR write attempts to clear the TO, but again the flag remains set since conditions are still valid for trace. To clear TO and capture additional FIFO values, do the following:

1. Write OCR disabling trace, FIFO begins capturing
2. Write OCNTR with desired value
3. Write OCR to enable trace
4. Poll for TO = 1

If step one above is omitted, TO is never reset and the FIFO does not begin capturing because the conditions for valid trace are still present.

Note: There are sequential breakpoints enabling the Trace mode. The Trace mode operation is identical to the BK[4:0] = 10111 operation, except the HBO bit is set.

A common use of the trace logic is to execute a single instruction (OCNTR = 0), then immediately return to the Debug mode. Upon returning to Debug mode, the user can display registers or memory locations. When this process is repeated, the user can step through individual instructions and see their effect on the state of the processor.

17.12 The Debug Processing State

A DSP56800 device in a user application can enter any of six different processing modes:

- Reset mode
- Normal mode
- Exception mode
- Wait mode
- Stop mode
- Debug mode

The first five of these operating modes are referenced in *Chapter 7, Interrupts and the Processing States,* in the *DSP56800 Family Manual (DSP56800FM/AD)*. The last processing mode, Debug mode, is described in this subsection.

Debug mode supports the on-chip emulation features of the device. In this mode, the controller core is halted and set to accept OnCE commands through the JTAG port. Once the OnCE module is set up correctly, the controller leaves Debug mode, returning control to the user program. The controller reenters Debug mode when the previously set trigger condition occurs, provided EM = 00, OnCE events cause entry to Debug mode.

Capabilities available in the Debug mode include the following:

- Read and write the OnCE registers
- Read the instruction FIFO
- Reset the OnCE event counter
- Execute a single controller instruction and return to this mode
- Execute a single controller instruction and exit this mode

17.12.1 OnCE Normal and Debug and Stop Modes

The OnCE module has three operational modes:

1. Normal
2. Debug
3. Stop

Whenever a stop instruction is executed by the controller, the OnCE module is no longer accessible. The OnCE module is in the Normal mode except when the controller enters Debug mode, or if it is in the Stop mode. The OnCE module is in the Debug mode whenever the controller enters the Debug mode. The major difference between the states is register access. The following OnCE module registers can be accessed in the Normal or Debug modes:

- OnCE Control Register (OCR)
- OnCE Status Register (OSR)
- OnCE Breakpoint and Trace Counter (OCNTR)
- OnCE Breakpoint Address Register (OBAR)
- OnCE Program Address Bus Fetch Register (OPABFR) (if FIFO halted)
- OnCE PAB Decode Register (OPABDR) (if FIFO halted)
- OnCE PAB Execute Register (OPABER) (if FIFO halted)
- OnCE PAB Change-of-Flow FIFO (OPFIFO) (if FIFO halted)

The following OnCE registers can only be accessed when the module is in Debug mode:

- OnCE Peripheral Global Data Bus Register (OPGDBR)

- OnCE Program Data Bus Register (OPDBR)

If a Stop is executed while the user is accessing OnCE in User mode, problems may occur since few or no internal clocks are running anymore. This should be avoided. Recognize this occurrence by capturing the OSR bits in the JTAGIR in capture-IR then choose to send a DEBUG_REQUEST to bring the core out of stop.

17.12.2 Entering Debug Mode

There are six ways to enter Debug mode:

1. JTAG DEBUG_REQUEST during Hardware Reset
2. JTAG DEBUG_REQUEST during Stop or Wait
3. JTAG DEBUG_REQUEST during Wait states
4. Software breakpoint (DEBUG) during normal use with PWD = 0 and EM = 00
5. Trigger events, Breakpoint/Trace modes, when EM = 00
6. Execute a controller instruction from Debug mode with EX = 0

17.12.2.1 JTAG DEBUG_REQUEST

The core reacts to a debug request by either of these sources in the same way. To send a JTAG DEBUG_REQUEST, the 0111 opcode must be shifted into the JTAGIR, then update-IR must be passed through. This instructs the core to halt and enter Debug mode. When the controller enters Debug mode in response to these requests, the $\overline{\text{DE}}$ pin is asserted, or pulled low, if it is enabled.

The JTAG/OnCE interface is accessible when $\overline{\text{RESET}}$ is asserted, provided $\overline{\text{TRST}}$ is not asserted, i.e., the JTAG port is not being reset. The user can load DEBUG_REQUEST into the JTAGIR while $\overline{\text{RESET}}$ is held low. If $\overline{\text{RESET}}$ is then disallowed, the chip exit hardware reset directly into Debug mode. After sending the DEBUG_REQUEST instruction, poll the JTAGIR to see whether the chip has entered Debug mode.

If the chip is in either Wait or Stop mode, either type of debug request brings the chip out of these modes, much like an external interrupt. Upon leaving the Wait or Stop modes, the chip enters Debug mode. As always, the user should poll JTAGIR for status after sending the debug request. It is important to remember the OSR cannot be polled during the Stop mode because no OnCE module access is allowed. However, JTAG access is allowed.

If the chip is in wait states, because of a non-zero value in BCR or transfer acknowledge deassertion on chips having this function, the debug request is latched and the core halts upon execution of the instruction in Wait states. The period of time between debug request and the OS bits being set to 11, Debug mode, is typically much shorter than the time it takes to poll status in JTAGIR, meaning OS = 11 on the first poll. The only time this is not the case is when a transfer

acknowledge is generating a large or infinite number of Wait states. For this reason, *polling for OS = 11 is always recommended*.

Sending a debug request when the chip is in the Normal mode, results in the chip entering Debug mode as soon as the instruction currently executing finishes. Again, the JTAGIR should be polled for status. Please refer to **Section 18.6, "JTAG Port Architecture"** for information about using the JTAG TAP controller and its instructions.

17.12.2.2 Software Request During Normal Activity

Upon executing the DEBUG instruction, the chip enters the debug processing state provided PWD = 0 and EM = 00.

17.12.2.3 Trigger Events (Breakpoint/Trace Modes)

The 56F80x allows configuration of specific trigger events. These events can include breakpoints, Trace modes, or combinations of breakpoints and Trace mode operations. The following conditions must occur to halt the core due to breakpoint/trace:

- EM = 00
- If OCNTR = 0, breakpoints are enabled (BE not 00), the next valid address compare causes the core to halt, provided it is not the initial enabling breakpoint in a sequential breakpoint
- If OCNTR = 0, Trace mode is selected, one of the BK encodings with BK4 = 1, the next instruction executed causes the core to halt

17.12.2.4 Re-entering Debug Mode with EX = 0

If a controller instruction is executed from Debug mode with EX = 0, Debug mode is automatically entered a second time after the instruction finishes executing. When the instruction is being executed, the core is not in Debug mode. The OS[1:0] bits reflect this state. This change in status is typically not observable, because the core leaves and reenters Debug mode in a very short time. Still, polling for status in JTAGIR is recommended to guarantee the chip is in Debug mode.

17.12.2.5 Exiting Debug Mode

There are three ways to exit Debug mode:

- Restore the pipeline by writing original OPDBR value back to OPDBR twice; first with GO = EX = 0 and last with GO = EX = 1. PAB is restored from OPABFR so that fetching continues from the correct address

- Change program flow by writing jmp opcode to OPDBR with GO = EX = 0 and then writing target address to OPDBR with GO = 1, EX = 0. Next, write a NOP to OPDBR with GO = EX = 1
- Hardware reset (assertion of $\overline{\text{RESET}}$) brings the chip out of Debug mode provided DEBUG_REQUEST is not decoded in the JTAGI.

17.13 Accessing the OnCE Module

This sub-section describes useful example sequences involving the JTAG/OnCE interface. The sequences are described in a hierarchical manner. Low-level sequences describe basic operations such as JTAG instruction and data register accesses. Building on this, the second group of sequences describe more complicated sequences. For example, OnCE command entry and status polling. The final set builds further on the lower-level sequences to describe how to display core registers, set breakpoints, and change memory.

17.13.1 Primitive JTAG Sequences

The JTAG/OnCE serial protocol is identical to the protocol described in the *IEEE 1149.1a-1993 Standard Test Access Port and Boundary Scan Architecture. It involves the control of fou*r input pins: $\overline{\text{TRST}}$, actually bidirectional, TDI, TMS and TCK, and the observance of one output pin, TDO. TDI and TDO are the serial input and output, respectively. TCK is the serial clock and TMS is an input used to selectively step through the JTAG state machine. $\overline{\text{TRST}}$ is an asynchronous reset of the JTAG port.

The following descriptions refer to states in the JTAG state machine diagram described in **Figure 18-8.** Please refer to this diagram or to the IEEE 1149.1a-1993 document.

17.13.2 Entering the JTAG Test-Logic-Reset State

The test-logic-reset state is the convenient starting point for primitive JTAG/OnCE module sequences. While in this state, JTAG is reset. This means TDO is disabled. No shifting is taking place and the JTAGIR is decoding the IDCODE instruction. This state is entered only on power-up, or during the initial phase of a series of OnCE module sequences. Additionally, this state can be entered to get JTAG into a known state. To enter the test-logic-reset state on power-up, both $\overline{\text{TRST}}$ and $\overline{\text{RESET}}$ should be asserted, as shown in **Figure 17-23.** See the appropriate technical data sheet for minimum assertion pulse widths. $\overline{\text{TRST}}$ can change at any time with respect to TCK.

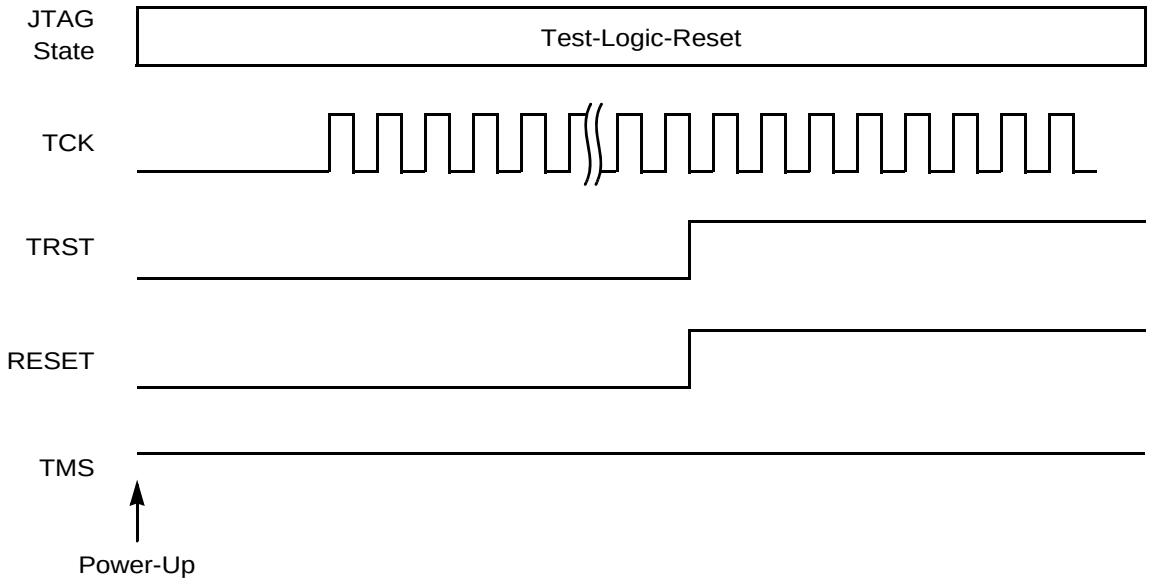


Figure 17-23. Entering the JTAG Test Logic-Reset State

At any other time, the test-logic-reset state can be entered by holding TMS high for five or more TCK pulses, as shown in **Figure 17-24.** TMS is sampled by the chip on the rising edge of TCK. To explicitly show this timing, TMS is shown to change on the falling edge of TCK. The JTAG state machine changes state on rising edges of TCK, or on $\overline{\text{TRST}}$ assertion and power-up. This sequence provides a simple way of resetting JTAG into a known state.

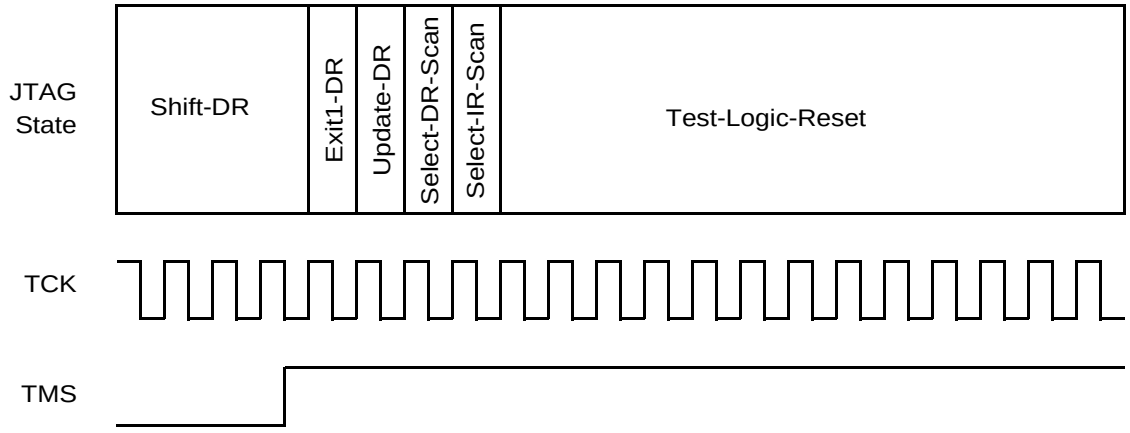


Figure 17-24. Holding TMS High to Enter Test-Logic-Reset State

17.13.3 Loading the JTAG Instruction Register

JTAG instructions are loaded through the JTAGIR path in the state machine. Shifting takes place in the Shift-IR path while the actual instruction register update occurs on Update-IR.

Note: Bit order for JTAG/OnCE shifting is always as shown in **Figure 17-25.**

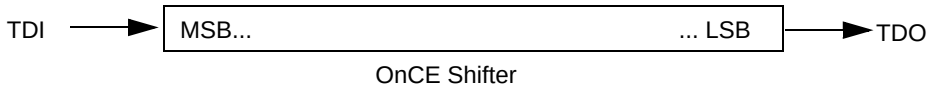


Figure 17-25. Bit Order for JTAG/OnCE Shifting

Note: OnCE instructions are loaded into the instruction register through the DR path of the state machine. JTAG instructions are loaded in the IR path.

The following sequence shows how to load the instruction DEBUG_REQUEST into the JTAGIR, as shown in **Figure 17-26.**

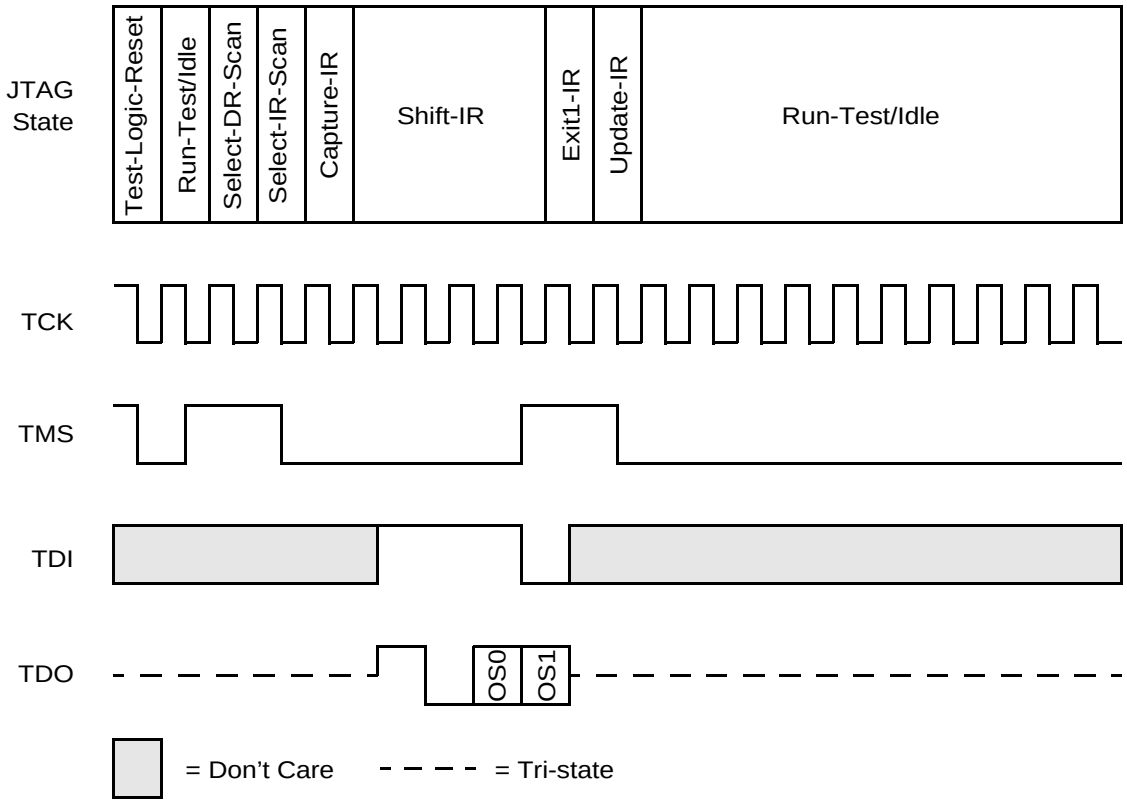


Figure 17-26. Loading DEBUG_REQUEST

During Shift-IR, a 4-bit shifter is connected between TDI and TDO. The opcode shifted in is loaded into the JTAGIR on Update-IR. The data shifted out is captured on Capture-IR. TDI, like TMS, is sampled on the rising edge of TCK and changes on the falling edge of TCK. TDI is first sampled on the TCK rising edge following entry into the Shift-IR state. It is last sampled on the

TCK rising edge when entering exit1-IR. TDO changes on falling edges of TCK in Shift-IR. It switches back to tri-state on the falling edge of TCK in Exit1-IR. The first two bits shifted out of TDO are constant: first one, then zero. The following two bits are the OnCE status bits, OS[1:0].

Note: The value in OS[1:0] is shifted out whenever a new JTAG instruction is shifted in. This provides a convenient means to obtain status information.

17.13.4 Accessing a JTAG Data Register

JTAG Data Registers are loaded via the DR path in the state machine. Shifting takes place in the Shift-DR state and the shifter connected between TDI and TDO is selected by the instruction decoded in the JTAGIR. When applicable, data is captured in the selected register on Capture-DR, shifted out on Shift-DR while new data is shifted in, and finally the new data is loaded into the selected register on Update-DR.

Assume BYPASS has been loaded into the JTAGIR and the state machine is in the run-test/idle state. In BYPASS, a one-bit register is selected as the data register. The following sequence shows how data can be shifted through the BYPASS register, illustrated in **Figure 17-27.**

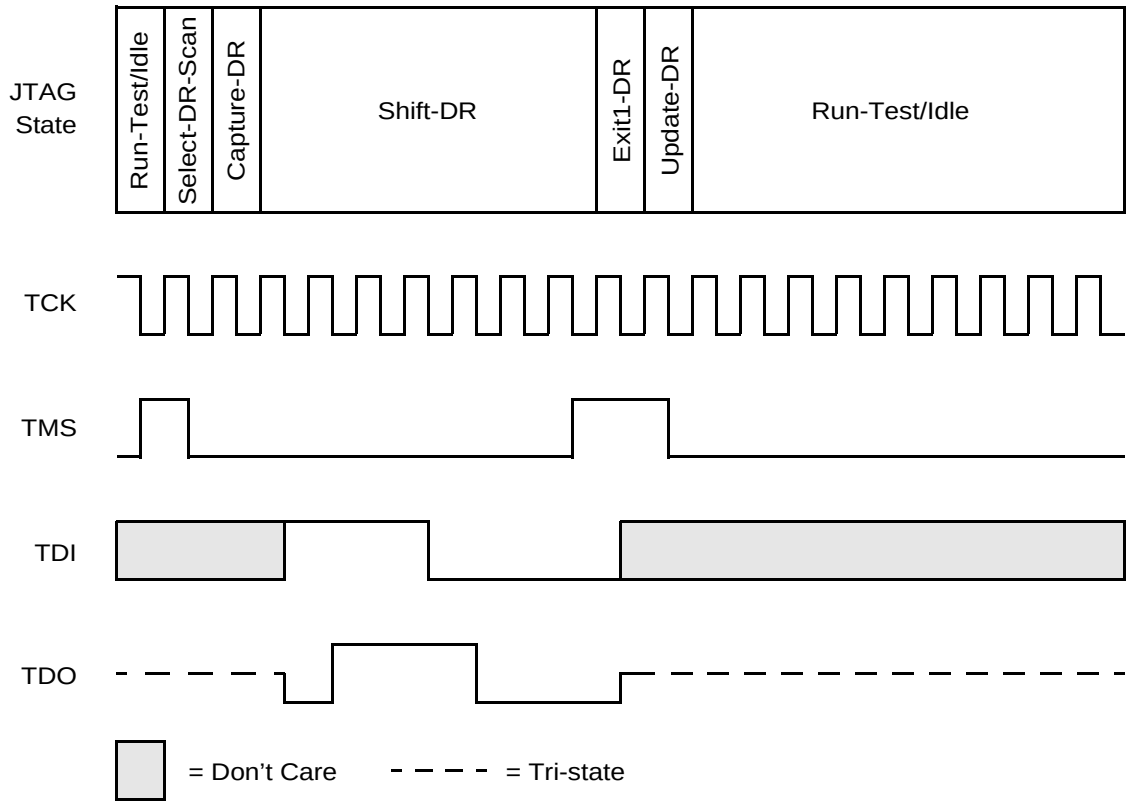


Figure 17-27. Shifting Data through the BYPASS Register

The first bit shifted out of TDO is a constant zero because the BYPASS register captures zero on Capture-DR per the IEEE standard. The ensuing bits are just the bits shifted into TDI delayed by one period.

17.13.4.1 JTAG/OnCE Interaction: Basic Sequences

JTAG controls the OnCE module by way of two basic JTAG instructions: DEBUG_REQUEST and ENABLE_ONCE. DEBUG_REQUEST provides a simple way to halt the controller core. The halt request is latched in the OnCE module so a new JTAG instruction can be shifted in without waiting for the request to be granted. After DEBUG_REQUEST has been shifted in, JTAGIR status polling takes place to see if the request has been granted. This polling sequence is described in **Section 17.13.4.5, "JTAGIR Status Polling".** Like any other JTAG instruction, DEBUG_REQUEST selects a data register to be connected between TDI and TDO in the DR path. The one-bit BYPASS register is selected. Values shifted into the BYPASS register have no effect on the OnCE logic.

ENABLE_ONCE is decoded in the JTAGIR for most of the time during a OnCE sequence. When ENABLE_ONCE is decoded, access to the OnCE registers is available through the DR path. Depending on which register is being accessed, the shifter connected between TDI and TDO during shift-DR can be either eight or 16 bits long. The shifter is eight bits long for OCMDR and OSR accesses, and 16 bits long for all other register accesses. Meaning, if the OnCE module is expecting a command to be entered, to be loaded into the OCMDR, an 8-bit shifter is selected. If the OnCE command is loaded into the OCMDR has a 16-bit, read/write associated with it, a 16-bit shifter is connected between TDI and TDO during Shift-DR. The OnCE shifter selection can be understood in terms of the state diagram shown in **Figure 17-28.**

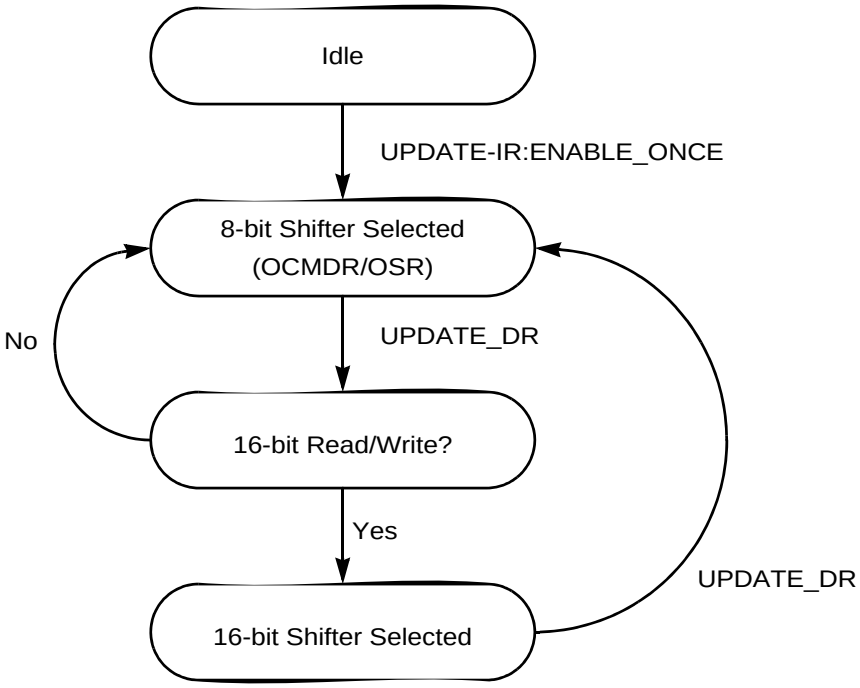


Figure 17-28. OnCE Shifter Selection State Diagram

As long as ENABLE_ONCE is decoded in JTAGIR, one of the two shifters is available for shifting. If a different JTAG instruction is shifted in, the BYPASS register is selected.

17.13.4.2 Executing a OnCE Command by Reading the OCR

The following sequence shows how to read the OCR, assuming ENABLE_ONCE is being decoded in JTAGIR, the JTAG state machine is at run-test/idle, and the DR path has not yet been entered, meaning the OnCE module has selected the 8-bit shifter. Please refer to **Figure 17-29.**

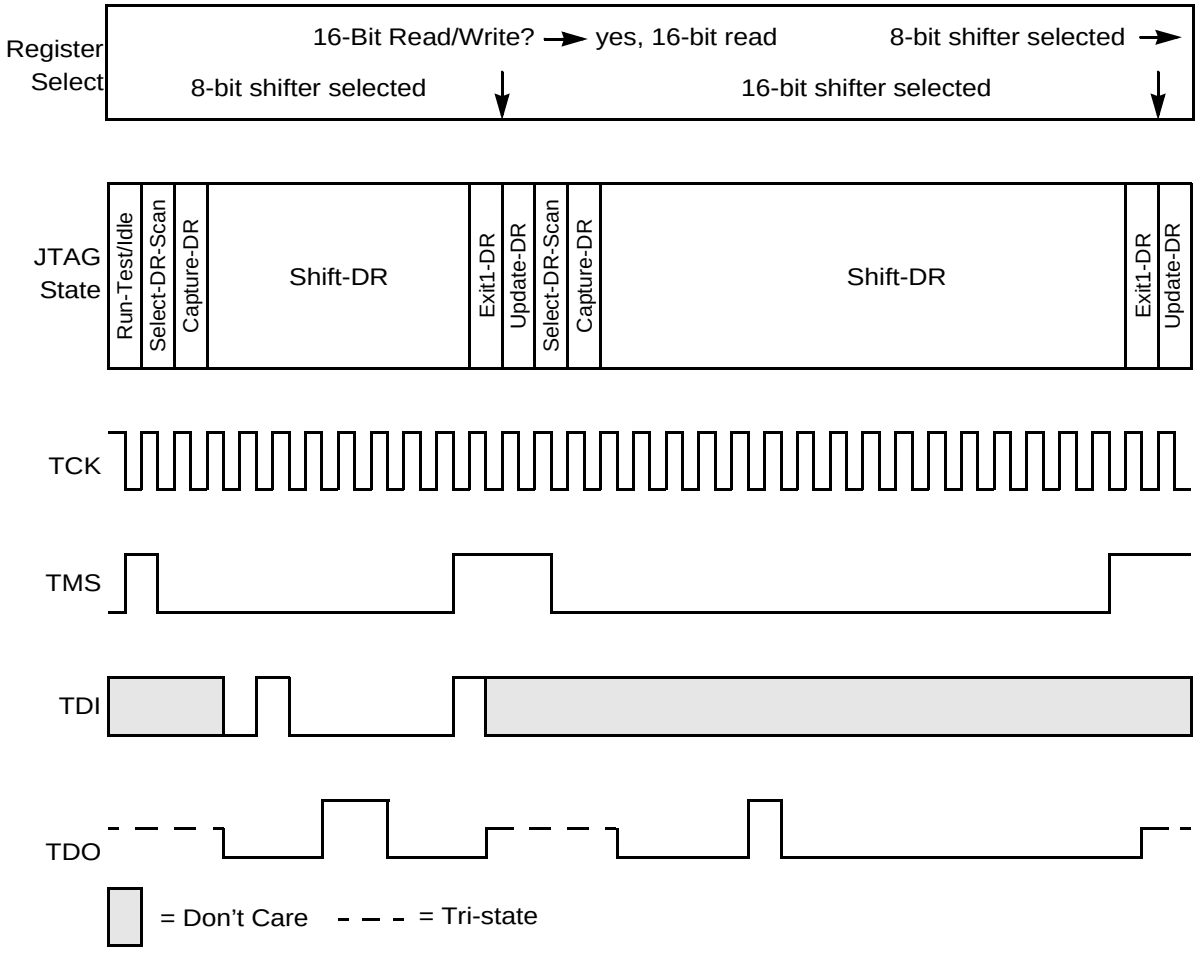


Figure 17-29. Executing a OnCE Command by Reading the OCR

In the first shift sequence, the 8-bit shifter is selected. Whenever the 8-bit shifter is selected, the OCMDR is written, on Update-DR, with the value shifted into TDI. In this case, $82 is shifted in. This is the OnCE opcode for read OCR. Similarly, whenever the 8-bit shifter is selected, it captures the value of the OSR when passing through Capture-DR. This value is then shifted out of TDO in the ensuing shift. In this case, $18 was shifted out, indicating the controller is in Debug mode.

When Update-DR is passed through in an OCMDR write, the OnCE module begins decoding the opcode in the OCMDR. During command decoding, the OnCE module determines whether a 16-bit shift is to occur (in this case, yes) and if so, whether it is a read or write. If it is a read, the

register selected by the RS field in the OCMDR is captured in the 16-bit shifter on Capture-DR. If it is a legal write, the selected register is written on Update-DR following the 16-bit shift.

17.13.4.3 Executing a OnCE Command by Writing the OCNTR

The 8-bit OCNTR is written in this sequence. First the write OCR opcode is entered, followed by a 16-bit shift sequence even though the OCNTR is only eight bits long. If a selected register is less than 16 bits, it always reads to or writes from the LSB of the 16-bit shifter. The initial set-up is identical to the previous example. Please see **Figure 17-30.**

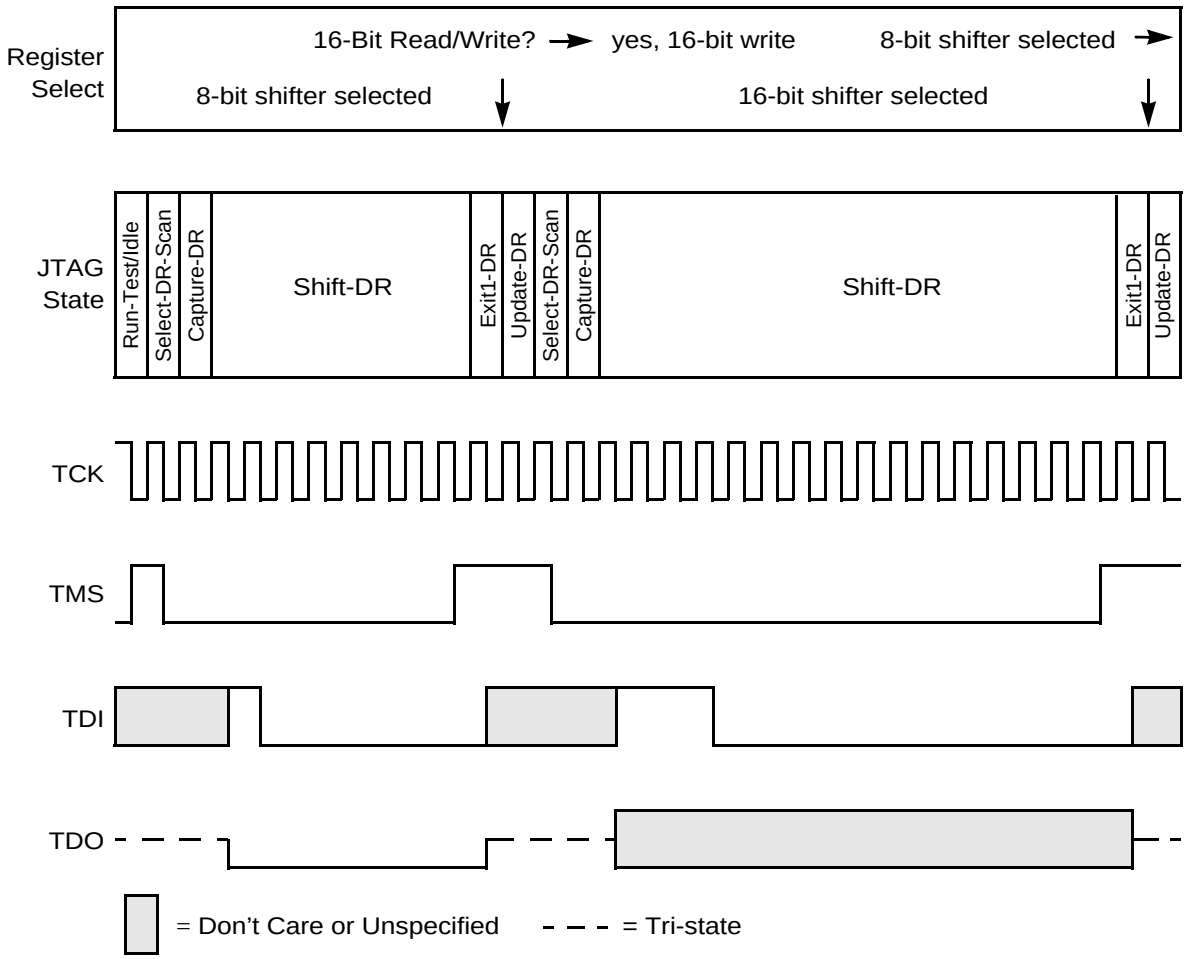


Figure 17-30. Executing a OnCE Command by Writing the OCNTR

In this sequence, $00 was read out from the OSR, indicating the chip is in Normal mode, that is: the controller core is running. The OnCE opcode shifted in is $01, corresponding to write OCNTR. In the ensuing 16-bit shift, $0003 is shifted in. Since the OCNTR is only eight bits wide, it loads the eight LSB, or $03. The bits coming out of TDO are unspecified during 16-bit writes.

17.13.4.4 OSR Status Polling

As described in the previous examples, status information from the OSR is made available each time a new OnCE command is shifted in. This provides a convenient means for status polling. The following sequence shows the OSR status polling. Assume a breakpoint has been set up to halt the core See **Figure 17-31.**

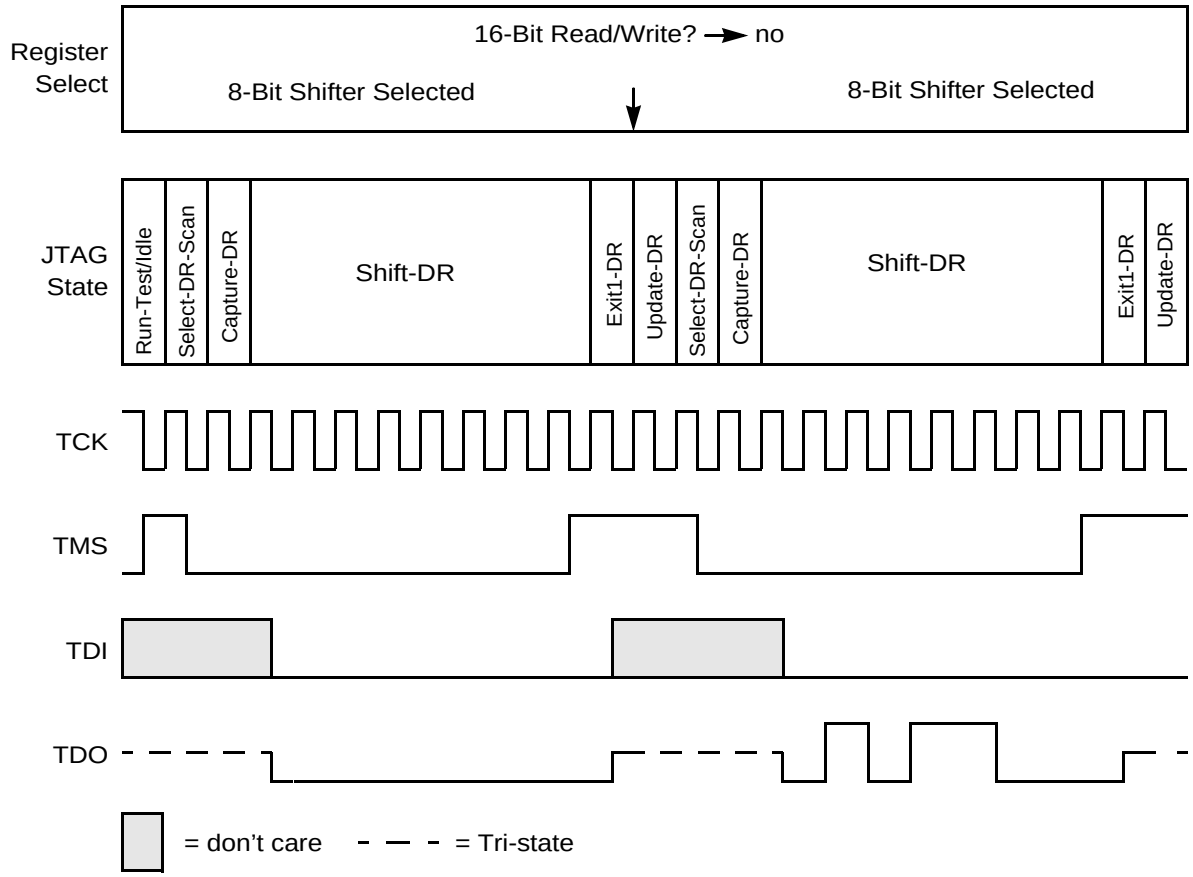


Figure 17-31. OSR Status Polling

On the first 8-bit sequence, $00 is shifted into the OCMDR, corresponding to no-register selected. Next, $00 is read out from the OSR. This read-out indicates the controller core is still in Normal mode. The OnCE module decodes the $00 opcode and again selects the 8-bit shifter since no 16-bit access is associated with this opcode. On the second 8-bit sequence, $1A is read from the OSR. This read indicates the chip is in Debug mode and a hardware breakpoint has occurred.

17.13.4.5 JTAGIR Status Polling

OSR status polling has some disadvantages. First, the *OSR is not accessible* when the controller is executing a stop instruction. OSR access, and all other OnCE register accesses, can continue only after the Stop state has been exited, either by an interrupt or a by DEBUG_REQUEST. Secondly, 8-bit shifts are required. Polling the JTAGIR provides a more efficient and more reliable means of gathering status information. The following sequence shows how the JTAGIR can be polled. Again, assume a breakpoint has been created to halt the core. Please refer to **Figure 17-32.**

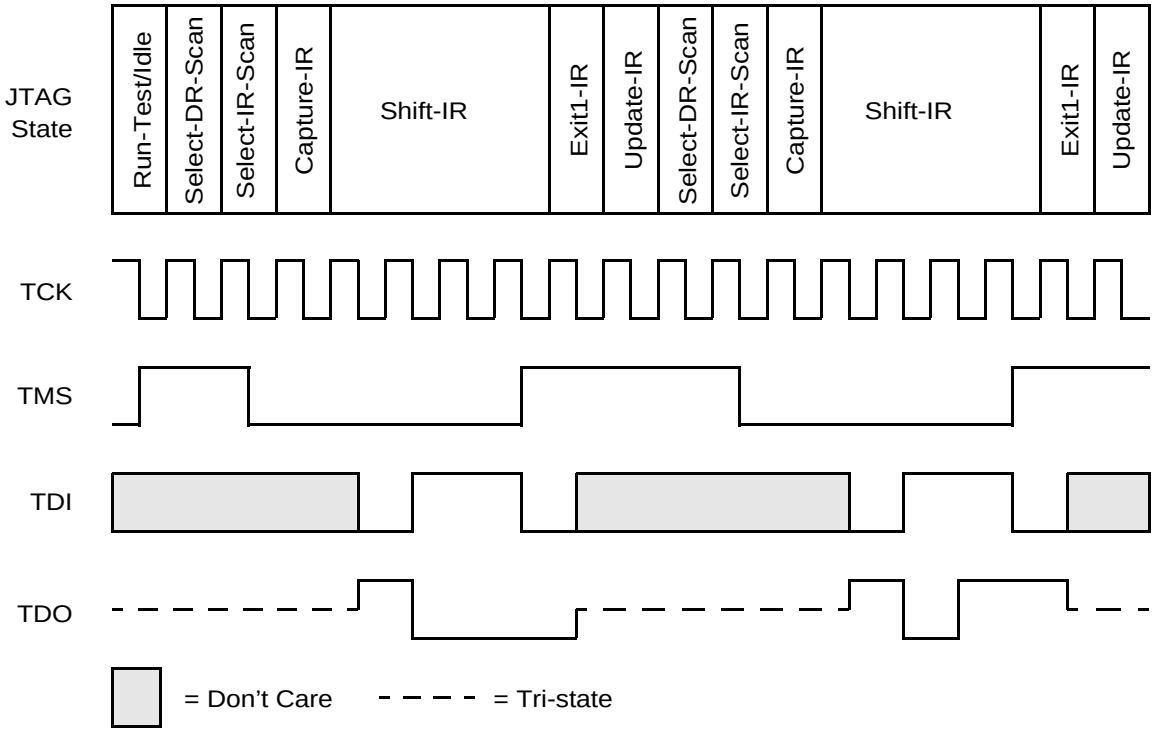


Figure 17-32. JTAGIR Status Polling

On the first 4-bit shift sequence, $6, ENABLE_ONCE, is shifted into JTAGIR. The first two bits coming out of TDO are the standard constants. The last two are output shifter (OS) bits. OS is 00, indicating the controller is in Normal mode. On the second 4-bit shift sequence, ENABLE_ONCE is again shifted in. The OS bits are now 11, indicating the chip is in Debug mode. ENABLE_ONCE does not have to be shifted in for JTAGIR polling. After reading proper status, the DR path can be entered directly for OnCE register accesses.

17.13.4.6 $\overline{\text{DE}}$ Pin Polling

The $\overline{\text{DE}}$ pin is also used to provide information about the processor state. $\overline{\text{DE}}$ goes low upon entry into Debug mode. The pin is not released until Debug mode is exited.

17.13.5 OnCE Module Low Power Operation

If OnCE module's debug capability is not required by an application, it is possible to shut off the breakpoint units and the OnCE module for low power consumption. This is done by setting the PWD bit in the OCR register. This prevents the breakpoint units from latching any values for comparison.

17.13.6 Resetting the Chip Without Resetting the OnCE Unit

Digital signal controller can be reset without resetting the OnCE module. This reset allows creating breakpoints on an application's final target hardware with a potential power-on reset circuit. The OnCE module reset is disabled using the following technique: If an ENABLE_ONCE instruction is in the JTAGIR when a hardware or COP timer reset occurs, then the OnCE module unit is *not* reset. All OnCE module registers retain their current values while all valid breakpoints remain enabled. If any other instruction is in the JTAGIR when a chip reset occurs, the OnCE module is reset. The preceding capabilities permits the following sequence, useful for setting breakpoints on final target hardware:

1. Reset the controller chip using the $\overline{\text{RESET}}$ pin.
2. Come out of reset and begin to execute the application code, perhaps out of program ROM if this is where the user's application code is located.
3. Halt the controller and enter the Debug mode.
4. Program the desired breakpoint(s) and leave the ENABLE_ONCE instruction in the JTAGIR.
5. Reset the controller using the $\overline{\text{RESET}}$ pin again, but this time the OnCE module is *not* reset and the breakpoints remain valid and enabled.
6. Come out of reset and begin to execute the application code, perhaps out of program ROM if this is where the user's application code is located.
7. The controller then correctly triggers in the application code when the desired breakpoint condition is detected.
8. The JTAG port can be polled to detect the occurrence of this breakpoint.

Another technique for loading breakpoints is achieved as follows:

1. Reset the controller by asserting both the $\overline{\text{RESET}}$ pin and the $\overline{\text{TRST}}$ pin.
2. Release the $\overline{\text{TRST}}$ pin.
3. Shift in a ENABLE_ONCE instruction into the JTAGIR.
4. Set up the desired breakpoints.
5. Release the $\overline{\text{RESET}}$ pin.

6. Come out of reset and begin to execute the application code.
7. The controller then correctly triggers in the application code when the desired breakpoint condition is detected.
8. The JTAG port can be polled to detect the occurrence of this breakpoint.

Another useful sequence is to enter the Debug Mode directly from reset. This sequence is approached as follows:

1. Reset the controller by asserting both the $\overline{\text{RESET}}$ pin and the $\overline{\text{TRST}}$ pin.
2. Release the $\overline{\text{TRST}}$ pin.
3. Shift in a DEBUG_REQUEST instruction into the JTAGIR.
4. Release the $\overline{\text{RESET}}$ pin.
5. Come out of reset and directly enter Debug mode.

The OnCE module reset can still be forced on controller reset even if there is an ENABLE_ONCE instruction in the JTAGIR. This is accomplished by asserting the $\overline{\text{TRST}}$ pin in addition to the $\overline{\text{RESET}}$ pin, guaranteeing reset of the OnCE module.

Table 17-12. Document Revision History for Chapter 17

Version History	Description of Change
Rev. 8	Formatting, layout, spelling, and grammar corrections. Added revision history table.

Chapter 18
JTAG Port

18.1 Introduction

This chapter describes the 56F800 core-based family devices providing board and chip-level debugging, and high-density circuit board testing specific to Joint Test Action Group (JTAG).

The 56F80x family of digital signal controllers provides board and chip-level testing capability through two on-chip modules, both accessed through the JTAG port/OnCE module interface:

- On-Chip Emulation (OnCE) module
- Test Access Port (TAP) and 16-state controller, also known as the JTAG port

Presence of the JTAG port/OnCE module interface permits insertion of the controller into a target system while retaining debug control. This capability is especially important for devices without an external bus because it eliminates the need for a costly cable to bring out the footprint of the chip required by a traditional emulator system.

The OnCE module is a module used in controller to debug application software employed with the chip. The port is a separate on-chip block allowing non-intrusive controller interaction with accessibility through the pins of the JTAG interface. The OnCE module makes it possible to examine registers, memory, or on-chip peripherals' contents in a special debug environment. This avoids sacrificing any user-accessible on-chip resources to perform debugging procedures. Please see Chapter 17, "OnCE Module" for details about the OnCE module implementation of the 56F8x series.

The JTAG port is a dedicated user-accessible TAP compatible with the *IEEE 1149.1a-1993 Standard Test Access Port and Boundary Scan Architecture*. Problems associated with testing high-density circuit boards have led to the development of this proposed standard under the sponsorship of the Test Technology Committee of IEEE and the JTAG. Digital signal controllers, 56F80x devices, support circuit board test strategies based on this standard.

Five dedicated pins interface to the TAP containing a 16-state controller. The TAP uses a boundary scan technique to test the interconnections between integrated circuits after they are assembled onto a Printed Circuit Board (PCB). Boundary scans allow a tester to observe and control signal levels at each component pin through a shift register placed next to each pin. This is important for testing continuity and determining if pins are stuck at a one or zero level.

18.2 Features

Features of the TAP port include:

- Perform boundary scan operations to test circuit board electrical continuity
- Bypass the controller for a given circuit board test by replacing the Boundary Scan Register (BSR) with a single-bit register
- Sample the controller system pins during operation and transparently shift out the result in the CSR; pre-load values to output pins prior to invoking the EXTEST instruction
- Disable the output drive to pins during circuit board testing
- Provide a means of accessing the OnCE module controller and circuits to control a target system
- Query identification information, manufacturer, part number, and version from a controller
- Force test data onto the outputs of a controller IC while replacing its BSR in the serial data path with a single bit register
- Enable a weak pull-up current device on all input signals of a controller IC, helping to assure deterministic test results in the presence of continuity fault during interconnect testing

This chapter includes aspects of the JTAG implementation specific to the 56F80x device. Chapter data is intended to be utilized with IEEE 1149.1a. The discussion includes those items required by the standard to be defined and, in certain cases, provide additional information specific to the 56F80x device. For internal details and applications of the standard, refer to IEEE 1149.1a.

18.3 Block Diagram

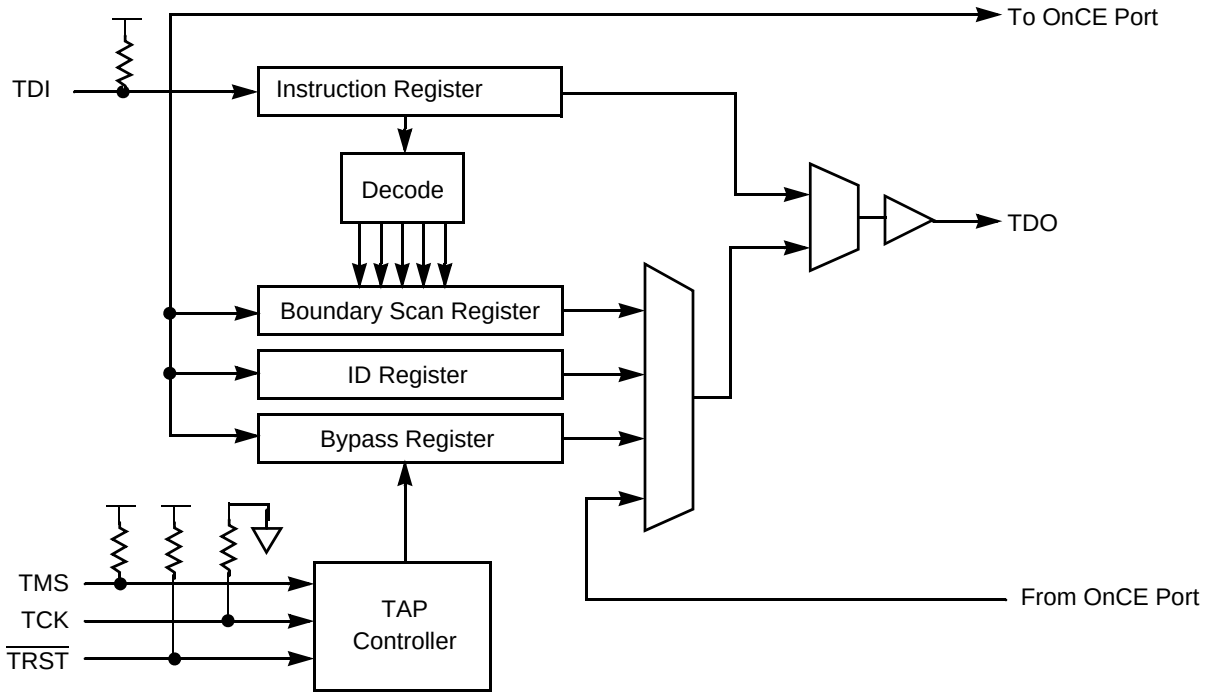


Figure 18-1. JTAG Block Diagram

A block diagram of the JTAG port is shown in **Table 18-1.** The JTAG port has four read/write registers:

1. JTAG Instruction Register (JTAGIR)
2. Boundary Scan Register (BSR)
3. Chip Identification Register (CID)
4. JTAG Bypass Register (JTAGBR)

Access to the OnCE registers is described in **Chapter 17, "OnCE Module".**

The TAP controller provides access to the JTAG instruction register (JTAGIR) through the JTAG port. The other JTAG registers must be individually selected by the JTAGIR.

18.4 Register Summary

The DSP56800 family has these registers:

- JTAG Instruction Register (JTAGIR)
- Chip Identification register (CID)
- Boundary Scan Register (BSR)
- JTAG Bypass Register (JTAGBR)

18.5 Pin Descriptions

As described in IEEE 1149.1a, the JTAG port requires a minimum of four pins to support TDI, TDO, TCK, and TMS signals. The DSP56800 family also uses the optional $\overline{\text{TRST}}$ input signal and $\overline{\text{DE}}$ output signal (not available on the 802 device) used by the OnCE module interface. The pin functions are described in **Table 18-1.**

Table 18-1. JTAG Pin Descriptions

Pin Name	Pin Description
TDI	**Test Data Input**—This input pin provides a serial input data stream to the JTAG and the OnCE modules. It is sampled on the rising edge of TCK and has an on-chip pull-up resistor.
TDO	**Test Data Output**—This tri-state output pin provides a serial output data stream from the JTAG and the OnCE modules. It is driven in the Shift-IR and Shift-DR controller states of the JTAG state machine and changes on the falling edge of TCK.
TCK	**Test Clock Input**—This input pin provides a gated clock to synchronize the test logic and shift serial data to and from the JTAG/OnCE port. If the OnCE module is not being accessed, the maximum TCK frequency is as specified in the corresponding Technical Data Sheet (DSP56F801, DSP56F802, DSP56F803, DSP56F805, or DSP56F807). When accessing the OnCE module through the JTAG TAP, the maximum frequency for TCK is 1/8 the maximum frequency specified for the 56800 core (i.e., 5MHz for TCK if the maximum CLK input is 40MHz). The TCK pin has an on-chip pull-down resistor.
TMS	**Test Mode Select Input**—This input pin is used to sequence the JTAG TAP controller's state machine. It is sampled on the rising edge of TCK and has an on-chip pull-up resistor.
$\overline{\text{TRST}}$	**Test Reset**—This input provides a reset signal to the JTAG TAP controller. The $\overline{\text{TRST}}$ pin has an on-chip pull-up resistor.
$\overline{\text{DE}}$	**Debug Event**—This output signal debugs events detected on a trigger condition. (Not available on the 802 device.)

18.6 JTAG Port Architecture

The TAP controller is a simple state machine used to sequence the JTAG port through its valid operations:

- Serially shift in or out a JTAG port command

- Update (and decode) the JTAG port Instruction Register (IR)

  Serially input or output a data value
- Update a JTAG Port (or OnCE module) Register

Note: The JTAG port oversees the shifting of data into and out of the OnCE module through the TDI and TDO pins respectively. In this case, the shifting is guided by the same TAP controller used when shifting JTAG information.

18.6.1 JTAG Instruction Register (JTAGIR) and Decoder

The TAP controller contains a 4-bit instruction register. The instruction is presented to an instruction decoder during the update-instruction register state. See **Section 18.7, "TAP Controller"** for a description of the TAP controller operating states. The instruction decoder interprets and executes the instructions according to the conditions defined by the TAP controller state machine. The 56F80x includes the three mandatory public instructions, BYPASS, SAMPLE/PRELOAD, and EXTEST, and six public instructions, CLAMP, HIGHZ, EXTEST_PULLUP, IDCODE, ENABLE_ONCE, and DEBUG_REQUEST.

The four bits B[3:0] of the IR, decode the nine instructions, illustrated in **Table 18-2.** All other encodings are reserved.

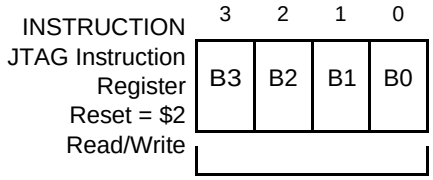


Figure 18-2. JTAGIR Register

CAUTION

Reserved JTAG instruction encodings should not be used. Hazardous operation of the chip could occur if these instructions are used.

Table 18-2. JTAGIR Encodings

B[3:0]	Instruction
0000	EXTEST
0001	SAMPLE/PRELOAD
0010	IDCODE
0011	EXTEST_PULLUP
0100	HIGHZ
0101	CLAMP
0110	ENABLE_ONCE
0111	DEBUG_REQUEST
1111	BYPASS

The JTAGIR is reset to 0010 in the Test-Logic-Reset controller state. Therefore, the IDCODE instruction is selected on JTAG reset. In the capture-IR state, the two Least Significant Bits (LSBs) of the Instruction Shift register are preset to 01, where the one is in the LSB location as required by the standard. The two Most Significant Bits (MSBs) may either capture status or be set to 0. New instructions are moved into the Instruction Shift register stage in shift-IR state.

18.6.1.1 EXTEST (B[3:0] = 0000)

The External Test (EXTEST) instruction enables the BSR between TDI and TDO, including cells for all digital device signals and associated control signals. The EXTAL pin, and any codec pins associated with analog signals, are not included in the BSR path.

In EXTEST, the BSR is capable of scanning user-defined values onto output pins, capturing values presented to input signals, and controlling the direction and value of bidirectional pins. EXTEST instruction asserts internal system reset for the controller system logic during its run in order to force a predictable internal state while performing external boundary scan operations.

18.6.1.2 SAMPLE/PRELOAD (B[3:0] = 0001)

The SAMPLE/PRELOAD instruction enables the BSR between TDI and TDO. When this instruction is selected, the test logic operation has no effect on the operation of the on-chip system logic. Nor does it have an effect on the flow of a signal between the system pin and the on-chip system logic, as specified by IEEE 1149.1-1993a. This instruction provides two separate functions. First, it provides a means to obtain a snapshot of system data and control signals (SAMPLE). The snapshot occurs on the rising edge of TCK in the capture-DR controller state. The data can be observed by shifting it transparently through the BSR.

In a normal system configuration, many signals require external pull-ups assuring proper system operation. Consequently, the same is true for the SAMPLE/PRELOAD functionality. Data latched into the BSR during the capture-DR controller state may not match the drive state of the package signal if the system-requiring pull-ups are not present within the test environment.

The second function of the SAMPLE/PRELOAD instruction is to initialize the BSR output cells (PRELOAD) prior to selection of the CLAMP, EXTEST, or EXTEST_PULLUP instruction. This initialization ensures known data appears on the outputs when executing EXTEST. The data held in the shift register stage is transferred to the output latch on the falling edge of TCK in the update-DR controller state. Data is not presented to the pins until the CLAMP, EXTEST, or EXTEST_PULLUP instruction is executed.

Note: Since there is no internal synchronization between the JTAG Clock (TCK) and the System Clock (CLK), some form of external synchronization to achieve meaningful results when sampling system values using the SAMPLE/PRELOAD instruction must be provided.

18.6.1.3 IDCODE (B[3:0] = 0010)

The IDCODE instruction enables the IDREGISTER between TDI and TDO. It is provided as a public instruction to allow the manufacturer, part number, and version of a component to be determined through the TAP.

When the IDCODE instruction is decoded, it selects the IDREGISTER, a 32-bit test data register. IDREGISTER loads a constant Logic 1into its LSB. Since the Bypass register loads a Logic 0 at the start of a scan cycle, examination of the first bit of data shifted out of a component during a test data scan sequence immediately following exit from the Test-Logic-Reset controller state shows whether an IDREGISTER is included in the design.

When the IDCODE instruction is selected, the operation of the test logic has no effect on the operation of the on-chip system logic, as required in IEEE 1149.1a-1993.

18.6.1.4 EXTEST_PULLUP (B[3:0] = 0011)

The EXTEST_PULLUP instruction is provided as a public instruction to aid in fault diagnoses during boundary scan testing of a circuit board. This instruction functions similarly to EXTEST, with the only difference being the presence of a weak pull-up device on all input signals. Given an appropriate charging delay, the pull-up current supplies a deterministic Logic 1 result on an open input. When this instruction is used in board-level testing with heavily loaded nodes, it may require a charging delay greater than the two TCK periods needed to transition from the update-DR state to the capture-DR state. Two methods of providing an increase delay are available: traverse into the run-test/idle state for extra TCK periods of charging delay, or limit the maximum TCK frequency, slow down the TCK, so two TCK periods are adequate. The EXTEST_PULLUP instruction asserts internal system reset for the controller system logic for the duration of EXTEST_PULLUP in order to force a predictable internal state while performing external boundary scan operations.

18.6.1.5 HIGHZ (B[3:0] = 0100)

The HIGHZ instruction enables the single-bit bypass register between TDI and TDO. It is provided as a public instruction in order to prevent having to drive the output signals back during circuit board testing. When the HIGHZ instruction is invoked, all output drivers are placed in an inactive-drive state. HIGHZ asserts internal system reset for the controller system logic for the duration of HIGHZ in order to force a predictable internal state while performing external boundary scan operations.

18.6.1.6 CLAMP (B[3:0] = 0101)

The CLAMP instruction enables the single-bit bypass register between TDI and TDO. It is provided as a public instruction. When the CLAMP instruction is invoked, the package output signals respond to the preconditioned values within the update latches of the BSR, even though the bypass register is enabled as the test data register. In-circuit testing, it can be facilitated by setting up guarding signal conditions controlling the operation of logic not involved in the test, but with use of the SAMPLE/PRELOAD or EXTEST instructions. When the CLAMP instruction is executed, the state and drive of all signals remain static until a new instruction is invoked. A feature of the CLAMP instruction is while the signals continue to supply the guarding inputs to the in-circuit test site, the bypass mode is enabled, minimizing overall test time. Data in the boundary scan cell remains unchanged until a new instruction is shifted in or the JTAG state machine is set to its reset state. CLAMP asserts internal system reset for the controller system logic for the duration of CLAMP in order to force a predictable internal state while performing external boundary scan operations.

18.6.1.7 ENABLE_ONCE (B[3:0] = 0110)

The ENABLE_ONCE instruction enables the JTAG port to communicate with the OnCE state machine and registers. It is provided as a public instruction to allow the user to perform system debug functions. When the ENABLE_ONCE instruction is invoked, the TDI and TDO pins are connected directly to the OnCE registers. The particular OnCE register connected between TDI and TDO is selected by the OnCE state machine and the OnCE instruction being executed. All communication with the OnCE instruction controller is done through the select-DR-scan path of the JTAG state machine. Refer to the *DSP56800 Family Manual* (DSP56800FM) for more information.

18.6.1.8 DEBUG_REQUEST (B[3:0] = 0111)

The DEBUG_REQUEST instruction asserts a request to halt the core for entry to debug mode. It is typically used in conjunction with ENABLE_ONCE to perform system debug functions. It is provided as a public instruction. When the DEBUG_REQUEST instruction is invoked, the TDI and TDO pins are connected to the bypass register. Refer to the *DSP56800 Family Manual (DSP56800FM)* for more information.

18.6.1.9 BYPASS (B[3:0] = 1111)

The BYPASS instruction enables the single-bit bypass register between TDI and TDO, illustrated in **Figure 18-3.** This creates a shift register path from TDI to the Bypass register and finally to TDO, circumventing the BSR. This instruction is used to enhance test efficiency by shortening the overall path between TDI and TDO when no test operation of a component is required. In this instruction, the controller system logic is independent of the TAP. When this instruction is selected, the test logic has no effect on the operation of the on-chip system logic, as required in IEEE 1149.1-1993a.

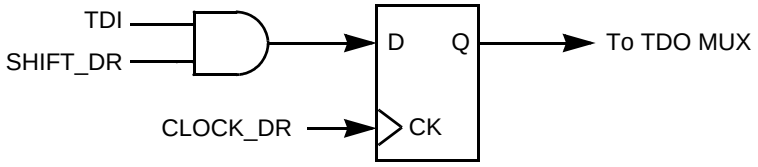


Figure 18-3. Bypass Register

18.6.2 JTAG Chip Identification (CID) Register

The Chip Identification (CID) register is a 32-bit register providing a unique JTAG ID for the 56F80x family. It is offered as a public instruction allowing the manufacturer, part number, and version of a component to be determined through the TAP. **Figure 18-5** illustrates the CID register configuration.

IR = $2	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	PNUM 3	PNUM 2	PNUM 1	PNUM 0	MFG ID 11	MFG ID 10	MFG ID 9	MFG ID 8	MFG ID 7	MFG ID 6	MFG ID 5	MFG ID 4	MFG ID 3	MFG ID 2	MFG ID 1	MFG ID 0
Write																
Reset	0	1	0	1	0	0	0	0	0	0	0	1	1	1	0	1

IR = $2	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Read	VER 3	VER 2	VER 1	VER 0	PNUM 15	PNUM 14	PNUM 13	PNUM 12	PNUM 11	PNUM 10	PNUM 9	PNUM 8	PNUM 7	PNUM 6	PNUM 5	PNUM 4
Write																
Reset	0	0	0	0	0	0	0	1	1	1	1	1	0	0	1	0

Figure 18-4. JTAG Chip Identification Register (CID)

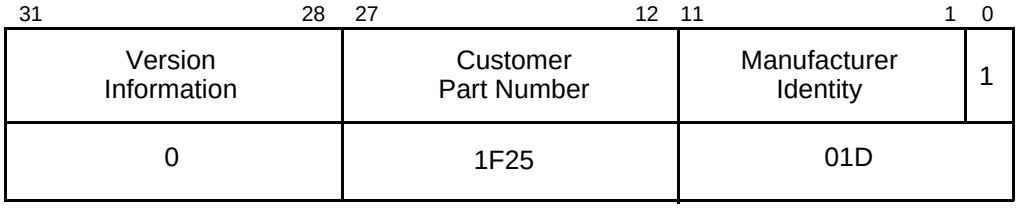


Figure 18-5. Chip Identification Register Configuration

For the initial release of the 56F805, the device identification number is $01F2501D. The version code is $0. Future revisions increment this version code. The value of the customer part number code is $1F23. **Table 18-3** provides the JTAG ID for the various versions of the chips in this series.

Table 18-3. JTAG ID Codes

JTAG ID code is expressed in hex form, and is calculated as (Version, Design_Center, Part_Number, Manufacturer_ID, 1'b1)

Part Number	Version	Design Center	JTAG ID Code Part Number	Manufacturer ID	Constant	JTAG ID Code
56F801	0	7	801	14	1	01F2101D
56F802	0	7	801	14	1	01F21010
56F803	0	7	805	14	1	01F2501D
56F803	1	7	805	14	1	1F2501D
56F805	0	7	805	14	1	01F2501D
56F805	1	7	805	14	1	11F2501D
56F807	0	7	807	14	1	01F2701D

Freescales's manufacturer identity is: 00000001110. The customer part number consists of two parts: design center number, bits 27–22, and design center assigned sequence number, bits 21–12. The controller standard products design center number is 000111. Version information

and design center assigned sequence number values vary depending on the current revision and implementation of a specific chip. The bit assignment for the identification code is given in **Table 18-4.**

Table 18-4. Device ID Register Bit Assignment

Bit No.	Code Use	Value for 56F80x
31–28	Version Number	0000 (For initial version only—these bits may vary)
27–22	Freescale Design Center ID	00 0111
21–12	Family and part ID	11 0010 0101
11-1	Freescale Manufacturer ID	000 0000 1110
0	IEEE Requirement	Always 1

18.6.3 JTAG Boundary Scan Register (BSR)

The Boundary Scan Register (BSR) is used to examine or control the scannable pins on the 56F80x family of controllers. The BSR for these devices contains 338 bits. Each scannable pin has at least one BSR bit associated with it. **Table 18-5** illustrates the contents of the BSR for the 56F80x family.

IR = $0, $1, $3	337	336	335	334	333	Bits 332 through 5	4	3	2	1	0
Read-Only											

Figure 18-6. Boundary Scan Register (BSR)

Table 18-5. BSR Contents for 56F80x

Bit #	Pin/Bit Name	Pin Type	BSR Cell Type	56F801 Pin # (48 LQFP Pkg.)	56F802 Pin # (48 LQFP Pkg.)	56F803 Pin # (100 LQFP Pkg.)	56F805 Pin # (144 LQFP Pkg.)	56F807 Pin # (144 LQFP Pkg.)	56F807 Pin # (160 MAPBGA Pkg.)
0	D9—Input	Input/Output	BC_1	–	–	100	144	32	K2
1	D9—Output	Input/Output	BC_1	–	–	100	144	32	K2
2	D9—Output Enable	Control	BC_1	–	–	–	–	–	–
3	D9—Pull-up Enable	Control	BC_1	–	–	–	–	–	–
4	D8—Input	Input/Output	BC_1	–	–	99	143	31	L1
5	D8—Output	Input/Output	BC_1	–	–	99	143	31	L1

Table 18-5. BSR Contents for 56F80x (Continued)

Bit #	Pin/Bit Name	Pin Type	BSR Cell Type	56F801 Pin # (48 LQFP Pkg.)	56F802 Pin # (48 LQFP Pkg.)	56F803 Pin # (100 LQFP Pkg.)	56F805 Pin # (144 LQFP Pkg.)	56F807 Pin # (144 LQFP Pkg.)	56F807 Pin # (160 MAPBGA Pkg.)
6	D8— Output Enable	Control	BC_1	–	–	–	–	–	–
7	D8— Pull-up Enable	Control	BC_1	–	–	–	–	–	–
8	D7— Input	Input/ Output	BC_1	–	–	98	142	30	K1
9	D7— Output	Input/ Output	BC_1	–	–	98	142	30	K1
10	D7— Output Enable	Control	BC_1	–	–	–	–	–	–
11	D7— Pull-up Enable	Control	BC_1	–	–	–	–	–	–
12	D6— Input	Input/ Output	BC_1	–	–	97	141	29	K3
13	D6— Output	Input/ Output	BC_1	–	–	97	141	29	K3
14	D6—Output Enable	Control	BC_1	–	–	–	–	–	–
15	D6— Pull-up Enable	Control	BC_1	–	–	–	–	–	–
16	D5— Input	Input/ Output	BC_1	–	–	96	140	28	J2
17	D5— Output	Input/ Output	BC_1	–	–	96	140	28	J2
18	D5— Output Enable	Control	BC_1	–	–	–	–	–	–
19	D5— Pull-up Enable	Control	BC_1	–	–	–	–	–	–
20	D4— Input	Input/ Output	BC_1	–	–	95	139	27	J1
21	D4— Output	Input/ Output	BC_1	–	–	95	139	27	J1
22	D4— Output Enable	Control	BC_1	–	–	–	–	–	–
23	D4— Pull-up Enable	Control	BC_1	–	–	–	–	–	–
24	HOME— Input	Input/ Output	BC_1	–	–	–	138	155	C4
25	HOME— Output	Input/ Output	BC_1	–	–	–	138	155	C4
26	HOME1— Output Enable	Control	BC_1	–	–	–	–	–	–
27	HOME1— Pull-up Enable	Control	BC_1	–	–	–	–	–	–

Table 18-5. BSR Contents for 56F80x (Continued)

Bit #	Pin/Bit Name	Pin Type	BSR Cell Type	56F801 Pin # (48 LQFP Pkg.)	56F802 Pin # (48 LQFP Pkg.)	56F803 Pin # (100 LQFP Pkg.)	56F805 Pin # (144 LQFP Pkg.)	56F807 Pin # (144 LQFP Pkg.)	56F807 Pin # (160 MAPBGA Pkg.)
28	D3—Input	Input/Output	BC_1	–	–	94	137	26	J3
29	D3—Output	Input/Output	BC_1	–	–	94	137	26	J3
30	D3—Output Enable	Control	BC_1	–	–	–	–	–	–
31	D3—Pull-up Enable	Control	BC_1	–	–	–	–	–	–
32	PHASEA1—Input	Input/Output	BC_1	–	–	–	136	151	E4
33	PHASEA1—Output	Input/Output	BC_1	–	–	–	136	151	E4
34	PHASEA1—Output Enable	Control	BC_1	–	–	–	–	–	–
35	PHASEA1—Pull-up Enable	Control	BC_1	–	–	–	–	–	–
36	PHASEB1—Input	Input/Output	BC_1	–	–	–	134	152	B5
37	PHASEB1—Output	Input/Output	BC_1	–	–	–	134	152	B5
38	PHASEB1—Output Enable	Control	BC_1	–	–	–	–	–	–
39	PHASEB1—Pull-up Enable	Control	BC_1	–	–	–	–	–	–
40	INDEX1—Input	Input/Output	BC_1	–	–	–	132	149	D4
41	INDX1—Output	Input/Output	BC_1	–	–	–	132	149	D4
42	INDX1—Output Enable	Control	BC_1	–	–	–	–	–	–
43	INDX1—Pull-up Enable	Control	BC_1	–	–	–	–	–	–
44	D2—Input	Input/Output	BC_1	–	–	91	131	25	H2
45	D2—Output	Input/Output	BC_1	–	–	91	131	25	H2
46	D2—Output Enable	Control	BC_1	–	–	–	–	–	–
47	D2—Pull-up Enable	Control	BC_1	–	–	–	–	–	–
48	D1—Input	Input/Output	BC_1	–	–	90	130	24	H1
49	D1—Output	Input/Output	BC_1	–	–	90	130	24	H1

Table 18-5. BSR Contents for 56F80x (Continued)

Bit #	Pin/Bit Name	Pin Type	BSR Cell Type	56F801 Pin # (48 LQFP Pkg.)	56F802 Pin # (48 LQFP Pkg.)	56F803 Pin # (100 LQFP Pkg.)	56F805 Pin # (144 LQFP Pkg.)	56F807 Pin # (144 LQFP Pkg.)	56F807 Pin # (160 MAPBGA Pkg.)
50	D1— Output Enable	Control	BC_1	–	–	–	–	–	–
51	D1— Pull-up Enable	Control	BC_1	–	–	–	–	–	–
52	D0— Input	Input/ Output	BC_1	–	–	89	128	23	H3
53	D0— Output	Input/ Output	BC_1	–	–	89	128	23	H3
54	D0— Output Enable	Control	BC_1	–	–	–	–	–	–
55	D0— Pull-up Enable	Control	BC_1	–	–	–	–	–	–
56	MPIOD5— Input	Input/ Output	BC_1	–	–	–	127	54	K6
57	MPIOD5— Output	Input/ Output	BC_1	–	–	–	127	54	K6
58	MPIOD5— Output Enable	Control	BC_1	–	–	–	–	–	–
59	MPIOD5— Pull-up Enable	Control	BC_1	–	–	–	–	–	–
60	SCLK— Input	Input/ Output	BC_1	7	–	87	125	144	E5
61	SCLK— Output	Input/ Output	BC_1	7	–	87	125	144	E5
62	SCLK— Output Enable	Control	BC_1	–	–	–	–	–	–
63	SCLK— Pull-up Enable	Control	BC_1	–	–	–	–	–	–
64	MOSI— Input	Input/ Output	BC_1	6	–	86	124	146	A6
65	MOSI— Output	Input/ Output	BC_1	6	–	86	124	146	A6
66	MOSI— Output Enable	Control	BC_1	–	–	–	–	–	–
67	MOSI— Pull-up Enable	Control	BC_1	–	–	–	–	–	–
68	MPIOD4— Input	Input/ Output	BC_1	–	–	–	123	53	L6
69	MPIOD4— Output	Input/ Output	BC_1	–	–	–	123	53	L6
70	MPIOD4— Output Enable	Control	BC_1	–	–	–	–	–	–
71	MPIOD4— Pull-up Enable	Control	BC_1	–	–	–	–	–	–

Table 18-5. BSR Contents for 56F80x (Continued)

Bit #	Pin/Bit Name	Pin Type	BSR Cell Type	56F801 Pin # (48 LQFP Pkg.)	56F802 Pin # (48 LQFP Pkg.)	56F803 Pin # (100 LQFP Pkg.)	56F805 Pin # (144 LQFP Pkg.)	56F807 Pin # (144 LQFP Pkg.)	56F807 Pin # (160 MAPBGA Pkg.)
72	MISO—Input	Input/Output	BC_1	5	–	85	122	145	B7
73	MISO—Output	Input/Output	BC_1	5	–	85	122	145	B7
74	MISO—Output Enable	Control	BC_1	–	–	–	–	–	–
75	MISO—Pull-up Enable	Control	BC_1	–	–	–	–	–	–
76	MPIOD3—Input	Input /Output	BC_1	–	–	–	121	52	M6
77	MPIOD3—Output	Input/Output	BC_1	–	–	–	121	52	M6
78	MPIOD3—Output Enable	Control	BC_1	–	–	–	–	–	–
79	MPIDO3—Pull-up Enable	Control	BC_1	–	–	–	–	–	–
80	$\overline{SS}$—Input	Input/Output	BC_1	4	–	84	120	143	A7
81	$\overline{SS}$—Output	Input/Output	BC_1	4	–	84	120	143	A7
82	$\overline{SS}$—Output Enable	Control	BC_1	–	–	–	–	–	–
83	$\overline{SS}$—Pull-up Enable	Control	BC_1	–	–	–	–	–	–
84	$\overline{RSTO}$—Output	Output	BC_1	–	–	–	119	97	H12
85	$\overline{RSTO}$—Output Enable	Control	BC_1	–	–	–	–	–	–
86	TD3—Input	Input/Output	BC_1	–	–	–	118	129	B10
87	TD3—Output	Input/Output	BC_1	–	–	–	118	129	B10
88	TD3—Output Enable	Control	BC_1	–	–	–	–	–	–
89	TD3—Pull-up Enable	Control	BC_1	–	–	–	–	–	–
90	TD2—Input	Input/Output	BC_1	3	4	83	116	128	D10
91	TD2—Output	Input/Output	BC_1	3	4	83	116	128	D10
92	TD2—Output Enable	Control	BC_1	–	–	–	–	–	–

Table 18-5. BSR Contents for 56F80x (Continued)

Bit #	Pin/Bit Name	Pin Type	BSR Cell Type	56F801 Pin # (48 LQFP Pkg.)	56F802 Pin # (48 LQFP Pkg.)	56F803 Pin # (100 LQFP Pkg.)	56F805 Pin # (144 LQFP Pkg.)	56F807 Pin # (144 LQFP Pkg.)	56F807 Pin # (160 MAPBGA Pkg.)
93	TD2— Pull-up Enable	Control	BC_1	–	–	–	–	–	–
94	TD1— Input	Input/ Output	BC_1	2	3	82	114	127	A11
95	TD1— Output	Input/ Output	BC_1	2	3	82	114	127	A11
96	TD1— Output Enable	Control	BC_1	–	–	–	–	–	–
97	TD1— Pull-up Enable	Control	BC_1	–	–	–	–	–	–
98	TD0— Input	Input/ Output	BC_1	1	–	–	113	126	B11
99	TD0— Output	Input/ Output	BC_1	1	–	–	113	126	B11
100	TD0— Output Enable	Control	BC_1	–	–	–	–	–	–
101	TD0— Pull-up Enable	Control	BC_1	–	–	–	–	–	–
102	CLKO— Output	Output	BC_1	–	–	81	112	158	A2
103	CLKO— Output Enable	Control	BC_1	–	–	–	–	–	–
104	$\overline{DE}$— Output	Output	BC_1	12	–	80	111	121	B13
105	$\overline{DE}$—Output Enable	Control	BC_1	–	–	–	–	–	–
106	$\overline{RESET}$	Input	BC_1	–	16	79	110	98	H14
107	EXTBOOT	Input	BC_1	–	–	78	109	86	M14
108	RXD0— Input	Input/ Output	BC_1	11	8	77	108	160	A1
109	RXD0— Output	Input/ Output	BC_1	11	8	77	108	160	A1
110	RXD0— Output Enable	Control	BC_1	–	–	–	–	–	–
111	RXD0— Pull-up Enable	Control	BC_1	–	–	–	–	–	–
112	TXD0— Input	Input/ Output	BC_1	8	5	76	107	159	B3
113	TXD0— Output	Input/ Output	BC_1	8	5	76	107	159	B3
114	TXD0— Output Enable	Control	BC_1	–	–	–	–	–	–

Table 18-5. BSR Contents for 56F80x (Continued)

Bit #	Pin/Bit Name	Pin Type	BSR Cell Type	56F801 Pin # (48 LQFP Pkg.)	56F802 Pin # (48 LQFP Pkg.)	56F803 Pin # (100 LQFP Pkg.)	56F805 Pin # (144 LQFP Pkg.)	56F807 Pin # (144 LQFP Pkg.)	56F807 Pin # (160 MAPBGA Pkg.)
115	TXD0—Pull-up Enable	Control	BC_1	–	–	–	–	–	–
116	PWMA5—Output	Output	BC_1	48	2	75	106	81	P14
117	PWMA5—Output Enable	Control	BC_1	–	–	–	–	–	–
118	PWMA4—Output	Output	BC_1	47	1	74	105	80	M12
119	PWMA4—Output Enable	Control	BC_1	–	–	–	–	–	–
120	MPIOD2—Input	Input/Output	BC_1	–	–	–	104	51	K5
121	MPIOD2—Output	Input/Output	BC_1	–	–	–	104	51	K2
122	MPIOD2—Output Enable	Control	BC_1	–	–	–	–	–	–
123	MPIOD2—Pull-up Enable	Control	BC_1	–	–	–	–	–	–
124	PWMA3—Output	Output	BC_1	46	31	73	103	79	N13
125	PWMA3—Output Enable	Control	BC_1	–	–	–	–	–	–
126	MPIOD1—Input	Input/Output	BC_1	–	–	–	102	50	P5
127	MPIOD1—Output	Input/Output	BC_1	–	–	–	102	50	P5
128	MPIOD1—Output Enable	Control	BC_1	–	–	–	–	–	–
129	MPIOD1—Pull-up Enable	Control	BC_1	–	–	–	–	–	–
130	PWMA2—Output	Output	BC_1	45	31	72	101	78	N12
131	PWMA2—Output Enable	Control	BC_1	–	–	–	–	–	–
132	MPIOD0—Input	Input/Output	BC_1	–	–	–	100	49	N5
133	MPIOD0—Output	Input/Output	BC_1	–	–	–	100	49	N5
134	MPIOD0—Output Enable	Control	BC_1	–	–	–	–	–	–
135	MPIOD0—Pull-up Enable	Control	BC_1	–	–	–	–	–	–
136	PWMA1—Output	Output	BC_1	44	30	71	99	77	P13

Table 18-5. BSR Contents for 56F80x (Continued)

Bit #	Pin/Bit Name	Pin Type	BSR Cell Type	56F801 Pin # (48 LQFP Pkg.)	56F802 Pin # (48 LQFP Pkg.)	56F803 Pin # (100 LQFP Pkg.)	56F805 Pin # (144 LQFP Pkg.)	56F807 Pin # (144 LQFP Pkg.)	56F807 Pin # (160 MAPBGA Pkg.)
137	PWMA1—Output Enable	Control	BC_1	–	–	–	–	–	–
138	MPIOB7—Input	Input/Output	BC_1	–	–	–	98	47	M4
139	MBIOB7—Output	Input/Output	BC_1	–	–	–	98	47	M4
140	MPIOB7—Output Enable	Control	BC_1	–	–	–	–	–	–
141	MPIOB7—Pull-up Enable	Control	BC_1	–	–	–	–	–	–
142	PWMA0—Output	Output	BC_1	40	28	70	97	75	P12
143	PWMA0—Output Enable	Control	BC_1	–	–	–	–	–	–
144	MPIOB6—Input	Input/Output	BC_1	–	–	–	96	46	P4
145	MPIOB6—Output	Input/Output	BC_1	–	–	–	96	46	P4
146	MPIOB6—Output Enable	Control	BC_1	–	–	–	–	–	–
147	MPIOB6—Pull-up Enable	Control	BC_1	–	–	–	–	–	–
148	HOME0—Input	Input/Output	BC_1	–	–	69	95	150	A5
149	HOME0—Output	Input/Output	BC_1	–	–	69	95	150	A5
150	HOME0—Output Enable	Control	BC_1	–	–	–	–	–	–
151	HOME0—Pull-up Enable	Control	BC_1	–	–	–	–	–	–
152	MPIOB5—Input	Input/Output	BC_1	–	–	–	94	45	N4
153	MPIOB5—Output	Input/Output	BC_1	–	–	–	94	45	N4
154	MPIOB5—Output Enable	Control	BC_1	–	–	–	–	–	–
155	MPIOB5—Pull-up Enable	Control	BC_1	–		–	–	–	–
156	INDEX0—Input	Input/Output	BC_1	–	–	68	93	149	B6
157	INDEX0—Output	Input/Output	BC_1	–	–	68	93	149	B6
158	INDEX0—Output Enable	Control	BC_1	–	–	–	–	–	–

Table 18-5. BSR Contents for 56F80x (Continued)

Bit #	Pin/Bit Name	Pin Type	BSR Cell Type	56F801 Pin # (48 LQFP Pkg.)	56F802 Pin # (48 LQFP Pkg.)	56F803 Pin # (100 LQFP Pkg.)	56F805 Pin # (144 LQFP Pkg.)	56F807 Pin # (144 LQFP Pkg.)	56F807 Pin # (160 MAPBGA Pkg.)
159	INDEX0—Pull-up Enable	Control	BC_1	–	–	–	–	–	–
160	MPIOB4—Input	Input/Output	BC_1	–	–	–	92	44	P3
161	MPIOB4—Output	Input/Output	BC_1	–	–	–	92	44	P3
162	MPIOB4—Output Enable	Control	BC_1	–	–	–	–	–	–
163	MPIOB4—Pull-up Enable	Control	BC_1	–	–	–	–	–	–
164	MPIOB3—Input	Input/Output	BC_1	–	–	–	90	43	P2
165	MPIOB3—Output	Input/Output	BC_1	–	–	–	90	43	P2
166	MPIOB3—Output Enable	Control	BC_1	–	–	–	–	–	–
167	MPIOB3—Pull-up Enable	Control	BC_1	–	–	–	–	–	–
168	MPIOB2—Input	Input/Output	BC_1	–	–	–	88	42	N3
169	MPIOB2—Output	Input/Output	BC_1	–	–	–	88	42	N3
170	MPIOB2—Output Enable	Control	BC_1	–	–	–	–	–	–
171	MPIOB2—Pull-up Enable	Control	BC_1	–	–	–	–	–	–
172	PHASEB0—Input	Input/Output	BC_1	–	–	65	87	148	D5
173	PHASEB0—Output	Input/Output	BC_1	–	–	65	87	148	D5
174	PHASEB0—Output Enable	Control	BC_1	–	–	–	–	–	–
175	PHASEB0—Pull-up Enable	Control	BC_1	–	–	–	–	–	–
176	MPIOB1—Input	Input/Output	BC_1	–	–	–	86	41	P1
177	MPIOB1—Output	Input/Output	BC_1	–	–	–	86	41	P1
178	MPIOB1—Output Enable	Control	BC_1	–	–	–	–	–	–
179	MPIOB1—Pull-up Enable	Control	BC_1	–	–	–	–	–	–
180	PHASEA0—Input	Input/Output	BC_1	–	–	64	85	147	E6

Table 18-5. BSR Contents for 56F80x (Continued)

Bit #	Pin/Bit Name	Pin Type	BSR Cell Type	56F801 Pin # (48 LQFP Pkg.)	56F802 Pin # (48 LQFP Pkg.)	56F803 Pin # (100 LQFP Pkg.)	56F805 Pin # (144 LQFP Pkg.)	56F807 Pin # (144 LQFP Pkg.)	56F807 Pin # (160 MAPBGA Pkg.)
181	PHASEA0—Output	Input/Output	BC_1	–	–	64	85	147	E6
182	PHASEA0—Output Enable	Control	BC_1	–	–	–	–	–	–
183	PHASEA0—Pull-up Enable	Control	BC_1	–	–	–	–	–	–
184	MPIOB0—Input	Input/Output	BC_1	–	–	–	84	40	L4
185	MPIOB0—Output	Input/Output	BC_1	–	–	–	84	40	L4
186	MPIOB0—Output Enable	Control	BC_1	–	–	–	–	–	–
187	MPIOB0—Pull-up Enable	Control	BC_1	–	–	–	–	–	–
188	FAULTA3	Input	BC_1	–	–	–	67	85	L13
189	FAULTA2	Input	BC_1	–	–	47	66	84	N14
190	MSCAN_RX	Input	BC_1	–	–	46	65		D6
191	FAULTA1	Input	BC_1	–	–	45	64	83	L12
192	MSCAN_TX	Output	BC_1	–	–	44	63		E8
193	FAULTA0	Input	BC_1	30	21	43	62	82	M13
194	RXD1—Input	Input/Output	BC_1	–	–	–	61	56	N7
195	RXD1—Output	Input/Output	BC_1	–	–	–	61	56	N7
196	RXD1—Output Enable	Control	BC_1	–	–	–	–	–	–
197	RXD1—Pull-up Enable	Control	BC_1	–	–	–	–	–	–
198	ISA2	Input	BC_1	–	–	42	60	125	A12
199	ISA1	Input	BC_1	–	–	41	58‘	124	A13
200	ISA0	Input	BC_1	–	–	40	56	123	B12
201	TXD1—Input	Input/Output	BC_1	–	–	–	52	55	P6
202	TXD1—Output	Input/Output	BC_1	–	–	–	52	55	P6
203	TXD1—Output Enable	Control	BC_1	–	–	–	–	–	–
204	TXD1—Pull-up Enable	Control	BC_1	–	–	–	–	–	–
205	TC1—Input	Input/Output	BC_1	–	–	–	50	131	E10
206	TC1—Output	Input/Output	BC_1	–	–	–	50	131	E10

Table 18-5. BSR Contents for 56F80x (Continued)

Bit #	Pin/Bit Name	Pin Type	BSR Cell Type	56F801 Pin # (48 LQFP Pkg.)	56F802 Pin # (48 LQFP Pkg.)	56F803 Pin # (100 LQFP Pkg.)	56F805 Pin # (144 LQFP Pkg.)	56F807 Pin # (144 LQFP Pkg.)	56F807 Pin # (160 MAPBGA Pkg.)
207	TC1— Output Enable	Control	BC_1	–	–	–	–	–	–
208	TC1— Pull-up Enable	Control	BC_1	–	–	–	–	–	–
209	TC0— Input	Input/ Output	BC_1	–	–	–	48	130	A10
210	TC0— Output	Input/ Output	BC_1	–	–	–	48	130	A10
211	TC0— Output Enable	Control	BC_1	–	–	–	–	–	–
212	TC0— Pull-up Enable	Control	BC_1	–	–	–	–	–	–
213	FAULTB3	Input	BC_1	–	–	–	46	74	M11
214	FAULTB2	Input	BC_1	–	–	–	44	73	L11
215	$\overline{\text{IRQB}}$	Input	BC_1	–	–	32	43	70	P10
216	$\overline{\text{IRQA}}$	Input	BC_1	–	–	31	42	69	N9
217	$\overline{\text{RD}}$— Input	Input/ Output	BC_1	–	–	30	41	22	K4
218	$\overline{\text{RD}}$— Output	Input/ Output	BC_1	–	–	30	41	22	K4
219	$\overline{\text{RD}}$— Output Enable	Control	BC_1	–	–	–	–	–	–
220	$\overline{\text{RD}}$— Pull-up Enable	Control	BC_1	–	–	–	–	–	–
221	$\overline{\text{WR}}$— Input	Input/ Output	BC_1	–	–	29	40	21	J4
222	$\overline{\text{WR}}$— Output	Input/ Output	BC_1	–	–	29	40	21	J4
223	$\overline{\text{WR}}$— Output Enable	Control	BC_1	–	–	–	–	–	–
224	$\overline{\text{WR}}$— Pull-up Enable	Control	BC_1	–	–	–	–	–	–
225	A15— Input	Input /Output	BC_1	–	–	27	38	17	G1
226	A15— Output	Input/ Output	BC_1	–	–	27	38	17	G1
227	A15— Output Enable	Control	BC_1	–	–	–	–	–	–
228	A15— Pull-up Enable	Control	BC_1	–	–	–	–	–	–
229	A14— Input	Input/ Output	BC_1	–	–	26	37	16	G2

Table 18-5. BSR Contents for 56F80x (Continued)

Bit #	Pin/Bit Name	Pin Type	BSR Cell Type	56F801 Pin # (48 LQFP Pkg.)	56F802 Pin # (48 LQFP Pkg.)	56F803 Pin # (100 LQFP Pkg.)	56F805 Pin # (144 LQFP Pkg.)	56F807 Pin # (144 LQFP Pkg.)	56F807 Pin # (160 MAPBGA Pkg.)
230	A14—Output	Input/Output	BC_1	–	–	26	37	16	G2
231	A14—Output Enable	Control	BC_1	–	–	–	–	–	–
232	A14—Pull-up Enable	Control	BC_1	–	–	–	–	–	–
233	$\overline{DS}$—Input	Input/Output	BC_1	–	–	25	36	20	H4
234	$\overline{DS}$—Output	Input/Output	BC_1	–	–	25	36	20	H4
235	$\overline{DS}$—Output Enable	Control	BC_1	–	–	–	–	–	–
236	$\overline{DS}$—Pull-up Enable	Control	BC_1	–	–	–	–	–	–
237	$\overline{PS}$—Input	Input/Output	BC_1	–	–	24	35	19	G4
238	$\overline{PS}$—Output	Input/Output	BC_1	–	–	24	35	19	G4
239	$\overline{PS}$—Output Enable	Control	BC_1	–	–	–	–	–	–
240	$\overline{PS}$—Pull-up Enable	Control	BC_1	–	–	–	–	–	–
241	A13—Input	Input/Output	BC_1	–	–	22	33	15	G3
242	A13—Output	Input/Output	BC_1	–	–	22	33	15	G3
243	A13—Output Enable	Control	BC_1	–	–	–	–	–	–
244	A13—Pull-up Enable	Control	BC_1	–	–	–	–	–	–
245	A12—Input	Input/Output	BC_1	–	–	21	32	14	F1
246	A12—Output	Input/Output	BC_1	–	–	21	32	14	F1
247	A12—Output Enable	Control	BC_1	–	–	–	–	–	–
248	A12—Pull-up Enable	Control	BC_1	–	–	–	–	–	–
249	FAULTB1	Input	BC_1	–	–	–	31	72	N10
250	A11—Input	Input/Output	BC_1	–	–	20	30	13	F2

Table 18-5. BSR Contents for 56F80x (Continued)

Bit #	Pin/Bit Name	Pin Type	BSR Cell Type	56F801 Pin # (48 LQFP Pkg.)	56F802 Pin # (48 LQFP Pkg.)	56F803 Pin # (100 LQFP Pkg.)	56F805 Pin # (144 LQFP Pkg.)	56F807 Pin # (144 LQFP Pkg.)	56F807 Pin # (160 MAPBGA Pkg.)
251	A11—Output	Input/Output	BC_1	–	–	20	30	13	F2
252	A11—Output Enable	Control	BC_1	–	–	–	–	–	–
253	A11—Pull-up Enable	Control	BC_1	–	–	–	–	–	–
254	FAULTB0	Input	BC_1	–	–	–	29	71	P11
255	A10—Input	Input/Output	BC_1	–	–	19	28	12	F3
256	A10—Output	Input/Output	BC_1	–	–	19	28	12	F3
257	A10—Output Enable	Control	BC_1	–	–	–	–	–	–
258	A10—Pull-up Enable	Control	BC_1	–	–	–	–	–	–
259	ISB2	Input	BC_1	–	–	–	27	67	P9
260	A9—Input	Input/Output	BC_1	–	–	18	26	11	E1
261	A9—Output	Input/Output	BC_1	–	–	18	26	11	E1
262	A9—Output Enable	Control	BC_1	–	–	–	–	–	–
263	A9—Pull-up Enable	Control	BC_1	–	–	–	–	–	–
264	ISB1	Input	BC_1	–	–	–	25	66	K9
265	A8—Input	Input/Output	BC_1	–	–	17	24	10	E2
266	A8—Output	Input/Output	BC_1	–	–	17	24	10	E2
267	A8—Output Enable	Control	BC_1	–	–	–	–	–	–
268	A8—Pull-up Enable	Control	BC_1	–	–	–	–	–	–
269	ISB0	Input	BC_1	–	–	–	23	64	N8
270	A7—Input	Input/Output	BC_1	–	–	16	22	8	D1
271	A7—Output	Input/Output	BC_1	–	–	16	22	8	D1
272	A7—Output Enable	Control	BC_1	–	–	–	–	–	–
273	A7—Pull-up Enable	Control	BC_1	–	–	–	–	–	–

Table 18-5. BSR Contents for 56F80x (Continued)

Bit #	Pin/Bit Name	Pin Type	BSR Cell Type	56F801 Pin # (48 LQFP Pkg.)	56F802 Pin # (48 LQFP Pkg.)	56F803 Pin # (100 LQFP Pkg.)	56F805 Pin # (144 LQFP Pkg.)	56F807 Pin # (144 LQFP Pkg.)	56F807 Pin # (160 MAPBGA Pkg.)
274	PWMB5— Output	Output	BC_1	–	–	–	21	62	P8
275	PWMB5— Output Enable	Control	BC_1	–	–	–	–	–	–
276	A6— Input	Input/ Output	BC_1	–	–	15	20	7	C1
277	A6— Output	Input/ Output	BC_1	–	–	15	20	7	C1
278	A6— Output Enable	Control	BC_1	–	–	–	–	–	–
279	A6— Pull-up Enable	Control	BC_1	–	–	–	–	–	–
280	PWMB4— Output	Output	BC_1	–	–	–	19	61	K8
281	PWMB4— Output Enable	Control	BC_1	–	–	–	–	–	–
282	A5— Input	Input/Out put	BC_1	–	–	14	18	6	D2
283	A5— Output	Input/Out put	BC_1	–	–	14	18	6	D2
284	A5— Output Enable	Control	BC_1	–	–	–	–	–	–
285	A5— Pull-up Enable	Control	BC_1	–	–	–	–	–	–
286	A4— Input	Input/ Output	BC_1	–	–	13	17	5	B1
287	A4— Output	Input/ Output	BC_1	–	–	13	17	5	B1
288	A4— Output Enable	Control	BC_1	–	–	–	–	–	–
289	A4— Pull-up Enable	Control	BC_1	–	–	–	–	–	–
290	A3— Input	Input/ Output	BC_1	–	–	12	16	4	C2
291	A3— Output	Input/ Output	BC_1	–	–	12	16	4	C2
292	A3— Output Enable	Control	BC_1	–	–	–	–	–	–
293	A3— Pull-up Enable	Control	BC_1	–	–	–	–	–	–
294	PWMB3— Output	Output	BC_1	–	–	–	15	60	L8
295	PWMB3— Output Enable	Control	BC_1	–	–	–	–	–	–

Table 18-5. BSR Contents for 56F80x (Continued)

Bit #	Pin/Bit Name	Pin Type	BSR Cell Type	56F801 Pin # (48 LQFP Pkg.)	56F802 Pin # (48 LQFP Pkg.)	56F803 Pin # (100 LQFP Pkg.)	56F805 Pin # (144 LQFP Pkg.)	56F807 Pin # (144 LQFP Pkg.)	56F807 Pin # (160 MAPBGA Pkg.)
296	A2— Input	Input/ Output	BC_1	–	–	11	14	3	D3
297	A2— Output	Input/Output	BC_1	–	–	11	14	3	D3
298	A2— Output Enable	Control	BC_1	–	–	–	–	–	–
299	A2— Pull-up Enable	Control	BC_1	–	–	–	–	–	–
300	PWMB2— Output	Output	BC_1	–	–	–	13	59	K7
301	PWMB2— Output Enable	Control	BC_1	–	–	–	–	–	–
302	A1— Input	Input/ Output	BC_1	–	–	10	12	2	B2
303	A1— Output	Input/ Output	BC_1	–	–	10	12	2	B2
304	A1— Output Enable	Control	BC_1	–	–	–	–	–	–
305	A1— Pull-up Enable	Control	BC_1	–	–	–	–	–	–
306	PWMB1— Output	Output	BC_1	–	–	–	11	58	P7
307	PWMB1— Output Enable	Control	BC_1	–	–	–	–	–	–
308	PWMB0— Output	Output	BC_1	–	–	–	9	57	L7
309	PWMB0— Output Enable	Control	BC_1	–	–	–	–	–	–
310	A0— Input	Input/ Output	BC_1	–	–	7	7	1	C3
311	A0— Output	Input/ Output	BC_1	–	–	7	7	1	C3
312	A0— Output Enable	Control	BC_1	–	–	–	–	–	–
313	A0— Pull-up Enable	Control	BC_1	–	–	–	–	–	–
314	D15— Input	Input/ Output	BC_1	–	–	6	6	39	M3
315	D15— Output	Input/ Output	BC_1	–	–	6	6	39	M3
316	D15— Output Enable	Control	BC_1	–	–	–	–	–	–
317	D15— Pull-up Enable	Control	BC_1	–	–	–	–	–	–

Table 18-5. BSR Contents for 56F80x (Continued)

Bit #	Pin/Bit Name	Pin Type	BSR Cell Type	56F801 Pin # (48 LQFP Pkg.)	56F802 Pin # (48 LQFP Pkg.)	56F803 Pin # (100 LQFP Pkg.)	56F805 Pin # (144 LQFP Pkg.)	56F807 Pin # (144 LQFP Pkg.)	56F807 Pin # (160 MAPBGA Pkg.)
318	D14—Input	Input/Output	BC_1	–	–	5	5	38	N2
319	D14—Output	Input/Output	BC_1	–	–	5	5	38	N2
320	D14—Output Enable	Control	BC_1	–	–	–	–	–	–
321	D14—Pull-up Enable	Control	BC_1	–	–	–	–	–	–
322	D13—Input	Input/Output	BC_1	–	–	4	4	37	M2
323	D13—Output	Input/Output	BC_1	–	–	4	4	37	M2
324	D13—Output Enable	Control	BC_1	–	–	–	–	–	–
325	D13—Pull-up Enable	Control	BC_1	–	–	–	–	–	–
326	D12—Input	Input/Output	BC_1	–	–	3	3	36	N1
327	D12—Output	Input/Output	BC_1	–	–	3	3	36	N1
328	D12—Output Enable	Control	BC_1	–	–	–	–	–	–
329	D12—Pull-up Enable	Control	BC_1	–	–	–	–	–	–
330	D11—Input	Input/Output	BC_1	–	–	2	2	35	L2
331	D11—Output	Input/Output	BC_1	–	–	2	2	35	L2
332	D11—Output Enable	Control	BC_1	–	–	–	–	–	–
333	D11—Pull-up Enable	Control	BC_1	–	–	–	–	–	–
334	D10—Input	Input/Output	BC_1	–	–	1	1	33	L3
335	D10—Output	Input/Output	BC_1	–	–	1	1	33	L3
336	D10—Output Enable	Control	BC_1	–	–	–	–	–	–
337	D10—Pull-up Enable	Control	BC_1	–	–	–	–	–	–

18.6.4 JTAG Bypass Register (JTAGBR)

The JTAG Bypass Register is a one-bit register used to provide a simple, direct path from the TDI pin to the TDO pin. This is useful in boundary scan applications where many chips are serially connected in a daisy-chain. Individual controllers, or other devices, can be programmed with the BYPASS instruction so individually they become pass-through devices during testing. This allows testing of a specific chip, while still having all of the chips connected through the JTAG ports.

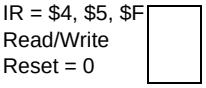


Figure 18-7. JTAG Bypass Register (JTAGBR)

18.7 TAP Controller

The TAP Controller is a synchronous finite state machine containing sixteen states, as illustrated in **Figure 18-8.** The TAP Controller responds to changes at the TMS and TCK signals. Transitions from one state to another occur on the rising edge of TCK. The value shown adjacent to each state transition in this figure represents the signal present at TMS at the time of a rising edge at TCK. The TDO pin remains in the high impedance state except during the Shift-DR or Shift-IR Controller states. In these controller states, TDO is updated on the falling edge of TCK. TDI is sampled on the rising edge of TCK.

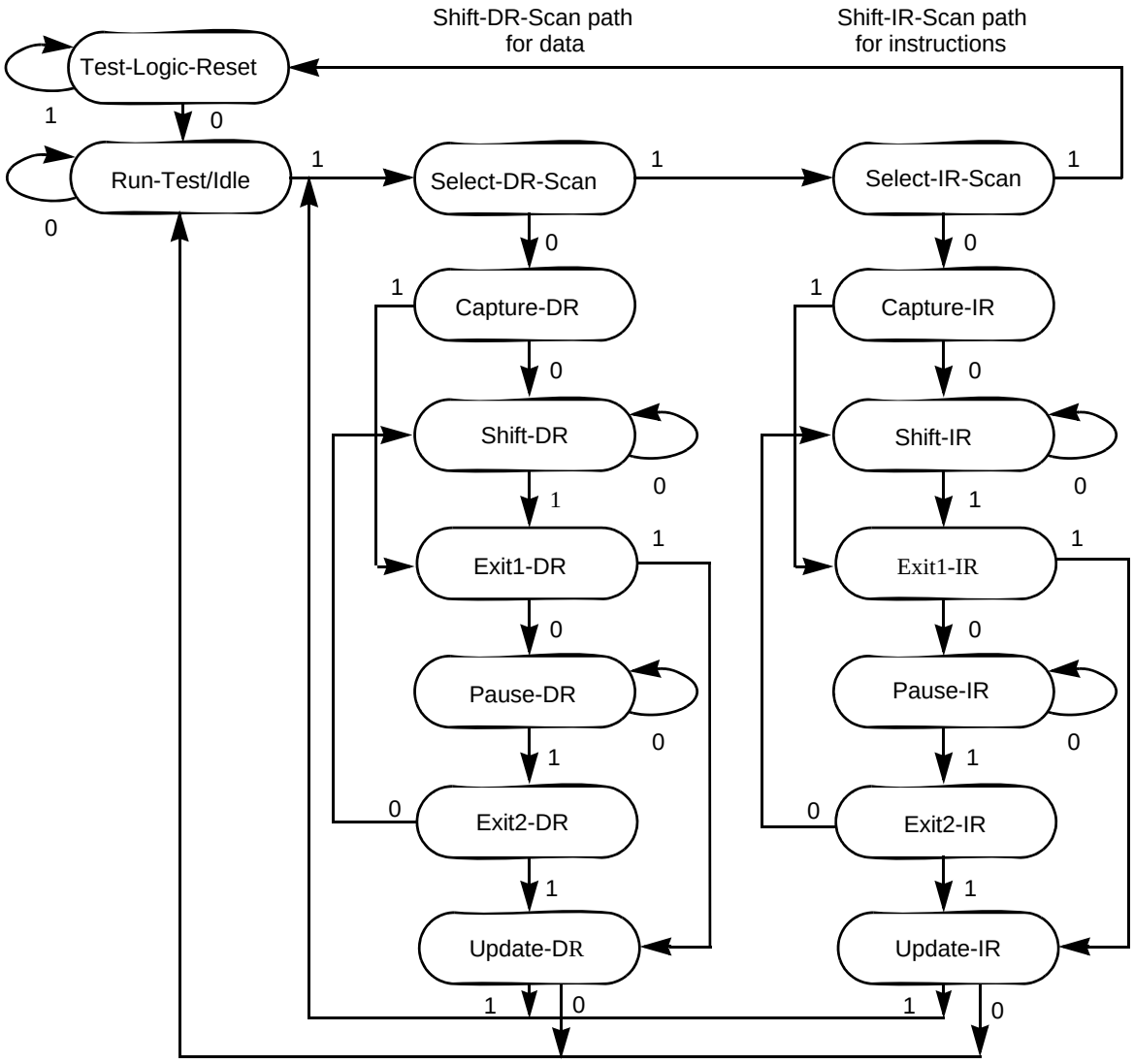


Figure 18-8. TAP Controller State Diagram

There are two paths through the 16-state machine.

1. The Shift-IR-Scan path captures and loads JTAG instructions into the JTAGIR.
2. The Shift-DR-Scan path captures and loads data into the other JTAG registers.

The TAP controller executes the last instruction decoded until a new instruction is entered at the Update-IR state, or until the Test-Logic-Reset state is entered. When using the JTAG port to access OnCE module registers, accesses are first enabled by shifting the ENABLE_ONCE instruction into the JTAGIR. After this is selected, the OnCE module registers and commands are read and written through the JTAG pins using the Shift-DR-Scan path. Asserting the JTAG's $\overline{\text{TRST}}$ pin asynchronously forces the JTAG state machine into the Test-Logic-Reset state.

18.8 56F80x Restrictions

The control afforded by the output enable signals using the BSR and the EXTEST instruction requires a compatible circuit board test environment to avoid any device-destructive configurations. *Avoid situations when the 56F80x output drivers are enabled into actively driven networks.*

During power-up, the $\overline{\text{TRST}}$ pin must be externally asserted to force the TAP controller into this state. After power-up is concluded, TMS must be sampled as a Logic 1 for five consecutive TCK rising edges. If TMS either remains unconnected or is connected to V_{DD}, then the TAP controller cannot leave the Test-Logic-Reset state, regardless of the state of TCK.

56F80x features a low-power Stop mode invoked using the stop instruction. JTAG interaction with low-power Stop mode is as follows:

1. The TAP controller must be in the Test-Logic-Reset state to either enter or remain in Stop mode. Leaving the TAP controller Test-Logic-Reset state negates the ability to achieve low-power, but does not otherwise affect device functionality.
2. The TCK input is not blocked in low-power Stop mode. To consume minimal power, the TCK input should be externally connected to V_{DD} or ground.
3. The TMS and TDI pins include on-chip pull-up resistors. In low-power Stop mode, these two pins should remain either unconnected or connected to V_{DD} to achieve minimal power consumption.

Because all 56F80x clocks are disabled during Stop state, the JTAG interface provides the means of polling the device status, sampled in the Capture-IR state.

Table 18-6. Document Revision History for Chapter 18

Version History	Description of Change
Rev. 8	Formatting, layout, spelling, and grammar corrections. Added revision history table.

Appendix A
Programmer's Sheets

A.1 Introduction

The following pages provide a set of reference tables and programming sheets intended to simplify programming the 56F800 Family. The programming sheets provide room to add the value of each bit and the hexadecimal value for each register. These pages may be photocopied. Copy at 122 percent to enlarge these pages to a full 8-1/2 x 11 inch sheet.

A.2 Notation

The following tables provide brief summaries of the instruction set for the 56F800 Family. For complete instruction set details, refer to Appendix A of the *DSP56800 Family Manual (DSP56800FM).*

Each described instruction contains notations used to abbreviate certain operands and operations described in **Table A-7.** Keys to the symbols and their respective descriptions of the *Set Instructions* located in **Table A-7** are listed in **Table A-1** through **A-6.**

Table A-1 illustrates the register set available for the most important move instruction. Sometimes the register field is broken into two different fields--one where the register is used as a source and the other where it is used as a destination. This is important because a different notation is used when an accumulator is being stored without saturation.

Table A-1. Register Fields for General-Purpose Writes and Reads

Register Field	Registers in This Field	Comments
HHH	A, B, A1, B1 X0, Y0, Y1	Seven data ALU registers---two accumulators, two 16-bit MSP portions of the accumulators and three 16-bit data registers
HHHH	A, B, A1, B1 X0, Y0,Y1 R0-R3, N	Seven data ALU and five AGU registers
DDDDD	A, A2, A1, A0 B, B2, B1, B0 Y1,Y0, X0 R0, R1, R2, R3 N, SP M01 OMR, SR LA, LC HWS	All CPU registers

Table A-2 illustrates the register set available for use as pointers in address-register- indirect addressing modes. The most common fields used in this table are Rn and RRR. This table also shows the notations used for AGU registers in AGU arithmetic operations.

Table A-2. Data AGU Registers

Register Field	Registers in This Field	Comments
Rn	R0-R3 N	Five AGU registers available as pointers for addressing and as sources and destinations for move instructions
Rj	R0, R1, R2, R3	Four pointer registers available as pointers for addressing
N	N	One index register available only for indexed addressing modes
M01	M01	One modifier register

Table A-3 illustrates the register set available for use in data ALU arithmetic operations. The common field used in this table is FFF.

Table A-3. Data ALU Registers

Register Field	Registers in This Field	Comments
FDD	A, B X0,YO,Y1	Five data ALU registers--two 36-bit accumulators and three 16-bit data registers accessible during data ALU operations Contains the contents of the F and DD register fields
F1DD	A1,B1 X0,Y0,Y1	Five data ALU registers--two 16-bit MSP portions of the accumulators and three 16-bit data registers accessible during data ALU operations
F	A, B	Two 36-bit data registers
DD	X0, Y0, Y1	Three 16-bit data registers
F	A, B	Two 36-bit accumulators accessible during ALU operations
F1	A1, B1	The 16-bit MSP portion of two accumulators accessible as source operands in parallel move instructions

Address operands used in the instruction field sections of the instruction descriptions are provided in **Table A-4.**

Table A-4. Address Operands

Symbol	Description
ea	Effective address
eax	Effective address for X bus
xxxx	Absolute address (16 bits)
pp	I/O short address (6 bits, one-extended)
aa	Absolute address (6 bits, zero-extended)
<...>	Specifies the contents of the specified address
X:	X memory reference
P:	Program memory reference

Addressing mode operations accepted by the assembler for stipulating a specific addressing mode are provided in **Table A-5.**

Table A-5. Addressing Mode Operators

Symbol	Description
<<	I/O short or absolute addressing mode force operator
>	Long addressing mode force operator
#	Immediate addressing mode operator
#>	Immediate long addressing mode for operator
#<	Immediate short addressing mode force operator

Miscellaneous operand notation, including generic source and destination operands and immediate data specifiers, are summarized in **Table A-6.**

Table A-6. Miscellaneous Operands

Symbol	Description
S, Sn	Source operand register
D, Dn	Destination operand register
#xx	Immediate short data (7 bits for MOVE (I), 6 bits for DO/REP)
#xxxx	Immediate data (16 bits)
#ii00	8-bit immediate data mask in the upper byte
#00ii	8-bit immediate data mask in the lower byte

A.3 Instruction Set Summary

Please refer to the *DSP56800 Family Manual (DSP56800FM)*, **Section A.4.4** for a *complete* summary of notations used in tables. Only those notations used in the following table are represented here.

* = Set by the result of the operation according to the standard definition
– = Not affected by the operation
0 = Cleared
? = Set according to the special computation defined for the operation

Table A-7. Instruction Set Summary

					CCR							
Mnemonic	**Syntax**	**Parallel Moves**	**Prog. Word**	**Clock Cycles**	**SZ**	**L**	**E**	**U**	**N**	**Z**	**V**	**C**
ABS	D	(parallel move)	1	2 + mv	*	*	*	*	*	*	*	—
ADC	S,D	(no parallel move)	1	2	—	*	*	*	*	*	*	*
ADD	S,D	(parallel move)	1	2 + mv	*	*	*	*	*	*	*	*
AND	S,D	(no parallel move)	1	2	—	*	—	—	?	?	0	—
ANDC	#iiii,X:<ea> #iiii,D	. . .	2 + ea	4 + mvb	—	—	—	—	—	—	—	?
ASL	D	(parallel move)	1	2 + mv	*	*	*	*	*	*	?	?
ASLL	S1,S2,D	(no parallel move)	1	2	—	—	—	—	*	*	—	—
ASR	D	(parallel move)	1	2 + mv	*	*	*	*	*	*	0	?
ASRAC	S1,S2,D	(no parallel move)	1	2	—	—	—	—	*	*	—	—
ASRR	S1,S2,D	(no parallel move)	1	2	—	—	—	—	*	*	—	—
BCC	xxxx ee Rn	. . .	2 1 1	4 + jx	—	—	—	—	—	—	—	—
BFCHG	#iii,X:<aa> #iii,X:<pp> #iii,X:<ea> #iii,D	. . .	2 + ea	4 + mvb 4 + mvb 6 + mvb 4 + mvb	—	—	—	—	—	—	—	?
BFCLR	#iii,X:<aa> #iii,X:<pp> #iii,X:<ea> #iii,D	. . .	2 + ea	4 + mvb 4 + mvb 6 + mvb 4 + mvb	—	—	—	—	—	—	—	?
BFSET	#iii,X:<aa> #iii,X:<pp> #iii,X:<ea> #iii,D	. . .	2 + ea	4 + mvb 4 + mvb 6 + mvb 4 + mvb	—	—	—	—	—	—	—	?
BFTSTH	#iii,X:<aa> #iii,X:<pp> #iii,X:<ea> #iii,D	. . .	2 + ea	4 + mvb 4 + mvb 6 + mvb 4 + mvb	—	—	—	—	—	—	—	?
BFTSTL	#iii,X:<aa> #iii,X:<pp> #iii,X:<ea> #iii,D	. . .	2 + ea	4 + mvb 4 + mvb 6 + mvb 4 + mvb	—	—	—	—	—	—	—	?

Table A-7. Instruction Set Summary (Continued)

					CCR							
Mnemonic	**Syntax**	**Parallel Moves**	**Prog. Word**	**Clock Cycles**	**SZ**	**L**	**E**	**U**	**N**	**Z**	**V**	**C**
BRA	xxxx aa Rn	. . .	2 1 1	6 + jx	—	—	—	—	—	—	—	—
BRCLR	#iiii,X:<ea> ,aa #iiii,D,aa	. . .	2 + ea	8 + mvb	—	—	—	—	—	—	—	?
BRSET			2 + ea	8 + mvb	—	—	—	—	—	—	—	?
CLR	D	(parallel move)	1	2 + mv	*	*	*	*	*	*	0	—
CMP	S,D	(parallel move)	1 + ea	2 + mv	*	*	*	*	*	*	*	*
DEBUG		. . .	1	4	—	—	—	—	—	—	—	—
DEC(W)	D	(parallel move)	1 + ea	2 + mv	*	*	*	*	*	?	*	*
DIV	S,D	(parallel move)	1	2	—	*	—	—	—	—	?	
DO	X:(Rn),expr #xx,expr S,expr	(no parallel move)	2	6	—	*	—	—	—	—	—	—
ENDDO		. . .	1	2	—	—	—	—	—	—	—	—
EOR	S,D	. . .	1	2	—	*	—	—	?	?	0	?
EORC	#iiii,X:<ea> #iiii,D	. . .	2 + ea	4 + mvb	—	—	—	—	—	—	—	?
ILLEGAL		(no parallel move)	1	4	—	—	—	—	—	—	—	—
IMPY(16)	S1,S2,D	(no parallel move)	1	2	—	*	?	?	*	*	*	—
INC(W)	D	(parallel move)	1 + ea	2 + mv	*	*	*	*	*	?	*	*
JCC	xxxx (Rn)	. . .	2	4 + jx	—	—	—	—	—	—	—	—
JMP	xxxx (Rn)	. . .	2 1	6 + jx	—	—	—	—	—	—	—	—
JSR	xxxx AA Rn	. . .	2 1 1	8 + jx	—	—	—	—	—	—	—	—
LEA	ea,D	(no parallel move)	1 + ea	2 + ea	—	—	—	—	—	—	—	—
LSL	D	(no parallel move)	1	2	—	*	—	—	?	?	0	?

Table A-7. Instruction Set Summary (Continued)

					CCR							
Mnemonic	**Syntax**	**Parallel Moves**	**Prog. Word**	**Clock Cycles**	**SZ**	**L**	**E**	**U**	**N**	**Z**	**V**	**C**
LSLL	S1,S2,D	(no parallel move)	1	2	—	—	—	—	*	*	—	—
LSR	D	(no parallel move)	1	2	—	*	—	—	?	?	0	?
LSRAC	S1,S2,D	(no parallel move)	1	2	—	—	—	—	*	*	—	—
LSRR	S1,S2,D	(no parallel move)	1	2	—	—	—	—	*	*	—	
MAC	(±)S2,S1,D S2,S1,D S1,S2,D	(no parallel move) (one parallel move) (two parallel reads)	1	2 + mv	*	*	*	*	*	*	*	—
MACR	(±)S2,S1,D S2,S1,D S1,S2,D	(no parallel move) (one parallel move) (two parallel reads)	1	2 + mv	*	*	*	*	*	*	*	—
MACSU	S1,S2,D	(no parallel move)	1	2	—	*	*	*	*	*	*	—
MOVE[1]	X:<ea>,D S,X:<ea>	. . .	1 + ea	2 + ea	*	*	—	—	—	—	—	—
MOVE(C)	X:<ea>,D S,X:<ea> #xxxx,D S,D X:(R2 + xx),D S,X:(R2 + xx)	. . .	1 + ea	2 + mvc	*	?	?	?	?	?	?	?
MOVE(I)	#xx,D	. . .	1	2	—	—	—	—	—	—	—	
MOVE(M)	P:<ea>,D S,P:<ea> P:(R2 + xx),D S,P:(R2 + xx) P:<ea>,X:<ea> X:<ea>,P:<ea>	. . .	1	8 + mvm	—	*	—	—	—	—	—	—

Table A-7. Instruction Set Summary (Continued)

					CCR							
Mnemonic	**Syntax**	**Parallel Moves**	**Prog. Word**	**Clock Cycles**	**SZ**	**L**	**E**	**U**	**N**	**Z**	**V**	**C**
MOVE(P)	X:<pp>,D X:<ea>,X:<pp> S,X:<pp> X:<pp>,X:<ea>	. . .	1	1 + mvp	—	—	—	—	—	—	—	—
MOVE(S)	X:<a>,D S,X:<aa>	. . .	1	2 + mvs	*	*	—	—	—	—	—	—
MPY	(±)S1,S2,D S1,S2,D S1,S2,D	(one parallel move) (two parallel reads) $\overline{D}$,X:(Rn) + (Nn)	1	2 + mv	*	*	*	*	*	*	*	—
MPYR	(±)S1,S2,D S1,S2,D	(one parallel move) (two parallel reads)	1	2 + mv	*	*	*	*	*	*	*	—
MPYSU	S1,S2,D	(no parallel move)	1	2	—	*	*	*	*	*	*	—
NEG	D	(parallel move)	1	2 + mv	*	*	*	*	*	*	*	*
NOP		. . .	1	2	—	—	—	—	—	—	—	—
NORM	Rn,D		1	2	—	*	*	*	*	*	?	—
NOT	D	(no parallel move)	1	2	—	*	—	—	?	?	0	—
NOTC	X:<ea> D	. . .	2 + ea	4 + mvb	—	—	—	—	—	—	—	—
OR	S,D	(no parallel move)	1	2	—	*	—	—	?	?	0	—
ORC	#iiii,X:<ea> #iiii,D	. . .	2 + ea	4 + mvb	—	—	—	—	—	—	—	?
POP	D	. . .	2	2 + mv	—	?	?	?	?	?	?	?
REP	X:(Rn) #xx S	. . .	1	6	—	—	—	—	—	—	—	—
RND	D	(parallel move)	1	2 + mv	*	*	*	*	*	*	*	—
ROL	D	(parallel move)	1	2 + mv	—	*	—	—	?	?	0	?
ROR	D	(parallel move)	1	2 + mv	—	*	—	—	?	?	0	?

Table A-7. Instruction Set Summary (Continued)

					CCR							
Mnemonic	**Syntax**	**Parallel Moves**	**Prog. Word**	**Clock Cycles**	**SZ**	**L**	**E**	**U**	**N**	**Z**	**V**	**C**
RTI		. . .	1	10 + rx	—	—	?	?	?	?	?	?
RTS		. . .	1	10 + rx	—	—	—	—	—	—	—	—
SBC	S,D	(no parallel move)	1	2	—	*	*	*	*	*	*	*
STOP[2]		. . .	1	n/a	—	—	—	—	—	—	—	—
SUB	S,D S,D	(parallel move) (two parallel reads)	1 + ea	2 + mv	*	*	*	*	*	*	*	*
SWI		. . .	1	8	—	—	—	—	—	—	—	—
TCC	S,D S,D R0,R1	. . .	1	2	—	—	—	—	—	—	—	—
TFR	S,D	(parallel move)	1	2 + mv	?	—	—	—	—	—	—	—
TST	S	(parallel move)	1	2 + mv	*	*	*	*	*	*	0	—
TST(W)	S	(no parallel move)	1	2 + tst	—	*	*	*	*	*	0	0
WAIT[3]		. . .	1	n/a	—	—	—	—	—	—	—	—

1.This instruction applies only in the case when two reads are performed in parallel from the X memory.

2.The STOP instruction disables the internal clock oscillator. After the clock is turned on, an internal counter waits for 65,536 cycles before enabling the clock to the internal controller circuits.

3.The WAIT instruction takes a minimum of 16 cycles to execute when an internal interrupt is pending during the execution of the WAIT instruction.

A.4 Interrupt, Vector, and Address Tables

Refer to **Chapter 3, Section 3.5** and **3.10** for interrupt, vector, and address tables. Table titles are listed below.

A.5 Programmer's Sheets

The following pages provide programmer's sheets summarizing functions of the bits in various registers in the 56F800 Family. The programmer's sheets provide room to write in the value of each bit and the hexadecimal value for each register. Programmers may photocopy these sheets.

The programmer's sheets are arranged in the same order as the sections in this document. **Table B-1** lists the programmer's sheets by module, the registers in each module, and the pages in this appendix where the programmer's sheets are located.

Note: Reserved bits should only be set to zero unless otherwise stated.

Table A-8. List of Programmer's Sheets

Register Type	Register	Page/ Figure
CPU Registers		
Bus Control Register	(BCR)	B-19
Operating Mode Register	(OMR)	B-20
Interrupt Priority Regsiter	(IPR)	B-21
OnCE Program Global Data Bus Register	(OPGDBR)	Figure 17-17, Pg. 17-33
ITCN		
arch.h: ArchIO.IntController, registers.h: ArchIO_IntController	ITCN_BASE: 56F801/803/805 = $0E60, 56F807 = $1260	
Group Priority Register	(GPR0)	B-22

Table A-8. List of Programmer's Sheets (Continued)

Register Type	Register	Page/ Figure
Group Priority Register	(GPR1)	B-22
Group Priority Register	(GPR2)	B-22
Group Priority Register	(GPR3)	B-22
Group Priority Register	(GPR4)	B-23
Group Priority Register	(GPR5)	B-23
Group Priority Register	(GPR6)	B-23
Group Priority Register	(GPR7)	B-23
Group Priority Register	(GPR8)	B-24
Group Priority Register	(GPR9)	B-24
Group Priority Register	(GPR10)	B-24
Group Priority Register	(GPR11)	B-24
Group Priority Register	(GPR12)	B-25
Group Priority Register	(GPR13)	B-25
Group Priority Register	(GPR14)	B-25
Group Priority Register	(GPR15)	B-25
FLASH		
arch.h: ArchIO.ProgramFlash registers h:ArchIO_ProgramFlash	PFIU_BASE: 56F801/803/805 = $0F40	
	PFIU_BASE: 56F807 = $1340	
arch.h: ArchIO.DataFlash registers h:ArchIO_DataFlash	DFIU_BASE: 56F801/803/805 = $0F60	
	DFIU_BASE: 56F807 = $1360	
arch.h: ArchIO.BootFlash registers h:ArchIO_BootFlash	BFIU_BASE: 56F801/803/805 = $0F80	
	BFIU_BASE: 56F807 = $1380	
Flash Control Register	(FIU_CNTL)	B-26
Flash Erase Enable Register	(FIU_EE)	B-27
Flash Program Enable Register	(FIU_PE)	B-27
Flash Data Register	(FIU_DATA)	B-28
Flash Interrupt Enable Register	(FIU_IE)	B-29
Flash Interrupt Source Register	(FIU_IS)	B-30
Flash Interrupt Pending Register	(FIU_IP)	B-31
Flash Clock Divisor Register	(FIU_CKDIVISOR)	B-32
Flash Terase Limit Register	(FIU_TERASEL)	B-32
Flash Tme Limit Register	(FIU_TMEL)	B-32
Flash Tpgs Limit Register	(FIU_TPGSL)	B-33
Flash Tprog Limit Register	(FIU_TPROGL)	B-33
Flash Tnvs Limit Register	(FIU_TNVSL)	B-33
Flash Tnvh Limit Register	(FIU_TNVHL)	B-34
Flash Tnvh1 Limit Register	(FIU_TNVH1L)	B-34
Flash Trcv Limit Register	(FIU_TRCVL)	B-34

Table A-8. List of Programmer's Sheets (Continued)

Register Type	Register	Page/ Figure
GPIO		
arch.h:ArchIO.PortA, registers.h:ArchIO_PortA	GPIOA_BASE: 56F801/803/805 = $0FB0	
	GPIOA_BASE: 56F807 = $13B0	
arch.h:ArchIO.PortB, registers.h:ArchIO_PortB	GPIOB_BASE: 56F801/803/805 = $0FC0	
	GPIOB_BASE: 56F807 = $13C0	
arch.h:ArchIO.PortD, registers.h:ArchIO_PortD	GPIOD_BASE: 56F801/803/805 = $0FE0	
	GPIOD_BASE: 56F807 = $13E0	
arch.h:ArchIO.PortE, registers.h:ArchIO_PortE	GPIOE_BASE: 56F801/803/805 = $0FF0	
	GPIOE_BASE: 56F807 = $13F0	
Pull-Up Enable Register	(PUR)	B-35
Data Register	(DR)	B-35
Data Direction Register	(DDR)	B-36
Peripheral Enable Register	(PER)	B-37
Interrupt Enable Register	(IENR)	B-38
Interrupt Assert Register	(IAR)	B-38
Interrupt Pending Register	(IPR)	B-39
Interrupt Polarity Register	(IPOLR)	B-39
Interrupt Edge-Sensitive Register	(IESR)	B-40
CAN		
arch.h: ArchIO.CAN, registers. h:ArchIO_CAN	CAN_BASE: 56F803/805 = $0D80	
	CAN_BASE: 56F807 = $1180	
CAN Control Register 0	(CANCTL0)	B-41
CAN Control Register 1	(CANCTL1)	B-42
CAN Bus Timing Register 0	(CANBTR0)	B-43
CAN Bus Timing Register 1	(CANBTR1)	B-44
CAN Receiver Flag Register	(CANRFLG)	B-45
CAN Receiver Interrupt Enable Register	(CANRIER)	B-46
CAN Transmitter Flag Register	(CANTFLG)	B-47
CAN Transmitter Control Register	(CANTCR)	B-47
CAN Identifier Acceptance Control Register	(CANIDAC)	B-48
CAN Receive Error Counter Register	(CANRXERR)	B-49
CAN Transmit Error Counter Register	(CANTXERR)	B-49
CAN Identifier Acceptance Register	(CANIDAR0)	B-50
CAN Identifier Acceptance Register	(CANIDAR1)	B-50
CAN Identifier Acceptance Register	(CANIDAR2)	B-50
CAN Identifier Acceptance Register	(CANIDAR3)	B-50
CAN Identifier Acceptance Register	(CANIDAR4)	B-51
CAN Identifier Acceptance Register	(CANIDAR5)	B-51
CAN Identifier Acceptance Register	(CANIDAR6)	B-51

Table A-8. List of Programmer's Sheets (Continued)

Register Type	Register	Page/ Figure
CAN Identifier Acceptance Register	(CANIDAR7)	B-51
CAN Identifier Mask Register	(CANIDMR0)	B-52
CAN Identifier Mask Register	(CANIDMR1)	B-52
CAN Identifier Mask Register	(CANIDMR2)	B-52
CAN Identifier Mask Register	(CANIDMR3)	B-52
CAN Identifier Mask Register	(CANIDMR4)	B-53
CAN Identifier Mask Register	(CANIDMR5)	B-53
CAN Identifier Mask Register	(CANIDMR6)	B-53
CAN Identifier Mask Register	(CANIDMR7)	B-53
Standard Mapping: CAN Receive Buffer Identifier Register 0	(CAN_RB_IDR0)	B-54
Standard Mapping: CAN Receive Buffer Identifier Register 1	(CAN_RB_IDR1)	B-54
Standard Mapping: CAN Receive Buffer Identifier Register 2	(CAN_RB_IDR2)	B-54
Standard Mapping: CAN Receive Buffer Identifier Register 3	(CAN_RB_IDR3)	B-54
Standard Mapping: CAN Transmit Buffer 0 Identifier Register 0	(CAN_TB0_IDR0)	B-54
Standard Mapping: CAN Transmit Buffer 0 Identifier Register 1	(CAN_TB0_IDR1)	B-55
Standard Mapping: CAN Transmit Buffer 0 Identifier Register 2	(CAN_TB0_IDR2)	B-55
Standard Mapping: CAN Transmit Buffer 0 Identifier Register 3	(CAN_TB0_IDR3)	B-55
Standard Mapping: CAN Transmit Buffer 1 Identifier Register 0	(CAN_TB1_IDR0)	B-56
Standard Mapping: CAN Transmit Buffer 1 Identifier Register 1	(CAN_TB1_IDR1)	B-56
Standard Mapping: CAN Transmit Buffer 1 Identifier Register 2	(CAN_TB1_IDR2)	B-56
Standard Mapping: CAN Transmit Buffer 1 Identifier Register 3	(CAN_TB1_IDR3)	B-56
Standard Mapping: CAN Transmit Buffer 2 Identifier Register 0	(CAN_TB2_IDR0)	B-57
Standard Mapping: CAN Transmit Buffer 2 Identifier Register 1	(CAN_TB2_IDR1)	B-57
Standard Mapping: CAN Transmit Buffer 2 Identifier Register 2	(CAN_TB2_IDR2)	B-57
Standard Mapping: CAN Transmit Buffer 2 Identifier Register 3	(CAN_TB2_IDR3)	B-57
Extended Mapping: CAN Receive Buffer Identifier Register 0	(CAN_RB_IDR0)	B-58
Extended Mapping: CAN Receive Buffer Identifier Register 1	(CAN_RB_IDR1)	B-58
Extended Mapping: CAN Receive Buffer Identifier Register 2	(CAN_RB_IDR2)	B-58
Extended Mapping: CAN Receive Buffer Identifier Register 3	(CAN_RB_IDR3)	B-58
Extended Mapping: CAN Transmit Buffer 0 Identifier Register 0	(CAN_TB0_IDR0)	B-59
Extended Mapping: CAN Transmit Buffer 0 Identifier Register 1	(CAN_TB0_IDR1)	B-59
Extended Mapping: CAN Transmit Buffer 0 Identifier Register 2	(CAN_TB0_IDR2)	B-59
Extended Mapping: CAN Transmit Buffer 0 Identifier Register 3	(CAN_TB0_IDR3)	B-59
Extended Mapping: CAN Transmit Buffer 1 Identifier Register 0	(CAN_TB1_IDR0)	B-60
Extended Mapping: CAN Transmit Buffer 1 Identifier Register 1	(CAN_TB1_IDR1)	B-60

Table A-8. List of Programmer's Sheets (Continued)

Register Type	Register	Page/ Figure
Extended Mapping: CAN Transmit Buffer 1 Identifier Register 2	(CAN_TB1_IDR2)	B-60
Extended Mapping: CAN Transmit Buffer 1 Identifier Register 3	(CAN_TB1_IDR3)	B-60
Extended Mapping: CAN Transmit Buffer 2 Identifier Register 0	(CAN_TB2_IDR0)	B-61
Extended Mapping: CAN Transmit Buffer 2 Identifier Register 1	(CAN_TB2_IDR1)	B-61
Extended Mapping: CAN Transmit Buffer 2 Identifier Register 2	(CAN_TB2_IDR2)	B-61
Extended Mapping: CAN Transmit Buffer 2 Identifier Register 3	(CAN_TB2_IDR3)	B-61
CAN Receive Buffer Data Segment Register 0	(CAN_RB_DSR0)	B-62
CAN Receive Buffer Data Segment Register 1	(CAN_RB_DSR1)	B-62
CAN Receive Buffer Data Segment Register 2	(CAN_RB_DSR2)	B-62
CAN Receive Buffer Data Segment Register 3	(CAN_RB_DSR3)	B-62
CAN Receive Buffer Data Segment Register 4	(CAN_RB_DSR4)	B-62
CAN Receive Buffer Data Segment Register 5	(CAN_RB_DSR5)	B-62
CAN Receive Buffer Data Segment Register 6	(CAN_RB_DSR6)	B-62
CAN Receive Buffer Data Segment Register 7	(CAN_RB_DSR7)	B-62
CAN Transmit Buffer 0 Data Segment Register 0	(CAN_TB0_DSR0)	B-63
CAN Transmit Buffer 0 Data Segment Register 1	(CAN_TB0_DSR1)	B-63
CAN Transmit Buffer 0 Data Segment Register 2	(CAN_TB0_DSR2)	B-63
CAN Transmit Buffer 0 Data Segment Register 3	(CAN_TB0_DSR3)	B-63
CAN Transmit Buffer 0 Data Segment Register 4	(CAN_TB0_DSR4)	B-63
CAN Transmit Buffer 0 Data Segment Register 5	(CAN_TB0_DSR5)	B-63
CAN Transmit Buffer 0 Data Segment Register 6	(CAN_TB0_DSR6)	B-63
CAN Transmit Buffer 0 Data Segment Register 7	(CAN_TB0_DSR7)	B-63
CAN Transmit Buffer 1 Data Segment Register 0	(CAN_TB1_DSR0)	B-64
CAN Transmit Buffer 1 Data Segment Register 1	(CAN_TB1_DSR1)	B-64
CAN Transmit Buffer 1 Data Segment Register 2	(CAN_TB1_DSR2)	B-64
CAN Transmit Buffer 1 Data Segment Register 3	(CAN_TB1_DSR3)	B-64
CAN Transmit Buffer 1 Data Segment Register 4	(CAN_TB1_DSR4)	B-64
CAN Transmit Buffer 1 Data Segment Register 5	(CAN_TB1_DSR5)	B-64
CAN Transmit Buffer 1 Data Segment Register 6	(CAN_TB1_DSR6)	B-64
CAN Transmit Buffer 1 Data Segment Register 7	(CAN_TB1_DSR7)	B-64
CAN Transmit Buffer 2 Data Segment Register 0	(CAN_TB2_DSR0)	B-65
CAN Transmit Buffer 2 Data Segment Register 1	(CAN_TB2_DSR1)	B-65
CAN Transmit Buffer 2 Data Segment Register 2	(CAN_TB2_DSR2)	B-65
CAN Transmit Buffer 2 Data Segment Register 3	(CAN_TB2_DSR3)	B-65
CAN Transmit Buffer 2 Data Segment Register 4	(CAN_TB2_DSR4)	B-65
CAN Transmit Buffer 2 Data Segment Register 5	(CAN_TB2_DSR5)	B-65
CAN Transmit Buffer 2 Data Segment Register 6	(CAN_TB2_DSR6)	B-65
CAN Transmit Buffer 2 Data Segment Register 7	(CAN_TB2_DSR7)	B-65
CAN Receive Buffer Data Length Register	(CAN_RB_DLR)	B-66
CAN Transmit Buffer 0 Data Length Register	(CAN_TB0_DLR)	B-66

Table A-8. List of Programmer's Sheets (Continued)

Register Type	Register	Page/ Figure
CAN Transmit Buffer 1 Data Length Register	(CAN_TB1_DLR)	B-66
CAN Transmit Buffer 2 Data Length Register	(CAN_TB2_DLR)	B-66
CAN Transmit Buffer 0 Priority Register	(CAN_TB0_TBPR)	B-67
CAN Transmit Buffer 1 Priority Register	(CAN_TB1_TBPR)	B-67
CAN Transmit Buffer 2 Priority Register	(CAN_TB2_TBPR)	B-67
ADC		
arch.h:ArchIO.AdcA, registers.h:ArchIO_AdcA	ADCA_BASE: 56F801/803/805 = $$0E80	
	ADCA_BASE: 56F807 = $1280	
arch.h:ArchIO.AdcB, registers.h:ArchIO_AdcB	ADCB_BASE: 56F801/803/805 = $0EC0	
	ADCB_BASE: 56F807 = $12C0	
ADC Control Register 1	(ADCR1)	B-68
ADC Control Register	(ADCR2)	B-69
ADC Zero Crossing Control Register	(ADZCC)	B-69
ADC Channel List Register 2	(ADLST2)	B-70
ADC Channel List Register 1	(ADLST1)	B-70
ADC Sample Disable Register	(ADSDIS)	B-71
ADC Status Register	(ADSTAT)	B-72
ADC Limit Status Register	(ADLSTAT)	B-73
ADC Zero Crossing Status Register	(ADZCSTAT)	B-74
ADC Result Registers	(ADRSLT0–7)	B-75
ADC Low Limit Register 0–7	(ADLLMT0–7)	B-76
ADC High Limit Register 0–7	(ADHLMT0–7)	B-76
ADC Offset Registers 0–7	(ADOFS0–7)	B-77
DEC		
arch.h: ArchIO.Decoder0, registers.h: ArchIO_Decoder0	DEC0_BASE: 56F801/803/805 = $0E40	
	DEC0_BASE: 56F807 = $1240	
arch.h: ArchIO.Decoder1, registers.h: ArchIO_Decoder1	DEC1_BASE: 56F801/803/805 = $0E50	
	DEC1_BASE: 56F807 = $1250	
Decoder Control Register	(DECCR)	B-78
Decoder Control Register	(DECCR)	B-79
Filter Delay Register	(FIR)	B-80
Watchdog Timeout Register	(WTR)	B-80
Position Difference Counter Register	(POSD)	B-81
Position Difference Hold Register	(POSDH)	B-81
Revolution Counter Register	(REV)	B-82
Revolution Hold Register	(REVH)	B-82
Upper Position Counter Register	(UPOS)	B-83
Lower Position Counter Register	(LPOS)	B-83
Upper Position Hold Register	(UPOSH)	B-84

Table A-8. List of Programmer's Sheets (Continued)

Register Type	Register	Page/ Figure
Lower Position Hold Register	(LPOSH)	B-84
Upper Initialization Register	(UIR)	B-85
Lower Initialization Register	(LIR)	B-85
Input Monitor Register	(IMR)	B-86
PWM		
arch.h: ArchIO.PwmA, registers.h: ArchIO_PwmA	PWMA_BASE: 56F801/803/805 = $0E00	
	PWMA_BASE: 56F807 = $1200	
arch.h: ArchIO.PwmB, registers.h: ArchIO_PwmB	PWMB_BASE: 56F801/803/805 = $0E20	
	PWMB_BASE: 56F807 = $1220	
PWM Control Register	(PMCTL)	B-87
PWM Fault Control Register	(PMFCTL)	B-88
PWM Fault Status & Acknowledge Register	(PMFSA)	B-89
PWM Output Control Register	(PMOUT)	B-90
PWM Counter Register	(PMCNT)	B-91
PWM Counter Modulo Register	(PWMCM)	B-91
PWM Value Registers	(PWMVAL0-5)	B-92
PWM Deadtime Register	(PMDEADTM)	B-93
PWM Disable Mapping Register 1	(PMDISMAP1)	B-94
PWM Disable Mapping Register 2	(PMDISMAP2)	B-94
PWM Config Register	(PMCFG)	B-95
PWM Channel Control Register	(PMCCR)	B-96
PWM Port Register	(PMPORT)	B-97
SCI		
arch.h: ArchIO.Sci0, registers.h: ArchIO_Sci0	SCI0_BASE: 56F801/803/805 = $0F00	
	SCI0_BASE: 56F807 = $1300	
arch.h: ArchIO.Sci1, registers.h: ArchIO_Sci1	SCI1_BASE: 56F801/803/805 = $0F10	
	SCI1_BASE: 56F807 = $1310	
SCI Baud Rate Register	(SCIBR)	B-98
SCI Control Register	(SCICR)	B-99
SCI Control Register	(SCICR)	B-100
SCI Status Register	(SCISR)	B-101
SCI Data Register	(SCIDR)	B-102
SPI		
arch.h: ArchIO.Spi, registers.h: ArchIO_Spi	SPI _BASE: 56F801/803/805 = $0F20	
	SPI _BASE: 56F807 = $1320	
SPI Status and Control Register	(SPSCR)	B-103
SPI Status and Control Register	(SPSCR)	B-104
SPI Data Size Register	(SPDSR)	B-105
SPI Data Receive Register	(SPDRR)	B-106

Table A-8. List of Programmer's Sheets (Continued)

Register Type	Register	Page/ Figure
SPI Data Transmit Register	(SPDTR)	B-107
TMR		
arch.h: ArchIO.TimerA, registers.h:ArchIO_TimerA	TMRA_BASE: 56F801/803/805 = $0D00	
	TMRA_BASE: 56F807 = $1100	
arch.h: ArchIO.TimerB, registers.h:ArchIO_TimerB	TMRB_BASE: 56F801/803/805 = $0D20	
	TMRB_BASE: 56F807 = $1120	
arch.h: ArchIO.TimerC, registers.h:ArchIO_TimerC	TMRC_BASE: 56F801/803/805 = $0D40	
	TMRC_BASE: 56F807 = $1140	
arch.h: ArchIO.TimerD, registers.h:ArchIO_TimerD	TMRD_BASE: 56F801/803/805 = $0D60	
	TMRD_BASE: 56F807 = $1160	
TMR Control Register	(CTRL)	B-108
TMR Control Register	(CTRL)	B-109
TMR Control Register	(CTRL)	B-110
TMR Status and Control Register	(SCR)	B-111
TMR Status and Control Register	(SCR)	B-112
TMR Status and Control Register	(SCR)	B-113
TMR Compare Register #1	(CMP1)	B-114
TMR Compare Register #2	(CMP2)	B-115
TMR Capture Register	(CAP)	B-116
TMR Load Register	(LOAD)	B-117
TMR Hold Register	(HOLD)	B-118
TMR Counter Register	(CNTR)	B-119
OCCS		
arch.h: ArchIO.Pll, registers.h: ArchIO_Pll	CLKGEN_BASE: 56F801/803/805 = $0FA0	
	CLKGEN_BASE: 56F807 = $13A0	
PLL Control Register	(PLLCR)	B-120
PLL Control Register	(PLLCR)	B-120
PLL Divide-By Register	(PLLDB)	B-121
PLL Status Register	(PLLSR)	B-122
PLL Status Register	(PLLSR)	B-122
CLKO Select Register	(CLKOSR)	B-123
56F801 Internal Oscillator Control Register	(IOSCTL)	B-124
COP		
arch.h: ArchIO.Cop, registers.h: ArchIO_Cop	COP_BASE: 56F801/803/805 = $0F30	
	COP_BASE: 56F807 = $1330	
COP Control Register	(COPCTL)	B-125
COP Timeout Register	(COPTO)	B-126
COP Service Register	(COPSRV)	B-126
SIM		

Table A-8. List of Programmer's Sheets (Continued)

Register Type	Register	Page/ Figure
arch.h: ArchIO.Sim, registers.h: ArchIO_Sim	SYS_BASE: 56F801/803/805 = $0C00	
	SYS_BASE: 56F807 = $1000	
SIM Register	(SYS_CNTL)	B-127
SIM Register	(SYS_CNTL)	B-127
System Status Register	(SYS_STS)	B-128
Most Significant Half of JTAG_ID	(MSH_ID)	B-129
Least Significant Half of JTAG_ID	(LSH_ID)	B-129
CPU Register		
Operating Mode Register	(OMR)	B-20
Status Register	(SR) Fig. 5-4 on Pg 5-7 in the DSP56800 Family Manual, DSP56800FM	

Application: _____ Date: _____

_____ Programmer: _____

MEMORY

Bus Control Register (BCR)

arch. h: ArchCore.BusControlReg
registers. h: ArchCore_BusControlReg

Drive Control	DRV
External memory port pins are only actively driven during external access	0
External memory port pins are actively driven all the time	1

Wait State P Memory (WSPM[3:0])		
Bit String	Hex Value	Number of Wait States
0000	0	0
0100	4	4
1000	8	8
1100	C	12
All Others		Illegal

Wait State Data Memory (WSX[3:0])		
Bit String	Hex Value	Number of Wait States
0000	0	0
0100	4	4
1000	8	8
1100	C	12
All Others		Illegal

Bus Control Register (BCR) $FFF9	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	0	0	DRV	0	WAIT STATE FIELD for EXTERNAL X-MEMORY				WAIT STATE FIELD for EXTERNAL P-MEMORY			
	Write																
	Reset	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0

Reserved Bits

Application: _____ Date: _____

_____ Programmer: _____

MEMORY

Operating Mode Register (OMR)

arch. h: OMR
registers. h: OMR

Nested Looping	NL
Nested Looping Not Allowed	0
Nested Looping Allowed	1

SA	Saturation
0	Saturation Mode Disabled
1	Saturation Mode Enabled

Condition Code	CC
Codes not set in CCR Register	0
Codes set in CCR Register	1

EX	External X Memory
0	External X Memory Disabled
1	External X Memory Enabled

Stop Delay	SD
Stop Delay Enabled	0
Stop Delay Disabled	1

MB	MA	Operating Mode
0	0	Mode 0 - Single Chip
0	1	Mode 1 - Not Supported
1	0	Mode 2 - Not Supported
1	1	Mode 3 - External Memroy

Rounding	R
Rounding Not Allowed	0
Rounding Allowed	1

Operating Mode Register (OMR) (CPU Register)

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	NL	0	0	0	0	0	0	CC	0	SD	R	SA	EX	0	MB	MA
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Looping Status		
DO Loop Status	NL In OMR	LF in SR
No DO Loops active	0	0
Single DO loop active	0	1
Caution—Illegal combination. A hardware stack overflow interrupt will occur.	1	0
Two DO loops active	1	1

Reserved Bits

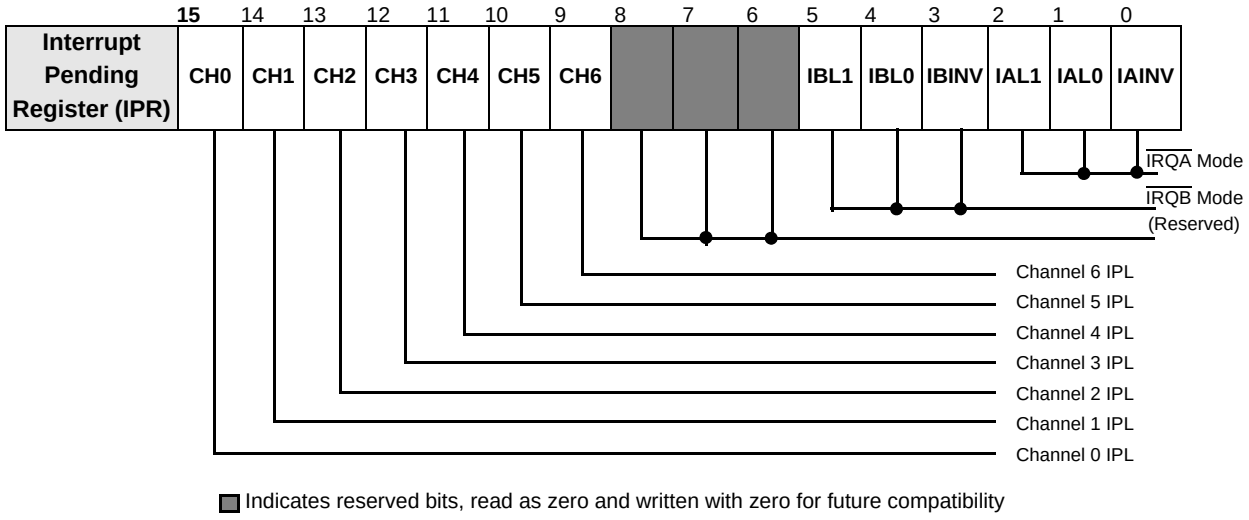
Application: _____ Date: _____

_____ Programmer: _____

Sheet 1 of 5

ITCN

Interrupt Priority Register (IPR)



Indicates reserved bits, read as zero and written with zero for future compatibility

ITCN

Group Priority Registers 0–3 (GPR0–3)

arch. h: ArchIO.IntController.GroupPriorityReg[0-3]

registers. h: ArchIO_InController_GroupPriorityReg+0 *thru* +3
ArchIO_InController_GroupPriorityReg_00 *thru*_03

PRL0–PRL63	Priority Level
0	Disabled
1–7	1 = Lowest Priority Level 7 = Highest Priority Level

Group Priority Register (GPR2) ITCN_BASE+$2

Boot Flash Interface

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read		PLR11				PLR10										
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Group Priority Register (GPR3) ITCN_BASE+$3

MSCAN Receiver Ready; MSCAN Transmitter Ready; Data Flash Interface; Program Flash Interface

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read		PLR15				PLR14				PLR13				PLR12		
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reserved Bits

ITCN

Group Priority Registers 4–7 (GPR4–7)

arch. h: ArchIO.IntController.GroupPriorityReg[4-7]
registers. h: ArchIO_IntController_GroupPriorityReg+4 *thru* +7
ArchIO_IntController_GroupPriorityReg_04 *thru* _07

PRL0–PRL63	Priority Level
0	Disabled
1–7	1 = Lowest Priority Level 7 = Highest Priority Level

Group Priority Register (GPR4) ITCN_BASE+$4

	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
			GPIO E								MSCAN Walk-up				MSCAN Error		
	Read		PLR19				PLR18				PLR17				PLR16		
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Group Priority Register (GPR5) ITCN_BASE+$5

	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
			GPIO A				GPIO B								GPIO D		
	Read		PLR23				PLR22				PLR21				PLR20		
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Group Priority Register (GPR6) ITCN_BASE+$6

	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
			Quadrature Decoder #1 INDEX Pulse				Quadrature Decoder 1 Home Switch or WD				SPI Receiver Full and/or Error				SPI Transmitter Empty		
	Read		PLR27				PLR26				PLR25				PLR24		
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Group Priority Register (GPR7) ITCN_BASE+$7

	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
			Timer D Channel 1				Timer D Channel 0				Quadrature Decoder 0 INDEX Pulse				Quadrature Decoder 0 Home Switch or WD		
	Read		PLR31				PLR30				PLR29				PLR28		
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reserved Bits

ITCN

Group Priority Registers 8–11 (GPR8–11)

arch. h: ArchIO.IntController.GroupPriorityReg [8-11]
registers. h: ArchIO_IntController_GroupPriorityReg_08 *thru* _11
ArchIO_IntController_GroupPriorityReg_08 *thru* _11

PRL0–PRL63	Priority Level
0	Disabled
1–7	1 = Lowest Priority Level 7 = Highest Priority Level

Group Priority Register (GPR8) ITCN_BASE+$8

	Timer C Channel 1				Timer C Channel 0				Timer D Channel 3				Timer D Channel 2			
Bits	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Read		PLR35				PLR34				PLR33				PLR32		
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Group Priority Register (GPR9) ITCN_BASE+$9

	Timer B Channel 1				Timer B Channel 0				Timer C Channel 3				Timer C Channel 2			
Bits	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Read		PLR39				PLR38				PLR37				PLR36		
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Group Priority Register (GPR10) ITCN_BASE+$A

	Timer A Channel 1				Timer A Channel 0				Timer B Channel 3				Timer B Channel 2			
Bits	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Read		PLR43				PLR42				PLR41				PLR40		
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Group Priority Register (GPR11) ITCN_BASE+$B

	SCI #1 Transmitter Ready				SCI #1 Transmitter Complete				Timer A Channel 3				Timer A Channel 2			
Bits	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Read		PLR47				PLR46				PLR45				PLR44		
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ITCN

Group Priority Registers 12–15 (GPR12–15)

arch. h: ArchIO.IntController.GroupPriorityReg [12-15]
registers. h: ArchIO_IntController_GroupPriorityReg +12 *thru* +15
ArchIO_IntController_GroupPriorityReg_12 *thru* _15

PRL0–PRL63	Priority Level
0	Disabled
1–7	1 = Lowest Priority Level 7 = Highest Priority Level

Group Priority Register (GPR12) ITCN_BASE+$C

	SCI #0 Transmitter Ready				SCI #0 Transmit Complete				SCI #1 Receiver Full				SCI #1 Receiver Error			
Bits	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Read		PLR51				PLR50				PLR49				PLR48		
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Group Priority Register (GPR13) ITCN_BASE+$D

	ADC A Zero Crossing or Limit Error				ADC B Conversion Complete				SCI #0 Receiver Full				SCI #0 Reciever Error			
Bits	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Read		PLR55				PLR54				PLR53				PLR52		
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Group Priority Register (GPR14) ITCN_BASE+$E

	Reload PWM A				Reload PWM B				ADC A zero Crossing or Limit Error				ADC B Zero Crossing or Limit Error			
Bits	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Read		PLR59				PLR58				PLR57				PLR56		
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Group Priority Register (GPR15) ITCN_BASE+$F

	Low Voltage Detector				PLL Loss of Lock				PWM A Fault				PWM B Fault			
Bits	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Read		PLR63				PLR62				PLR61				PLR60		
Write																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Application: _____ Date: _____

_____ Programmer: _____

FLASH

Flash Control Register (FIU_CNTL)

arch. h: ArchIO.ProgramFlash.ControlReg
registers. h: ArchIO_ProgramFlash_ControlReg

Also:
ArchIO.DataFlash / ArchIO.BootFlash
ArchIO_DataFlash / ArchIO_BootFlash

Busy	BUSY
Disabled	0
Write Inhibited to FIU Registers– Flash program/erase operation in progress.	1

Information Block Enable	IFREN
Disabled	0
Enabled	1

X Address Enable	XE
Disabled	0
Enabled	1

Y Address Enable	YE
Disabled	0
Enabled	1

PROG	Program Cycle Enable
0	Disabled
1	Enabled

ERASE	Erase Cycle Enable
0	Disabled
1	Enabled

MAS1	Mass Erase Cycle Enable
0	Disabled
1	Enabled

NVSTR	Non-Volatile Store Cycle Enable
0	Disabled
1	Enabled

Flash Control Register (FIU_CNTL) X:FIU_BASE+$0	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	BUSY	0	0	0	0	0	0	0	0	IFREN	XE	YE	PROG	ERASE	MAS1	NVSTR
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reserved Bits

FLASH

Flash Program/Erase Enable Registers (FIU_PE) and (FIU_EE)

arch. h: ArchIO.ProgramFlash.ProgrammingReg
registers. h: ArchIO_ProgramFlash_ProgramReg

arch. h: ArchIO.ProgramFlash.EraseReg
registers. h: ArchIO_ProgramFlash_EraseReg

Also:

ArchIO.DataFlash / ArchIO.BootFlash
ArchIO_DataFlash / ArchIO_BootFlash

Dumb Programming	DPE
Disabled	0
Enabled	1

ROW	Row Number
0–1024	Row number currently allowed to be programmed.

Intelligent Programming	IPE
Disabled	0
Enabled	1

Flash Program Enable Register (FIU_PE) FIU_BASE+$1	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	DPE	IPE	0	0	0	0	ROW[9:0]									
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Dumb Erase	DEE
Disabled	0
Enabled	1

PAGE	Page Number
0–128	Page number currently allowed to be erased.

Intelligent Erase	IEE
Disabled	0
Enabled	1

Flash Erase Enable Register (FIU_EE) FIU_BASE+$2	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	DEE	IEE	0	0	0	0	0	0	0	PAGE[6:0]						
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reserved Bits

FLASH

Flash Address/Data Registers (FIU_ADDR) and (FIU_DATA)

arch. h: ArchIO.ProgramFlash.AddressReg
registers. h: ArchIO_ProgramFlash_AddressReg

arch. h: ArchIOProgramFlash.DataReg
registers. h: ArchIO_ProgramFlash_Data
Also:

ArchIO.DataFlash / ArchIO.BootFlash
ArchIO_DataFlash / ArchIO_BootFlash

Note: This register is designed for program development and error analysis.

A	Current Program Address or Erase Address
$0000	This register is cleared upon any system reset.
Program Address	This register is set to the program/erase address by attempting to write to memory space occupied by flash memory.

Flash Address Register (FIU_ADDR) FIU_BASE+$3	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	A[15:0]															
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Note: This register is designed for program development and error analysis.

D	Last Value Programmed or Value Written to Initiate an Erase
$0000	This register is cleared upon any system reset.
Program Data	Writing to Flash memory space sets this register to the program data value.

Flash Data Register (FIU_DATA) FIU_BASE+$4	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	D[15:0]															
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reserved Bits

FLASH

Flash Interrupt Enable Register (FIU_IE)

arch. h: ArchIO.ProgramFlash.IntReg
registers. h: ArchIO_ProgramFlash_IntReg

Also:
ArchIO.DataFlash / ArchIO.BootFlash
ArchIO_DataFlash / ArchIO_BootFlash

IE	Interrupt Enable Bits	
0	Interrupt Bit Disabled	
1	Bit	Interrupt Bit Enabled
	11	Write to Register Attempt–Busy
	10	Write to FIU_CNTL During Program Erase
	9	Terase or Tmel Timeout
	8	Trcv Timeout
	7	Tprog Timeout
	6	Tpgs Timeout
	5	Tnvh1 Timeout
	4	Tnvh Timeout
	3	Tnvs Timeout
	2	Illegal Flash Read/Write Access Attempt During Erase
	1	Illegal Flash Read/Write Access Attempt During Program
	0	Flash Access Out-Of-Range

Flash Interrupt Enable Register (FIU_IE) FIU_BASE+$5	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	IE[11:0]											
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reserved Bits

FLASH

Flash Interrupt Source Register (FIU_IS)

arch. h: ArchIO.ProgramFlash.IntSourceReg
registers. h: ArchIO_ProgramFlash_IntSourceReg

Also:
ArchIO.DataFlash / ArchIO.BootFlash
ArchIO_DataFlash / ArchIO_BootFlash

IS	Interrupt Source Bits	
0	Interrupt Source Bit Inactive	
1	Bit	Interrupt Source Bit Active: Error Detected
	11	Write to Register Attempt while Busy
	10	Write to FIU_CNTL During Program Erase
	Bit	Interrupt Source Bit Active: Timeout Condition Met
	9	Terase or Tmel Timeout
	8	Trcv Timeout
	7	Tprog Timeout
	6	Tpgs Timeout
	5	Tnvh1 Timeout
	4	Tnvh Timeout
	3	Tnvs Timeout
	Bit	Interrupt Source Bit Active: Error Detected
	2	Illegal Flash Read/Write Access Attempt During Erase
	1	Illegal Flash Read/Write Access Attempt During Program
	0	Flash Access Out-Of-Range

Flash Interrupt Source Register (FIU_IS) FIU_BASE+$6	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	IS[11:0]											
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reserved Bits

FLASH

Flash Interrupt Pending Register (FIU_IP)

arch. h: ArchIO.ProgramFlash.IntPendingReg
registers. h: ArchIO_ProgramFlash_IntPendingReg

Also:

ArchIO.DataFlash / ArchIO.BootFlash
Arch**IO_DataFlash / ArchIO_BootFlash**

Note: Use this register for indirectly controlling the interrupt service routine.

IP	Interrupt Pending Bits	
0	Interrupt Pending Bit Inactive	
1	Bit	Interrupt Pending Bit Active
	11	Write to Register Attempt–Busy
	10	Write to FIU_CNTL During Program Erase
	9	Terase or Tmel Timeout
	8	Trcv Timeout
	7	Tprog Timeout
	6	Tpgs Timeout
	5	Tnvh1 Timeout
	4	Tnvh Timeout
	3	Tnvs Timeout
	2	Illegal Flash Read/Write Access Attempt During Erase
	1	Illegal Flash Read/Write Access Attempt During Program
	0	Flash Access Out-Of-Range

Flash Interrupt Pending Register (FIU_IP) FIU_BASE+$7	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	IP[11:0]											
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reserved Bits

FLASH

Flash Clock Divisor Register (FIU_CKDIVISOR)

Flash Terase Limit Register (FIU_TERASEL)

Flash Tme Limit Register (FIU_TMEL)

arch. h: ArchIOProgramFlash.DivisorReg
registers. h: ArchIOProgramFlash_DivisorReg

arch. h: ArchIO.ProgramFlash.TimerEraseReg
registers. h: ArchIO_ProgramFlash_TimerEraseReg

arch. h: ArchIO.ProgramFlash.TimerMassEraseReg
registers. h: ArchIO_ProgramFlash_TimerMassEraseReg
Also:

ArchIO.DataFlash / ArchIO.BootFlash
ArchIO_DataFlash / ArchIO_BootFlash

N	Clock Divisor
0–15	Value of Clock Divisor = $2^{(N+1)}$ (Values = 2, 4, 8, 16, . . . 65536)

Flash Clock Divisor Register (FIU_CKDIVISOR) FIU_BASE+$8	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	0	0	0	0	0	0	0	0	N[3:0]			
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

TERASEL	Timer Erase Limit
0–127	Terase = $2^{(N+1)}$ x (TERASEL + 1) where N = the Clock Divisior Value Default Erase Time = 2^{16} x 2^4 x 25 ns = 26.2 ms

Flash Terase Limit Register (FIU_TERASEL) FIU_BASE+$9	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	0	0	0	0	0	TERASEL[6:0]						
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

TMEL	Timer Mass Erase Limit
0–255	Tme = $2^{(N+1)}$ x (TMEL + 1) where N = the Clock Divisior Value Default Erase Time = 2^{16} x 2^4 x 25 ns = 26.2 ms

Flash Tme Limit Register (FIU_TMEL) FIU_BASE+$A	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	0	0	0	0	TMEL[7:0]							
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

Reserved Bits

FLASH

Flash Tnvs Limit Register (FIU_TNVSL)

Flash Tpgs Limit Register (FIU_TPGSL)

Flash Tprog Limit Register (FIU_TPROGL)

arch.h: ArchIO.ProgramFlash.TimerNVSStorageReg
registers.h: ArchIO_ProgramFlash_TimerNVSStorageReg

arch.h: ArchIO.ProgramFlash.TimerProgramSetupReg
registers.h: ArchIO_Program_FlashTimerProgramSetupReg

arch.h: ArchIO.ProgramFlash.TimerProgramReg
register.h: ArchIO_ProgramFlash_TimerProgramReg
Also:
ArchIO.DataFlash / ArchIO_DataFlash
ArchIO.BootFlash / ArchIO_BootFlash

TNVSL	Timer Non-Volatile Storage Limit
0–2047	Tnvs = (TNVSL + 1) Tnvs default value = 25 nsec x 2^8 = 6.4 μs

Flash Tnvs Limit Register (FIU_TNVSL) FIU_BASE+$B	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	0	TNVSL[10:0]										
	Write																
	Reset	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

TPGSL	Timer Erase Limit
0–4095	Tpgs = (TPGSL + 1) Tpgs default value = 25 nsec x 2^9 = 12.8 μs

Flash Tpgs Limit Register (FIU_TPGSL) FIU_BASE+$C	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	TPGSL[11:0]											
	Write																
	Reset	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1

TPROGL	Timer Program Limit
0–16383	Tprog = (TPROGL + 1) Default Program Time Limit = 2^{10} x 25 ns = 25.6 μsec.

Flash Tprog Limit Register (FIU_TPROGL) FIU_BASE+$D	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	TPROGL[13:0]													
	Write																
	Reset	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1

Reserved Bits

FLASH

Flash Tnvh Limit Register (FIU_TNVHL)

Flash Tnvh1 Limit Register (FIU_TNVH1L)

Flash Trcv Limit Register (FIU_TRCVL)

arch.h: ArchIO.ProgramFlash.TimerNVHoldReg
registers.h: ArchIO_ProgramFlash_TimerNVHoldReg

arch.h: ArchIO.ProgramFlash.TimerNVHold1Reg
registers.h: ArchIO_ProgramFlash_TimerNVHold1Reg

arch.h: ArchIO.ProgramFlash.TimerRecoveryReg
registers.h: ArchIO_ProgramFlash_TimerRecoverReg

Also:

ArchIO.DataFlash / ArchIO_DataFlash
ArchIO.BootFlash / ArchIO_BootFlash

TNVHL	Timer Non-Volatile Hold Limit
0–2047	Tnvh = (TNVHL + 1) Default Tnvh Value = 2^8 x 25 ns = 6.4 μs

Flash Tnvh Limit Register (FIU_TNVHL) FIU_BASE+$E	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	0	TNVHL[10:0]										
	Write																
	Reset	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

TNVH1L	Timer Non-Volatile Hold 1 Limit
0–32767	Tnvh1 = (TNVH1L + 1) Tnvh1 Default Value = 2^{12} x 25 ns = 102.4 μs

Flash Tnvh1 Limit Register (FIU_TNVH1L) FIU_BASE+$F	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	TNVH1L[14:0]														
	Write																
	Reset	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1

TRCVL	Timer Recovery Limit
0–511	Trcv = (TRCVL + 1) Trcv Default Value = 2^6 x 25 ns = 1.6 μs

Flash Trcv Limit Register (FIU_TRCVL) FIU_BASE+$10	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	0	0	0	TRCVL[8:0]								
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1

Reserved Bits

Application: _____ Date: _____

Programmer: _____

GPIO

arch.h: ArchIO.Port **?**. PullUpReg
registers. h: ArchIO_Port **?**_PullUpReg
arch.h: ArchIO.Port **?.**DataReg
registers.h: ArchIO_Port **?**_DataReg

Where **?** = A, B, D, *or* E

Pull-Up Enable Register (PUR)

Data Register (DR)

Pull-Up Enable Functions					
Peripheral Out Enable	PER	PUR	DDR	GPIO Pin State	GPIO Pin Pull-Up
x	0	0	0	Input	Disabled
x	0	0	1	Output	Disabled
x	0	1	0	Input	Enabled
x	0	1	1	Output	Disabled
0	1	x	x	Output	Disabled
0	1	x	x	Output	Disabled
1	1	x	x	Input	Disabled
1	1	x	x	Input	Enabled

PU	Pull-Up
0	Pull-Up Disabled
1	DDR and PER = 0: Pull-Up is enabled. Each bit performs the pull-up function on the corresponding GPIO pin if the GPIO is configured as an input. PER = 1: Pull-Up is controlled by the peripheral output enable and the PUR.
Note 1:	If the GPIO pin is configured as an output, the PUR is not recognized.
Note 2:	PUR is set to 1 on processor reset.
Note 3:	Refer to the above *Pull-Up Enable Functions* table for all possible combinations.

Pull-Up Enable Register (PUR) GPIO_BASE+$0	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	0	0	0	0	PU[7:0]							
	Write																
	Reset	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

D
This GPIO Pin / IPBus interface holds data coming from the GPIO pin or the IPBus.

Data Register (DR) GPIO:_BASE+$1	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	0	0	0	0	D[7:0]							
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reserved Bits

GPIO

Data Direction Register (DDR)

arch.h: ArchIO.Port ***?***. DataDirectionReg
registers.h: ArchIO_Port ***?***_DataDirectionReg

Where ***?*** = A, B, D, *or* E

Pull-Up Enable Functions					
Peripheral Out Enable	PER	PUR	DDR	GPIO Pin State	GPIO Pin Pull-Up
x	0	0	0	Input	Disabled
x	0	0	1	Output	Disabled
x	0	1	0	Input	Enabled
x	0	1	1	Output	Disabled
0	1	x	x	Output	Disabled
0	1	x	x	Output	Disabled
1	1	x	x	Input	Disabled
1	1	x	x	Input	Enabled

DD	Data Direction
0	PUR = 1: GPIO pin is an input with pull-up device. PUR = 0: GPIO pin is an input without pull-up device.
1	The GPIO pin is an output.
Note 1:	Refer to the above *Pull-Up Enable Functions* table for all possible combinations on using the DDR register.
Note 2:	Each bit performs the pull-up function on the corresponding GPIO pin.

Data Direction Register (DDR) GPIO_BASE+$2	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	0	0	0	0	DD[7:0]							
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reserved Bits

GPIO

Peripheral Enable Register (PER)

arch.h: ArchIO.Port ***?***. PeripheralReg
registers.h: ArchIO_Port ***?***_PeripheralReg

Where ***?*** = A, B, D, *or* E

Pull-Up Enable Functions					
Peripheral Out Enable	PER	PUR	DDR	GPIO Pin State	GPIO Pin Pull-Up
x	0	0	0	Input	Disabled
x	0	0	1	Output	Disabled
x	0	1	0	Input	Enabled
x	0	1	1	Output	Disabled
0	1	x	x	Output	Disabled
0	1	x	x	Output	Disabled
1	1	x	x	Input	Disabled
1	1	x	x	Input	Enabled

PE	Peripheral Enable
0	GPIO masters the pin. The DDR determines the data direction flow: DDR = 0: GPIO pin is input only DDR = 1: GPIO pin is output only
1	Peripheral masters the pin, configuring it as: – A required input (with or without pull-up), or – A required output (depending on peripheral output enable status, and includes data transfers from the GPIO pin to the peripheral.)

Note 1: Refer to the above *Pull-Up Enable Functions* table for all possible combinations on using the PER register.

Note 2: Each bit determines the peripheral function on the corresponding GPIO pin.

Peripheral Enable Register (PER) GPIO_BASE+$3	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	0	0	0	0	PE[7:0]							
	Write																
	Reset	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Reserved Bits

GPIO

Interrupt Assert Register (IAR)

Interrupt Enable Register (IENR)

arch.h: ArchIO.Port ***?***. IntAssertReg
registers.h: ArchIO_Port ***?***_IntAssertReg

arch.h: ArchIO.Port ***?***. IntEnableReg
registers.h: ArchIO_Port ***?***_IntEnableReg

Where ***?*** = A, B, D, *or* E

IA	Interrupt Assert
0	Interrupt is cleared
1	An interrupt is asserted.
Note: The IAR register is only for software testing.	

Interrupt Assert Register (IAR) GPIO:_BASE+$4	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	0	0	0	0	IA[7:0]							
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

IEN	Interrupt Enable
0	Interrupt detection disabled.
1	Interrupt detection enabled.

Interrupt Enable Register (IENR) GPIO_BASE+$5	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	0	0	0	0	IEN[7:0]							
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reserved Bits

GPIO

Interrupt Polarity Register (IPOLR)

Interrupt Pending Register (IPR)

arch.h: ArchIO.Port **?**. IntPolarityReg
registers.h: ArchIO_Port **?**_IntPolarityReg

arch.h: ArchIO.Port **?**. IntPendingReg
registers.h: ArchIO_Port **?**_IntPendingReg

Where ***?*** = A, B, D, *or* E

IPOL	Interrupt Polarity	
0	The interrupt at the GPIO pin is active high.	When register IENR = 1
1	The interrupt at the GPIO pin is active low.	
–	Register IPOLR is disabled	When register IENR = 0

Interrupt Polarity Register (IPOLR) GPIO_BASE+$6	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	0	0	0	0	IPOL[7:0]							
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

IPR	Interrupt Pending Register
0	Incoming interrupts do not exist.
1	A set bit indicates which pin caused an incoming interrupt.
Note: To Clear the IPR Register	
Software Interrupts	Write zeros into the IAR register.
External Interrupts	Write zeros into the IESR register.

Interrupt Pending Register (IPR) GPIO:_BASE+$7	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	0	0	0	0	IPR[7:0]							
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reserved Bits

GPIO

Interrupt Edge Sensitive Register (IESR)

arch.h: ArchIO.Port **?**. IntEdgeReg
registers.h: ArchIO_Port **?**_IntEdgeReg

Where **?** = A, B, D, *or* E

IES	Interrupt Edge Sensitive
0	Interrupts are not recorded
1	IESR recorded an interrupt. IESR records interrupts when the edge detector circuit detects an edge and the IENR is set to 1.

Interrupt Edge-Sensitive Register (IESR) GPIO_BASE+$8	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	0	0	0	0	IES[7:0]							
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reserved Bits

CAN

Control Register 0 (CANCTL0)

arch.h: ArchIO.CAN.ControlReg
registers.h: ArchIO_CAN_ControlReg

Received Frame Flag	RXFRM
No Valid Message Received	0
Valid Message Received	1

Receiver Active Status	RXACT
CAN is Transmitting or Idle	0
CAN is Receiving Message	1

CAN Stops in Wait Mode	CSWAI
CAN Clocking continues in Wait Mode	0
Clocking toCAN Module Stops During Wait Mode	1

CAN Bus Synchronization	SYNCH
CAN Not Synchronized to CAN Bus	0
CAN Synchronized to CAN Bus	1

SLPAK	Internal Sleep Mode
0	Wake-Up
1	CAN in Sleep Mode

SLPRQ	Sleep Request–Go Into Sleep Mode
0	Wake-Up, CAN Functions Normally
1	CAN Enters Sleep Mode

SFTRES	Soft Reset	
0	Normal Operation	
1	CAN Enters Soft Reset State	
	Exclusively when SFTRES = 1	
	Registers Entering Hard Reset State	Registers Writable by CPU
	CANCTL0	CANCTL1
	CANRFLG	CANBTR0
	CANIER	CANBTR1
	CANTFLG	CANIDAC
	CANTCR	CANIDAR0-7

CAN Control Register 0 (CANCTL0) CAN_BASE+$0	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	0	0	0	0	RXFRM	RXACT	CSWAI	SYNCH	0	SLPAK	SLPRQ	SFTRES
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Reserved Bits

CAN

Control Register 1 (CANCTL1)

arch.h: ArchIO.CAN.Control1Reg
registers.h: ArchIO_CAN_Control1Reg

CAN Enable	CANE
CAN module disabled	0
CAN module enabled	1

WUPM	Wakeup Mode
0	When CAN is in sleep mode, CAN wakes up the CPU after any recessive to dominant edge on CAN bus
1	When CAN is in sleep mode, CAN wakes up the CPU when dominant bus pulse width = Twup

Loop Back Self Test Mode	LOOPB
Normal operation	0
CAN performs internal loop back	1

CLKSRC	CAN Clock Source
0	Clock source is EXTAL-CLK
1	Clock source is IPBus Clock

CAN Control Register 1 (CANCTL1) CAN_BASE+$1	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	0	0	0	0	CANE	0	0	0	0	LOOPB	WUPM	CLKSRC
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reserved Bits

CAN

Bus Timing Register 0 (CANBTR0)

arch.h: ArchIO.CAN.BusTiming0Reg
registers.h: ArchIO_CAN_BusTiming0Reg

BRP5	BRP4	BRP3	BRP2	BRP1	BRP0	Baud Rate Prescaler Value (P)
0	0	0	0	0	0	1
0	0	0	0	0	1	2
0	0	0	0	1	0	3
0	0	0	0	1	1	4
1	1	1	1	1	0	63
1	1	1	1	1	1	64

Synchronization Jump Width	SJW1	SJW0
1 Tq Clock Cycle	0	0
2 Tq Clock Cycle	0	1
3 Tq Clock Cycle	1	0
4 Tq Clock Cycle	1	1

CAN Bus Timing Register 0 (CANBTR0) CAN_BASE+$2	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	0	0	0	0	SJW1	SJW0	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reserved Bits

CAN

Bus Timing Register 1 (CANBTR1)

arch.h: ArchIO.CAN.BusTiming1Reg
registers.h: ArchIO_CAN_BusTiming1Reg

Sampling	SAMP
One sample per bit	0
Three samples per bit	1

Time Segment 2	TSEG22	TSEG21	TSEG20
1 Tq clock cycle	0	0	0
2 Tq clock cycles	0	0	1
3 Tq clock cycles	0	1	0
4 Tq clock cycles	0	1	1
5 Tq clock cycles	1	0	0
6 Tq clock cycles	1	0	1
7 Tq clock cycles	1	1	0
8 Tq clock cycles	1	1	1

TSEG13	TSEG12	TSEG11	TSEG10	Time Segment 1
0	0	0	0	Not a valid setting
0	0	0	1	Not a valid setting
0	0	1	0	Not a valid setting
0	0	1	1	4 Tq clock cycles
0	1	0	0	5 Tq clock cycles
0	1	0	1	6 Tq clock cycles
0	1	1	0	7 Tq clock cycles
0	1	1	1	8 Tq clock cycles
1	0	0	0	9 Tq clock cycles
1	0	0	1	10 Tq clock cycles
1	0	1	0	11 Tq clock cycles
1	0	1	1	12 Tq clock cycles
1	1	0	0	13 Tq clock cycles
1	1	0	1	14 Tq clock cycles
1	1	1	0	15 Tq clock cycles
1	1	1	1	16 Tq clock cycles

CAN Bus Timing Register 1 (CANBTR1) CAN_BASE+$3	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	0	0	0	0	SAMP	TSEG22	TSEG21	TSEG20	TSEG13	TSEG12	TSEG11	TSEG10
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reserved Bits

CAN

Receiver Flag Register (CANRFLG)

arch.h: ArchIO.CAN.RxFlagReg
registers.h: ArchIO_CAN_RxFlagReg

Note: To ensure data integrity, do not read the receive buffer registers while the RXF flag is cleared.

Wakeup Interrupt Flag	WUPIF
No wake up activity observed in sleep mode.	0
CAN detected bus activity and requested wake up	1

Receiver Warning Interrupt Flag	RWRNIF
No receiver warning status received	0
CAN entered receiver warning status	1

Transmitter Warning Interrupt Flag	TWRNIF
No transmitter warning status reached	0
CAN entered transmitter warning status	1

Receiver Error Passive Interrupt Flag	RERRIF
No receiver error passive status reached	0
CAN entered receiver error passive status	1

TERRIF	Transmitter Error Passive Interrupt Flag
0	No transmitter error passive status reached
1	CAN entered transmitter error passive status

BOFFIF	Bus Off Interrupt Flag
0	No bus-off status reached
1	CAN entered bus-off status

OVRIF	Overrun Interrupt Flag
0	No data overrun condition
1	A data overrun detected

RXF	Receiver Buffer Full
0	Receive buffer is released (not full)
1	Receiver buffer is full and a new message is available

CAN Receiver Flag Register (CANRFLG) CAN_BASE+$4	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	0	0	0	0	WUPIF	RWRNIF	TWRNIF	RERRIF	TERRIF	BOFFIF	OVRIF	RXF
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reserved Bits

CAN

Receiver Interrupt Enable Register (CANRIER)

arch.h: ArchIO.CAN.RxIntEnableReg
registers.h: ArchIO_CAN_RxIntEnableReg

Wakeup Interrupt Enable	WUPIE
No interrupt request is generated from this event	0
Wake up event	1

Receiver Warning Interrupt Enable	RWRNIE
No interrupt request is generated from this event	0
Receiver warning status event	1

Transmitter Warning Interrupt Enable	TWRNIE
No interrupt request is generated from this event	0
Transmitter warning status event	1

Receiver Error Passive Interrupt Enable	RERRIE
No interrupt request is generated from this event	0
Receiver error passive status event	1

TERRIE	Transmitter Error Passive Interrupt Enable
0	No interrupt request is generated from this event
1	Transmitter error passive status event

BOFFIE	Bus Off Interrupt Enable
0	No interrupt request is generated from this event
1	Bus off event

OVRIE	Overrun Interrupt Enable
0	No interrupt request is generated from this event
1	Overrun event

RXFIE	Receiver Full Interrupt Enable
0	No interrupt request is generated from this event
1	Receive buffer full event

CAN Receiver Interrupt Enable Register (CANRIER) CAN_BASE+$5	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	0	0	0	0	WUPIE	RWRNIE	TWRNIE	RERRIE	TERRIE	BOFFIE	OVRIE	RXFIE
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reserved Bits

CAN

Transmitter Flag Register (CANTFLG)

Transmitter Control Register (CANTCR)

arch.h: ArchIO.CAN.TxFlagReg
registers.h: ArchIO_CAN_TxFlagReg

arch.h: ArchIO.CAN.TxControlReg
registers.h: ArchIO_CAN_TxControlReg

Note: To ensure data integrity, do not write to the transmit buffer registers while the TXE[2:0] flag is cleared.

Abort Acknowledge	ABTAK
Message sent. Not aborted	0
Message aborted	1

TXE	Transmitter Buffer Empty
0	Transmit message buffer is full
1	Transmit message buffer is empty (not scheduled)

CAN Transmitter Flag Register (CANTFLG) CAN_BASE+$6	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	0	0	0	0	0	ABTAK2	ABTAK1	ABTAK0	0	TXE2	TXE1	TXE0
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Abort Request	ABTRQ
No abort request	0
Abort request pending	1
Note: The software must not clear one or more of the TXE[2:0] flags in CANTFLG and simultaneously set the respective ABTRQ[2:0] bit(s).	

TXEIE	Transmitter Empty Interrupt Enable
0	No interrupt request generated from this event
1	Transmitter empty interrupt request

CAN Transmitter Control Register (CANTCR) CAN_BASE+$7	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	0	0	0	0	0	ABTRQ2	ABTRQ1	ABTRQ0	0	TXEIE2	TXEIE1	TXEIE0
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reserved Bits

CAN

Identifier Acceptance Control Register (CANIDAC)

arch.h: ArchIO.CAN.IdControlReg
registers.h: ArchIO_CAN_IdControlReg

Identifier Acceptance Mode	IDAM1	IDAM0
Two 32-bit Acceptance Filters	0	0
Four 16-bit Acceptance Filters	0	1
Eight 8-bit Acceptance Filters	1	0
Filter Closed	1	1

IDHIT2	IDHIT1	IDHIT0	Identifier Acceptance Hit
0	0	0	Filter 0 Hit
0	0	1	Filter 1 Hit
0	1	0	Filter 2 Hit
0	1	1	Filter 3 Hit
1	0	0	Filter 4 Hit
1	0	1	Filter 5 Hit
1	1	0	Filter 6 Hit
1	1	1	Filter 7 Hit

CAN Identifier Acceptance Control Register (CANIDAC) CAN_BASE+$8	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	0	0	0	0	0	0	IDAM1	IDAM0	0	IDHIT2	IDHIT1	IDHIT0
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reserved Bits

CAN

Receive/Transmit Error Counter Register (CANRXERR)

arch.h: ArchIO.CAN.RxErrorCountReg
registers.h: ArchIO_CAN_RxErrorCountReg

arch.h: ArchIO.CAN.TxErrorCountReg
registers.h: ArchIO_CAN_TxErrorCountReg

CAN Receive Error Counter Status (RXERR[7:0])

Note: Reading this register in any mode other than Sleep or Soft Reset may return an incorrect value.

CAN Receive Error Counter Reg. (CANRXERR) CAN_BASE+$E	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	0	0	0	0	RXERR7	RXERR6	RXERR5	RXERR4	RXERR3	RXERR2	RXERR1	RXERR0
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CAN Transmit Error Counter Status (TXERR[7:0])

Note: Reading this register in any mode other than Sleep or Soft Reset may return an incorrect value.

CAN Transmit Error Counter Reg. (CANTXERR) CAN_BASE+$F	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	0	0	0	0	TXERR7	TXERR6	TXERR5	TXERR4	TXERR3	TXERR2	TXERR1	TXERR0
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reserved Bits

CAN

Identifier Acceptance Registers–1st Bank (CANIDAR0–3)

arch.h: ArchIO.CAN.IdAcceptReg ***?***
registers.h: ArchIO_CAN_IdAcceptReg ***?***

Where ***?*** = 4, 5, 6, 7

Acceptance Code (AC[7:0])
A user defined bit sequence which is compared with the related identifier register (IDR0–IDR3) of the Receive Message buffer

CAN Identifier Acceptance Register (CANIDAR0) CAN_BASE+$10	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
	Write									
	Reset	0	0	0	0	0	0	0	0	0

CAN Identifier Acceptance Register (CANIDAR1) CAN_BASE+$11	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
	Write									
	Reset	0	0	0	0	0	0	0	0	0

CAN Identifier Acceptance Register (CANIDAR2) CAN_BASE+$12	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
	Write									
	Reset	0	0	0	0	0	0	0	0	0

CAN Identifier Acceptance Register (CANIDAR3) CAN_BASE+$13	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
	Write									
	Reset	0	0	0	0	0	0	0	0	0

Reserved Bits

CAN

Identifier Acceptance Registers–2nd Bank (CANIDAR4–7)

arch.h: ArchIO.CAN.IdAcceptReg ***?***
registers.h: ArchIO_CAN_IdAcceptReg ***?***

Where ***?*** = 4, 5, 6, 7

Acceptance Code (AC[7:0])
A user defined bit sequence which is compared with the related identifier register (IDR0–IDR3) of the Receive Message buffer

CAN Identifier Acceptance Register (CANIDAR4) CAN_BASE+$18	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
	Write									
	Reset	0	0	0	0	0	0	0	0	0

CAN Identifier Acceptance Register (CANIDAR5) CAN_BASE+$19	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
	Write									
	Reset	0	0	0	0	0	0	0	0	0

CAN Identifier Acceptance Register (CANIDAR6) CAN_BASE+$1A	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
	Write									
	Reset	0	0	0	0	0	0	0	0	0

CAN Identifier Acceptance Register (CANIDAR7) CAN_BASE+$1B	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
	Write									
	Reset	0	0	0	0	0	0	0	0	0

Reserved Bits

CAN

Identifier Mask Registers–1st Bank (CANIDMR0–3)

arch.h: ArchIO.CAN.IdMaskReg ***?***
registers.h: ArchIO_CAN_IdMaskReg ***?***

Where ***?*** = 0, 1, 2, 3

AM	Acceptance Mask
0	Match corresponding acceptance code register and identifier bits
1	Ignore corresponding acceptance code register bit

CAN Identifier Mask Register (CANIDMR0) CAN_BASE+$14	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
	Write									
	Reset	0	0	0	0	0	0	0	0	0

CAN Identifier Mask Register (CANIDMR1) CAN_BASE+$15	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
	Write									
	Reset	0	0	0	0	0	0	0	0	0

CAN Identifier Mask Register (CANIDMR2) CAN_BASE+$16	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
	Write									
	Reset	0	0	0	0	0	0	0	0	0

CAN Identifier Mask Register (CANIDMR3) CAN_BASE+$17	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
	Write									
	Reset	0	0	0	0	0	0	0	0	0

Mandatory Programming for Standard Identifier

32-Bit Filter Mode: To receive standard identifiers in 32-bit filter mode, it is necessary to program the last three bits (AM[2:0]) in the mask registers CANIDMR1 and CANIDMR5 as 111.

16-Bit Filter Mode: To receive standard identifiers in 16-bit filter mode, it is necessary to program the last three bits (AM[2:0]) in the mask registers CANIDMR1, CANIDMR3, CANIDMR5, and CANIDMR7 to 111.

Reserved Bits

CAN

Identifier Mask Registers–2nd Bank (CANIDMR4–7)

arch.h: ArchIO.CAN.IdMaskReg ***?***
registers.h: ArchIO_CAN_IdMaskReg ***?***

Where ***?*** = 4, 5, 6, 7

AM	Acceptance Mask
0	Match corresponding acceptance code register and identifier bits
1	Ignore corresponding acceptance code register bit

CAN Identifier Mask Register (CANIDMR4) CAN_BASE+$1C	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
	Write									
	Reset	0	0	0	0	0	0	0	0	0

CAN Identifier Mask Register (CANIDMR5) CAN_BASE+$1D	Bits	15-8	7	6	5	4	3	2	Bit 1	0
	Read	0	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
	Write									
	Reset	0	0	0	0	0	0	0	0	0

CAN Identifier Mask Register (CANIDMR6) CAN_BASE+$1E	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
	Write									
	Reset	0	0	0	0	0	0	0	0	0

CAN Identifier Mask Register (CANIDMR7) CAN_BASE+$1F	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0
	Write									
	Reset	0	0	0	0	0	0	0	0	0

Mandatory Programming for Standard Identifier

32-Bit Filter Mode: To receive standard identifiers in 32-bit filter mode, it is necessary to program the last three bits (AM[2:0]) in the mask registers CANIDMR1 and CANIDMR5 as 111.

16-Bit Filter Mode: To receive standard identifiers in 16-bit filter mode, it is necessary to program the last three bits (AM[2:0]) in the mask registers CANIDMR1, CANIDMR3, CANIDMR5, and CANIDMR7 to 111.

Reserved Bits

CAN

Standard Identifier Mapping Receive Buffer, Identifier Registers 0-3

arch.h: ArchIO.CAN.RxBuffer_IdReg ***?***
registers.h: ArchIO_CAN_RxBuffer_IdReg ***?***

Where ***?*** = 0, 1, 2, 3

ID Extended	IDE
Standard format (11-bit)	0
Extended format (29-bit)	1

RTR	Remote Transmission Request	
	Receive Buffer: Status of Received Frame	**Transmit Buffer:** Setting of RTR Bit to be Sent
0	Data frame	Data frame
1	Remote frame	Remote frame

Standard Format Identifier	ID
An identifier priority is highest for the smallest binary number	0 - 2047

Standard Mapping: CAN Receive Buffer Identifier Register 0 (CAN_RB_IDR0) CAN_BASE+$40	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3
	Write									

Standard Mapping: CAN Receive Buffer Identifier Register 1 (CAN_RB_IDR1) CAN_BASE+$41	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	ID2	ID1	ID0	RTR	IDE (=0)			
	Write									

Standard Mapping: CAN Receive Buffer Identifier Register 2 (CAN_RB_IDR2) CAN_BASE+$42	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0								
	Write									

Standard Mapping: CAN Receive Buffer Identifier Register 3 (CAN_RB_IDR3) CAN_BASE+$43	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0								
	Write									
	Reset	0	0	0	0	0	0	0	0	0

Reserved Bits

CAN

Standard Identifier Mapping Transmit Buffer 0, Identifier Registers 0-3

arch.h: ArchIO.CAN.TxBuffer[0].IdReg ***?***
registers.h: ArchIO_CAN_TxBuffer_0_IdReg ***?***

Where ***?*** = 0, 1, 2, 3

ID Extended	IDE
Standard format (11-bit)	0
Extended format (29-bit)	1

Standard Format Identifier	ID
An identifier priority is highest for the smallest binary number	0 - 2047

RTR	Remote Transmission Request	
	Receive Buffer: Status of Received Frame	**Transmit Buffer:** Setting of RTR Bit to be Sent
0	Data frame	Data frame
1	Remote frame	Remote frame

Standard Mapping: CAN Transmit Buffer 0 Identifier Register 0 (CAN_TB0_IDR0) CAN_BASE+$50	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3
	Write									

Standard Mapping: CAN Transmit Buffer 0 Identifier Register 1 (CAN_TB0_IDR1) CAN_BASE+$51	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	ID2	ID1	ID0	RTR	IDE (=0)			
	Write									

Standard Mapping: CAN Transmit Buffer 0 Identifier Register 2 (CAN_TB0_IDR2) CAN_BASE+$52	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0								
	Write									

Standard Mapping: CAN Transmit Buffer 0 Identifier Register 3 (CAN_TB0_IDR3) CAN_BASE+$53	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0								
	Write									
	Reset	0	0	0	0	0	0	0	0	0

Reserved Bits

CAN

Standard Identifier Mapping Transmit Buffer1 , Identifier Registers 0-3

arch.h: ArchIO.CAN.TxBuffer[1].IdReg ***?***
registers.h: ArchIO_CAN_TxBuffer_1_IdReg ***?***

Where ***?*** = 0, 1, 2, 3

ID Extended	IDE
Standard format (11-bit)	0
Extended format (29-bit)	1

Standard Format Identifier	ID
An identifier priority is highest for the smallest binary number	0 - 2047

RTR	Remote Transmission Request	
	Receive Buffer: Status of Received Frame	**Transmit Buffer:** Setting of RTR Bit to be Sent
0	Data frame	Data frame
1	Remote frame	Remote frame

Standard Mapping: CAN Transmit Buffer 1 Identifier Register 0 (CAN_TB1_IDR0) CAN_BASE+$60	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3
	Write									

Standard Mapping: CAN Transmit Buffer 1 Identifier Register 1 (CAN_TB1_IDR1) CAN_BASE+$61	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	ID2	ID1	ID0	RTR	IDE (=0)			
	Write									

Standard Mapping: CAN Transmit Buffer 1 Identifier Register 2 (CAN_TB1_IDR2) CAN_BASE+$62	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0								
	Write									

Standard Mapping: CAN Transmit Buffer 1 Identifier Register 3 (CAN_TB1_IDR3) CAN_BASE+$63	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0								
	Write									
	Reset	0	0	0	0	0	0	0	0	0

Reserved Bits

CAN

Standard Identifier Mapping Transmit Buffer2, Identifier Registers 0-3

arch.h: ArchIO.CAN.TxBuffer[2].IdReg ***?***
registers.h: ArchIO_CAN_TxBuffer_2_IdReg ***?***

Where ***?*** = 0, 1, 2, 3

Standard Format Identifier	ID
An identifier priority is highest for the smallest binary number	0 - 2047

ID Extended	IDE
Standard format (11-bit)	0
Extended format (29-bit)	1

RTR	Remote Transmission Request	
	Receive Buffer: Status of Received Frame	Transmit Buffer: Setting of RTR Bit to be Sent
0	Data frame	Data frame
1	Remote frame	Remote frame

Standard Mapping: CAN Transmit Buffer 2 Identifier Register 0 (CAN_TB2_IDR0) CAN_BASE+$70	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3
	Write									

Standard Mapping: CAN Transmit Buffer 2 Identifier Register 1 (CAN_TB2_IDR1) CAN_BASE+$71	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	ID2	ID1	ID0	RTR	IDE (=0)			
	Write									

Standard Mapping: CAN Transmit Buffer 2 Identifier Register 2 (CAN_TB2_IDR2) CAN_BASE+$72	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0								
	Write									

Standard Mapping: CAN Transmit Buffer 2 Identifier Register 3 (CAN_TB2_IDR3) CAN_BASE+$73	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0								
	Write									
	Reset	0	0	0	0	0	0	0	0	0

Reserved Bits

CAN

Extended Identifier Mapping Receive Buffer, Identifier Registers 0-3

arch.h: ArchIO.CAN.RxBuffer_IdReg ***?***
registers.h: ArchIO_CAN_RxBuffer_IdReg ***?***

Where ***?*** = 0, 1, 2, 3

Standard Format Identifier	ID
An identifier priority is highest for the smallest binary number	0 - {(2^{29}) – 1} (0 - 536870911)

IDE	ID Extended
0	Standard format (11-bit)
1	Extended format (29-bit)

Substitute Remote Request	SRR
Receive buffers. Stored as received on the CAN bus.	0
Transmission buffers	1

RTR	Remote Transmission Request	
	Receive Buffer: Status of Received Frame	Transmit Buffer: Setting of RTR Bit to be Sent
0	Data frame	Data frame
1	Remote frame	Remote frame

Extended Mapping: CAN Receive Buffer Identifier Register 0 (CAN_RB_IDR0) CAN_BASE+$40	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	ID28	ID27	ID26	ID25	ID24	ID23	ID22	ID21
	Write									

Extended Mapping: CAN Receive Buffer Identifier Register 1 (CAN_RB_IDR1) CAN_BASE+$41	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	ID20	ID19	ID18	SRR(=1)	IDE(=1)	ID17	ID16	ID15
	Write									

Extended Mapping: CAN Receive Buffer Identifier Register 2 (CAN_RB_IDR2) CAN_BASE+$42	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	ID14	ID13	ID12	ID11	ID10	ID9	ID8	ID7
	Write									

Extended Mapping: CAN Receive Buffer Identifier Register 3 (CAN_RB_IDR3) CAN_BASE+$43	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	ID6	ID5	ID4	ID3	ID2	ID1	ID0	RTR
	Write									
	Reset	0	0	0	0	0	0	0	0	0

Reserved Bits

CAN

Extended Identifier Mapping Transmit Buffer0, Identifier Registers 0-3

arch.h: ArchIO.CAN.TxBuffer[0].IdReg ***?***
registers.h: ArchIO_CAN_TxBuffer_0_IdReg ***?***

Where ***?*** = 0, 1, 2, 3

Standard Format Identifier	ID
An identifier priority is highest for the smallest binary number	0 - {(2^{29}) – 1} (0 - 536870911)

IDE	ID Extended
0	Standard format (11-bit)
1	Extended format (29-bit)

Substitute Remote Request	SRR
Receive buffers. Stored as received on the CAN bus.	0
Transmission buffers	1

RTR	Remote Transmission Request	
	Receive Buffer: Status of Received Frame	Transmit Buffer: Setting of RTR Bit to be Sent
0	Data frame	Data frame
1	Remote frame	Remote frame

Extended Mapping: CAN Transmit Buffer 0 Identifier Register 0 (CAN_TB0_IDR0) CAN_BASE+$50	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	ID28	ID27	ID26	ID25	ID24	ID23	ID22	ID21
	Write									

Extended Mapping: CAN Transmit Buffer 0 Identifier Register 1 (CAN_TB0_IDR1) CAN_BASE+$51	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	ID20	ID19	ID18	SRR(=1)	IDE(=1)	ID17	ID16	ID15
	Write									

Extended Mapping: CAN Transmit Buffer 0 Identifier Register 2 (CAN_TB0_IDR2) CAN_BASE+$52	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	ID14	ID13	ID12	ID11	ID10	ID9	ID8	ID7
	Write									

Extended Mapping: CAN Transmit Buffer 0 Identifier Register 3 (CAN_TB0_IDR3) CAN_BASE+$53	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	ID6	ID5	ID4	ID3	ID2	ID1	ID0	RTR
	Write									
	Reset	0	0	0	0	0	0	0	0	0

Reserved Bits

CAN

Extended Identifier Mapping Transmit Buffer1, Identifier Registers 0-3

arch.h: ArchIO.CAN.TxBuffer[1].IdReg ***?***
registers.h: ArchIO_CAN_TxBuffer_1_IdReg ***?***

Where ***?*** = 0, 1, 2, 3

Standard Format Identifier	ID
An identifier priority is highest for the smallest binary number	0 - {(2^{29}) – 1} (0 - 536870911)

IDE	ID Extended
0	Standard format (11-bit)
1	Extended format (29-bit)

Substitute Remote Request	SRR
Receive buffers. Stored as received on the CAN bus.	0
Transmission buffers	1

RTR	Remote Transmission Request	
	Receive Buffer: Status of Received Frame	Transmit Buffer: Setting of RTR Bit to be Sent
0	Data frame	Data frame
1	Remote frame	Remote frame

Extended Mapping: CAN Transmit Buffer 1 Identifier Register 0 (CAN_TB1_IDR0) CAN_BASE+$60	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	ID28	ID27	ID26	ID25	ID24	ID23	ID22	ID21
	Write									

Extended Mapping: CAN Transmit Buffer 1 Identifier Register 1 (CAN_TB1_IDR1) CAN_BASE+$61	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	ID20	ID19	ID18	SRR(=1)	IDE(=1)	ID17	ID16	ID15
	Write									

Extended Mapping: CAN Transmit Buffer 1 Identifier Register 2 (CAN_TB1_IDR2) CAN_BASE+$62	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	ID14	ID13	ID12	ID11	ID10	ID9	ID8	ID7
	Write									

Extended Mapping: CAN Transmit Buffer 1 Identifier Register 3 (CAN_TB1_IDR3) CAN_BASE+$63	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	ID6	ID5	ID4	ID3	ID2	ID1	ID0	RTR
	Write									
	Reset	0	0	0	0	0	0	0	0	0

Reserved Bits

CAN

Extended Identifier Mapping Transmit Buffer2, Identifier Registers 0-3

arch.h: ArchIO.CAN.TxBuffer[2].IdReg ***?***
registers.h: ArchIO_CAN_TxBuffer_2_IdReg ***?***

Where ***?*** = 0, 1, 2, 3

Standard Format Identifier	ID
An identifier priority is highest for the smallest binary number	0 - {(2^{29}) – 1} (0 - 536870911)

IDE	ID Extended
0	Standard format (11-bit)
1	Extended format (29-bit)

Substitute Remote Request	SRR
Receive buffers. Stored as received on the CAN bus.	0
Transmission buffers	1

RTR	Remote Transmission Request	
	Receive Buffer: Status of Received Frame	Transmit Buffer: Setting of RTR Bit to be Sent
0	Data frame	Data frame
1	Remote frame	Remote frame

Extended Mapping: CAN Transmit Buffer 2 Identifier Register 0 (CAN_TB2_IDR0) CAN_BASE+$70	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	ID28	ID27	ID26	ID25	ID24	ID23	ID22	ID21
	Write									

Extended Mapping: CAN Transmit Buffer 2 Identifier Register 1 (CAN_TB2_IDR1) CAN_BASE+$71	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	ID20	ID19	ID18	SRR(=1)	IDE(=1)	ID17	ID16	ID15
	Write									

Extended Mapping: CAN Transmit Buffer 2 Identifier Register 2 (CAN_TB2_IDR2) CAN_BASE+$72	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	ID14	ID13	ID12	ID11	ID10	ID9	ID8	ID7
	Write									

Extended Mapping: CAN Transmit Buffer 2 Identifier Register 3 (CAN_TB2_IDR3) CAN_BASE+$73	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	ID6	ID5	ID4	ID3	ID2	ID1	ID0	RTR
	Write									
	Reset	0	0	0	0	0	0	0	0	0

Reserved Bits

CAN

Receive Buffer Data Segment Registers 0-7 CAN_RB_DSR0-7

arch.h: ArchIO.CAN.RxBuffer.DataSegReg [***?***]
registers.h: ArchIO_CAN_RxBuffer_DataSegReg***+?***
ArchIO_CAN_RxBuffer_DataSegReg_***?***

Where ***?*** = 0, 1, 2, 3, 4, 5, 6, 7

Data Bytes (DB)
Bits DB[7:0] contain data to be received.
The data length code in the corresponding DLR register determines the number of bytes to be received.

CAN Receive Buffer Data Segment Register 0 (CAN_RB_DSR0) CAN_BASE+$44	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	Write									

CAN Receive Buffer Data Segment Register 1 (CAN_RB_DSR1) CAN_BASE+$45	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	Write									

CAN Receive Buffer Data Segment Register 2 (CAN_RB_DSR2) CAN_BASE+$46	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	Write									

CAN Receive Buffer Data Segment Register 3 (CAN_RB_DSR3) CAN_BASE+$47	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	Write									

CAN Receive Buffer Data Segment Register 4 (CAN_RB_DSR4) CAN_BASE+$48	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	Write									

CAN Receive Buffer Data Segment Register 5 (CAN_RB_DSR5) CAN_BASE+$49	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	Write									

CAN Receive Buffer Data Segment Register 6 (CAN_RB_DSR6) CAN_BASE+$4A	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	Write									

CAN Receive Buffer Data Segment Register 7 (CAN_RB_DSR7) CAN_BASE+$4B	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	Write									
	Reset	0	0	0	0	0	0	0	0	0

Reserved Bits

CAN

Transmit Buffer0 Data Segment Registers 0-7 CAN_TB0_DSR0-7

arch.h: ArchIO.CAN.TxBuffer[0].DataSegReg [***?***]
registers.h: ArchIO_CAN_TxBuffer_0_DataSegReg***+?***
ArchIO_CAN_TxBuffer_0_DataSegReg_***?***

Where ***?*** = 0, 1, 2, 3, 4, 5, 6, 7

Data Bytes (DB)
Bits DB[7:0] contain data to be transmitted.
The data length code in the corresponding DLR register determines the number of bytes to be transmitted.

CAN Transmit Buffer 0 Data Segment Register 0 (CAN_TB0_DSR0) CAN_BASE+$54	**Bits**	**15-8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	Read	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	Write									

CAN Transmit Buffer 0 Data Segment Register 1 (CAN_TB0_DSR1) CAN_BASE+$55	**Bits**	**15-8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	Read	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	Write									

CAN Transmit Buffer 0 Data Segment Register 2 (CAN_TB0_DSR2) CAN_BASE+$56	**Bits**	**15-8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	Read	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	Write									

CAN Transmit Buffer 0 Data Segment Register 3 (CAN_TB0_DSR3) CAN_BASE+$57	**Bits**	**15-8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	Read	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	Write									

CAN Transmit Buffer 0 Data Segment Register 4 (CAN_TB0_DSR4) CAN_BASE+$58	**Bits**	**15-8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	Read	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	Write									

CAN Transmit Buffer 0 Data Segment Register 5 (CAN_TB0_DSR5) .CAN_BASE+$59	**Bits**	**15-8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	Read	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	Write									

CAN Transmit Buffer 0 Data Segment Register 6 (CAN_TB0_DSR6) CAN_BASE+$5A	**Bits**	**15-8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	Read	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	Write									

CAN Transmit Buffer 0 Data Segment Register 7 (CAN_TB0_DSR7) CAN_BASE+$5B	**Bits**	**15-8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	Read	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	Write									
	Reset	0	0	0	0	0	0	0	0	0

Reserved Bits

CAN

Transmit Buffer1 Data Segment Registers 0-7 CAN_TB1_DSR0-7

arch.h: ArchIO.CAN.TxBuffer[1].DataSegReg [***?***]
registers.h: ArchIO_CAN_TxBuffer_1_DataSegReg***+?***
ArchIO_CAN_TxBuffer_1_DataSegReg_***?***

Where ***?*** = 0, 1, 2, 3, 4, 5, 6, 7

Data Bytes (DB)
Bits DB[7:0] contain data to be transmitted.
The data length code in the corresponding DLR register determines the number of bytes to be transmitted.

CAN Transmit Buffer 1 Data Segment Register 0 (CAN_TB1_DSR0) CAN_BASE+$64	**Bits**	**15-8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	Read	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	Write									

CAN Transmit Buffer 1 Data Segment Register 1 (CAN_TB1_DSR1) CAN_BASE+$65	**Bits**	**15-8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	Read	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	Write									

CAN Transmit Buffer 1 Data Segment Register 2 (CAN_TB1_DSR2) CAN_BASE+$66	**Bits**	**15-8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	Read	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	Write									

CAN Transmit Buffer 1 Data Segment Register 3 (CAN_TB1_DSR3) CAN_BASE+$67	**Bits**	**15-8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	Read	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	Write									

CAN Transmit Buffer 1 Data Segment Register 4 (CAN_TB1_DSR4) CAN_BASE+$68	**Bits**	**15-8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	Read	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	Write									

CAN Transmit Buffer 1 Data Segment Register 5 (CAN_TB1_DSR5) CAN_BASE+$69	**Bits**	**15-8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	Read	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	Write									

CAN Transmit Buffer 1 Data Segment Register 6 (CAN_TB1_DSR6) CAN_BASE+$6A	**Bits**	**15-8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	Read	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	Write									

CAN Transmit Buffer 1 Data Segment Register 7 (CAN_TB1_DSR7) CAN_BASE+$6B	**Bits**	**15-8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	Read	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	Write									
	Reset	0	0	0	0	0	0	0	0	0

Reserved Bits

CAN

Transmit Buffer2 Data Segment Registers 0-7 CAN_TB2_DSR0-7

arch.h: ArchIO.CAN.TxBuffer[2].DataSegReg [***?***]
registers.h: ArchIO_CAN_TxBuffer_2_DataSegReg***+?***
ArchIO_CAN_TxBuffer_2_DataSegReg_***?***

Where ***?*** = 0, 1, 2, 3, 4, 5, 6, 7

Data Bytes (DB)
Bits DB[7:0] contain data to be transmitted.
The data length code in the corresponding DLR register determines the number of bytes to be transmitted.

CAN Transmit Buffer 2 Data Segment Register 0 (CAN_TB2_DSR0) CAN_BASE+$74	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	Write									

CAN Transmit Buffer 2 Data Segment Register 1 (CAN_TB2_DSR1) CAN_BASE+$75	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	Write									

CAN Transmit Buffer 2 Data Segment Register 2 (CAN_TB2_DSR2) CAN_BASE+$76	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	Write									

CAN Transmit Buffer 2 Data Segment Register 3 (CAN_TB2_DSR3) CAN_BASE+$77	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	Write									

CAN Transmit Buffer 2 Data Segment Register 4 (CAN_TB2_DSR4) CAN_BASE+$78	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	Write									

CAN Transmit Buffer 2 Data Segment Register 5 (CAN_TB2_DSR5) CAN_BASE+$79	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	Write									

CAN Transmit Buffer 2 Data Segment Register 6 (CAN_TB2_DSR6) CAN_BASE+$7A	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	Write									

CAN Transmit Buffer 2 Data Segment Register 7 (CAN_TB2_DSR7) CAN_BASE+$7B	Bits	15-8	7	6	5	4	3	2	1	0
	Read	0	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
	Write									
	Reset	0	0	0	0	0	0	0	0	0

Reserved Bits

CAN

Receive Buffer Data Length Register (CAN_RB_DLR)

Transmit Buffer 0-3 Data Length Registers (CAN_TB0-3_DLRL)

arch.h: ArchIO.CAN.RxBuffer.DataLengthReg
registers.h: ArchIO_CAN_RxBuffer_DataLengthReg
arch.h: ArchIO.CAN.TxBuffer[***?***].DataLengthReg
registers.h: ArchIO_CAN_TxBuffer_***?***_DataLengthReg

Where ***?*** = 0, 1, 2 (*Xmit only*)

Data Byte Count	Data Length Code			
	DLC3	DLC2	DLC1	DLC0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0

CAN Receive Buffer Data Length Register (CAN_RB_DLR) CAN_BASE+$4C	Bits	15-4	3	2	1	0
	Read		DLC3	DLC2	DLC1	DLC0
	Write					
	Reset	*	0	0	0	0

*Reserved DLR register bits in Receive Buffer Data Length Register CAN_RB_DLR are indeterminate.

CAN Transmit Buffer 0 Data Length Register (CAN_TB0_DLR) CAN_BASE+$5C	Bits	15-4	3	2	1	0
	Read		DLC3	DLC2	DLC1	DLC0
	Write					
	Reset	0	0	0	0	0

CAN Transmit Buffer 1 Data Length Register (CAN_TB1_DLR) CAN_BASE+$6C	Bits	15-4	3	2	1	0
	Read		DLC3	DLC2	DLC1	DLC0
	Write					
	Reset	0	0	0	0	0

CAN Transmit Buffer 2 Data Length Register (CAN_TB2_DLR) CAN_BASE+$7C	Bits	15-4	3	2	1	0
	Read		DLC3	DLC2	DLC1	DLC0
	Write					
	Reset	0	0	0	0	0

Reserved Bits

CAN

Transmit Buffer0-2 Priority Register (CAN_TB0-2_TBPR)

arch.h: ArchIO.CAN.TxBuffer[***?***].TxBufferPriorityReg
registers.h: ArchIO_CAN_TxBuffer_***?***_TxBufferPriorityReg

Where ***?*** = 0, 1, 2

PRIO	Transmit Buffer Priority
0 – 255	PRIO[7:0] determines the priority of the associated message buffer. The highest priority goes to the smallest index number. When more than one bit in a PRIO[7:0] buffer is active, the lowest bit number gets the higher priority. When more than one buffer has an equal value high priority, the buffer with the lowest index number get the higher priority.

CAN Transmit Buffer 0 Priority Register (CAN_TB0_TBPR) CAN_BASE+$5D	Bits	15-8	7	6	5	4	3	2	1	0
	Read		PRIO7	PRIO6	PRIO5	PRIO4	PRIO3	PRIO2	PRIO1	PRIO0
	Write									
	Reset	0	0	0	0	0	0	0	0	0

CAN Transmit Buffer 1 Priority Register (CAN_TB1_TBPR) CAN_BASE+$6D	Bits	15-8	7	6	5	4	3	2	1	0
	Read		PRIO7	PRIO6	PRIO5	PRIO4	PRIO3	PRIO2	PRIO1	PRIO0
	Write									
	Reset	0	0	0	0	0	0	0	0	0

CAN Transmit Buffer 2 Priority Register (CAN_TB2_TBPR) CAN_BASE+$7D	Bits	15-8	7	6	5	4	3	2	1	0
	Read		PRIO7	PRIO6	PRIO5	PRIO4	PRIO3	PRIO2	PRIO1	PRIO0
	Write									
	Reset	0	0	0	0	0	0	0	0	0

Reserved Bits

ADC

ADC Control Register1 (ADCR1)

arch.h: ArchIO.Adc ***?***.ControlReg
registers.h: ArchIO_Adc ***?***_ControlReg

Where ***?*** = A *or* B

Stop	STOP
Normal Operation	0
Abort the current conversion process. Place ADC in Stop mode.	1

Start Conversion	START
No action	0
Start command is issued	1

SYNC Select	SYNC
Conversion initiated by a write to START bit only	0
Conversion initiated by a SYNC input or a write to the START bit	1

End of Scan Interrupt Enable	EOSIE
Interrupt disabled	0
Interrupt enabled	1

Zero Crossing Interrupt Enable	ZCIE
Interrupt disabled	0
Interrupt enabled	1

LLMTIE	Low Limit Interrupt Enable
0	Interrupt disabled
1	Interrupt enabled

HLMTIE	High Limit Interrupt Enable
0	Interrupt disabled
1	Interrupt enabled

Channel Configure CHNCFG			
Single Ended Input*	Single Ended Channels	Differential Input*	Differential Channel Pairs and Polarity
Bit 4 = 0	AN0 AN1	Bit 4 = 1	AN0 is +, AN1 is -
Bit 5 = 0	AN2 AN3	Bit 5 = 1	AN2 is +, AN3 is -
Bit 6 = 0	AN4 AN5	Bit 6 = 1	AN4 is +, AN5 is -
Bit 7 = 0	AN6 AN7	Bit 7= 1	AN6 is +, AN7 is -
*Each bit acts independently.			

SMODE			Scan Mode
0	0	0	Once Sequential
0	0	1	Once Simultaneous
0	1	0	Loop Sequential
0	1	1	Loop Simultaneous
1	0	0	Triggered Sequential
1	0	1	Triggered Simultaneous
1	1	0	Reserved
1	1	1	Reserved

ADC Control Register 1 (ADCR1) ADC_BASE+$0	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	STOP	0	SYNC	EOSIE	ZCIE	LLMTIE	HLMTIE	CHNCFG[3:0]				0	SMODE[2:0]		
	Write			START													
	Reset	0	1	0	1	0	0	0	0	0	0	0	0	0	1	0	1

Reserved Bits

ADC

ADC Control Register2 (ADCR2)

ADC Zero Crossing Control Register (ADZCC)

arch.h: ArchIO.Adc ***?***.Control2Reg
registers.h: ArchIO_Adc ***?***_Control2Reg
arch.h: ArchIO.Adc ***?***.ZeroCrossingReg
registers.h: ArchIO_Adc ***?***_ZeroCrossingReg

Where ***?*** = A *or* B

DIV	Clock Divisor Select
0–15	Clock Divisor Select Value = N = DIV + 1 $F_{ADC} = (F_{IPR}) / 2N$ F_{ADC} = Analog-to-Digital Converter Frequency F_{IPR} = Interface Peripheral Bus Clock Frequency

ADC Control Register 2 (ADCR2) ADC_BASE+$1	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	0	0	0	0	0	0	0	0	DIV[3:0]			
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

ZCEx	Zero Crossing Enable
00	Zero Crossing Disabled
01	Zero Crossing Enabled for Positive to Negative Sign Change
10	Zero Crossing Enabled for Negative to Positive Sign Change
11	Zero Crossing Enabled for any Sign Change
Note:	ZCE0 uses sample in ADRSLT0 ZCE7 uses sample in ADRSLT7

ADC ZeroCrossing Control Register (ADZCC) ADC_BASE+$2	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	ZCE7[1:0]		ZCE6[1:0]		ZCE5[1:0]		ZCE4[1:0]		ZCE3[1:0]		ZCE2[1:0]		ZCE1[1:0]		ZCE0[1:0]	
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reserved Bits

ADC

ADC Channel List Registers1-2 (ADLST1-2)

arch.h: ArchIO.Adc ***?***.ChannelList1Reg
registers.h: ArchIO_Adc ***?***_ChannelList1Reg
arch.h: ArchIO.Adc ***?***.ChannelList2Reg
registers.h: ArchIO_Adc ***?***_ChannelList2Reg

Where ***?*** = A *or* B

Input Channel Number to be Converted		
SAMPLEx[2:0] *x* designates the sample number.	Channel Destination. Any of the eight channels may be programmed.	
	Single Ended Configuration	Differential Configuration
000	AN0	AN0+, AN1 –
001	AN1	AN0+, AN1 –
010	AN2	AN2+, AN3 –
011	AN3	AN2+, AN3 –
100	AN4	AN4+, AN5 –
101	AN5	AN4+, AN5 –
110	AN6	AN6+, AN7 –
111	AN7	AN6+, AN7 –

SAMPLEx	
Sequential Sampling	Simultaneous Sampling
Sequential scan of inputs proceeds in order from SAMPLE0–SAMPLE7	Simultaneous Sampling Order: SAMPLE0 & SAMPLE4, SAMPLE1 & SAMPLE5 SAMPLE2 & SAMPLE6 SAMPLE3 & SAMPLE7

Configuring Channel Monitoring

- Select sequential or simultaneous Scan Mode with SMODE[2:0] bits in register ADCR1.
- Select Sample range in register ADSDIS with DSx bits. Sampling stops on encountering a set DSx bit.

  Sequential sampling starts at SAMPLE0.
  Simultaneous sampling starts at SAMPLE0 and SAMPLE4.
- Select Channel Configure with CHNCFG[3:0] bits in ADCR1 register.

  Observe differential input restrictions

ADC Channel List Register 1 (ADLST1) ADC_BASE+$3	**Bits**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	Read		SAMPLE3[2:0]				SAMPLE2[2:0]				SAMPLE1[2:0]				SAMPLE0[2:0]		
	Write																
	Reset	0	0	1	1	0	0	1	0	0	0	0	1	0	0	0	0

ADC Channel List Register 2 (ADLST2) ADC_BASE+$4	**Bits**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	Read		SAMPLE7[2:0]				SAMPLE6[2:0]				SAMPLE5[2:0]				SAMPLE4[2:0]		
	Write																
	Reset	0	1	1	1	0	1	1	0	0	1	0	1	0	1	0	0

Reserved Bits

ADC

ADC Sample Disable Register (ADSDIS)

arch.h: ArchIO.Adc ***?***.DisableReg
registers.h: ArchIO_Adc ***?***_DsiableReg

Where ***?*** = A *or* B

Test Bits	TEST
Normal Mode	00
Test Mode. Applies VREF voltage to AN0 and AN4. (Only AN0 and AN4 will have valid results.)	01
Reserved	10
Reserved	11

DS	Disable Sample
0–15	Enables SAMPLEx in ADLST1–2 registers

ADC Sample Disable Register (ADSDIS) ADC_BASE+$5	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	TEST[1:0]		0	0	0	0	0	0	DS7	DS6	DS5	DS4	DS3	DS2	DS1	DS0
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reserved Bits

ADC

ADC Status Register (ADSTAT)

arch.h: ArchIO.Adc ***?***.StatusReg
registers.h: ArchIO_Adc ***?***_StatusReg

Where ***?*** = A *or* B

Conversion in Progress	CIP
Idle	0
A Scan Cycle is in Progress	1

RDY	Ready Channel
0	Channel not ready or has been read
1	Channel Ready to be Read

End-of-Scan Interrupt	EOIS
Scan Cycle is Not Complete	0
Scan Cycle Completed	1

HLMTI	High Limit Interrupt
0	No High Limit IRQ
1	High Limit Exceeded. IRQ pending if HLMTIE is set.

Zero Crossing Interrupt	ZCI
No Zero Crossing Interrupt IRQ	0
Zero Crossing Encountered. IRQ pending if ZCIE is set.	1

LLMTI	Low Limit Interrupt
0	No Low Limit IRQ
1	Low Limit Exceeded. IRQ pending if LLMTIE is set.

ADC Status Register (ADSTAT) ADC_BASE+$6	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	CIP	0	0	0	EOSI	ZCI	LLMTI	HLMTI	RDY7	RDY6	RDY5	RDY4	RDY3	RDY2	RDY1	RDY0
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reserved Bits

ADC

ADC Limit Status Register (ADLSTAT)

arch.h: ArchIO.Adc ***?***.LimitReg
registers.h: ArchIO_Adc ***?***_LimitReg

Where ***?*** = A *or* B

High Limit	HLS
The value in the Result Register (ADRSLT0–7) is less than or equal to the value in the High Limit Register (ADHLMT0–7). Value in Result Register (ADDSLT0–7) < Value in High Limit Register (ADHLMT0–7)	0
The ADC compared the Result Register (ADRSLT0–7) with the value in the High Limit Register (ADHLMT0–7) and the Result Register value was greater than the high limit. Value in Result Register (ADDSLT0–7) > Value in High Limit Register (ADHLMT0–7)	**1**

LLS	Low Limit
0	The value in the Result Register (ADRSLT0–7) is greater than or equal to the value in the Low Limit Register (ADLLMT0–7). Value in Result Register (ADDSLT0–7) > Value in Low Limit Register (ADLLMT0–7)
1	The ADC compared the Result Register (ADRSLT0–7) with the value in the Low Limit Register (ADLLMT0–7) and the Result Register value was less than the low limit. Value in Result Register (ADDSLT0–7) < Value in Low Limit Register (ADLLMT0–7)

ADC Limit Status Register (ADLSTAT) ADC_BASE+$7	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	HLS7	HLS6	HLS5	HLS4	HLS3	HLS2	HLS1	HLS0	LLS7	LLS6	LLS5	LLS4	LLS3	LLS2	LLS1	LLS0
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ADC

ADC Zero Crossing Status Register (ADZCSTAT)

arch.h: ArchIO.Adc ***?***.ZeroCrossingStatusReg
registers.h: ArchIO_Adc ***?***_ZeroCrossingStatusReg

Where ***?*** = A *or* B

ZCS	Zero Crossing Status Register
0	1. A sign change did not occur in a comparison between the current channelx result and the previous channelx result, or 2. Zero Crossing Control is disabled for channelx in the Zero Crossing Control Register, ADZCC.
1	In a comparison between the current channelx result and the previous channelx result, a sign change condition occurred as defined in the Zero Crossing Control Register (ADZCC). **Note 1:** To clear a specific ZCS[7:0] bit, write a value of 1 to that bit.

ADC Zero Crossing Status Register (ADZCSTAT) ADC_BASE+$8	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	0	0	0	0	ZCS7	ZCS6	ZCS5	ZCS4	ZCS3	ZCS2	ZCS1	ZCS0
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reserved Bits

ADC

ADC Result Registers (ADRSLT0-7)

arch.h: ArchIO.Adc ***?***.ResultReg [***i***]
registers.h: ArchIO_Adc ***?***_ResultReg + ***i***
registers.h: ArchIO_Adc ***?***_ResultReg_***i***
Where ***?*** = A *or* B *and* ***i*** = 0, 1, 2,....,7

Sign Extend	SEXT
The result is positive. Note: If positive results are required, set the respective offset register to a value of zero.	0
The result is negative.	1

Digital Result of the Conversion — RSLT	
Signed Integer	Option 1. Right shift with sign extended (ASR) 3 places and interpret the right shifted number Option 2. Accept the number as presented with the lower 3 bits equal to zero
Signed Fractional Number	Use the ADRSLT directly

Note 1: Negative results (SEXT = 1) are presented in two's complement format. For applications requiring result registers to be positive, be sure the offset registers are set to zero.

Note 2: The interpretation of the numbers programmed into the limit and offset registers (ADLLMT, ADHLMT, ADOFS) should match the interpretation of the result register.

Note 3: Each result register may only be written when the ADC is in Stop mode (STOP bit = 1 in ADCR1 register).

Converted Results from a Scan — ADC Result Registers (ADRSLT0–7)			
Sequential Sampling		Simultaneous Sampling	
Channel Designated by:	**Scan Result Stored in:**	**Channel Pair Designated by:**	**Scan Result Stored in:**
SAMPLE0	ADRSLT0	SAMPLE0 SAMPLE4	ADRSLT0 ADRSLT4
SAMPLE1	ADRSLT1		
SAMPLE2	ADRSLT2	SAMPLE1 SAMPLE5	ADRSLT1 ADRSLT5
SAMPLE3	ADRSLT3		
SAMPLE4	ADRSLT4	SAMPLE2 SAMPLE6	ADRSLT2 ADRSLT6
SAMPLE5	ADRSLT5		
SAMPLE6	ADRSLT6	SAMPLE3 SAMPLE7	ADRSLT3 ADRSLT7
SAMPLE7	ADRSLT7		

ADC Result Registers (ADRSLT0–7) ADC_BASE+$9–$10	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	SEXT	RSLT[11:0]												0	0	0
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ADC Result Register 0—Address: ADC_BASE+$9
ADC Result Register 1—Address: ADC_BASE+$A
ADC Result Register 2—Address: ADC_BASE+$B
ADC Result Register 3—Address: ADC_BASE+$C
ADC Result Register 4—Address: ADC_BASE+$D
ADC Result Register 5—Address: ADC_BASE+$E
ADC Result Register 6—Address: ADC_BASE+$F
ADC Result Register 7—Address: ADC_BASE+$10

Reserved Bits

ADC

ADC Low/High Limit Registers 0-7 (ADLLMT0-7) (ADHLMT0-7)

arch.h: ArchIO.Adc ***?***.LowLimitReg [***i***]
registers.h: ArchIO_Adc ***?***_LowLimitReg + ***i***
ArchIO_Adc ***?***_LowLimitReg_***i***
arch.h: ArchIO.Adc ***?***.HighLimitReg [***i***]
registers.h: ArchIO_Adc ***?***_HighLimitReg + ***i***
ArchIO_Adc ***?***_HighLimitReg_***i***

Where ***?*** = A *or* B *and* ***i*** = 0, 1, 2,....,7

Low Limit – LLMT
The low limit value that the result (ADRSLT0–7) is compared against.

ADC Low Limit Register 0–7 (ADLLMT0–7) ADC_BASE+$11–$18	**Bits**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	Read	0	LLMT												0	0	0
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ADC Low Limit Register 0—Address: ADC_BASE+$11
ADC Low Limit Register 1—Address: ADC_BASE+$12
ADC Low Limit Register 2—Address: ADC_BASE+$13
ADC Low Limit Register 3—Address: ADC_BASE+$14
ADC Low Limit Register 4—Address: ADC_BASE+$15
ADC Low Limit Register 5—Address: ADC_BASE+$16
ADC Low Limit Register 6—Address: ADC_BASE+$17
ADC Low Limit Register 7—Address: ADC_BASE+$18

High Limit – HLMT
The high limit value the result (ADRSLT0–7) is compared against.

ADC High Limit Register 0–7 (ADHLMT0–7) ADC_BASE+$19–$20	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	HLMT												0	0	0
	Write																
	Reset	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0

ADC High Limit Register 0—Address: ADC_BASE+$19
ADC High Limit Register 1—Address: ADC_BASE+$1A
ADC High Limit Register 2—Address: ADC_BASE+$1B
ADC High Limit Register 3—Address: ADC_BASE+$1C
ADC High Limit Register 4—Address: ADC_BASE+$1D
ADC High Limit Register 5—Address: ADC_BASE+$1E
ADC High Limit Register 6—Address: ADC_BASE+$1F
ADC High Limit Register 7—Address: ADC_BASE+$20

Reserved Bits

ADC

ADC Offset Registers 0-7 (ADOFS0-7)

arch.h: ArchIO.Adc ***?***.OffsetReg [***i***]
registers.h: ArchIO_Adc ***?***_OffsetReg + ***i***
ArchIO_Adc ***?***_OffsetReg_***i***

Where ***?*** = A *or* B *and* ***i*** = 0, 1, 2,....,7

Offset Value – OFFSET
The OFFSET value is subtracted from the ADC result. To obtain unsigned results, program the respective offset register with a value of $0000 to give a result range of $0000 to $7FF8.

ADC Offset Registers 0–7 (ADOFS0–7) ADC_BASE+$21–$28	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	OFFSET[11:0]												0	0	0
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ADC Offset Register 0—Address: ADC_BASE+$21
ADC Offset Register 1—Address: ADC_BASE+$22
ADC Offset Register 2—Address: ADC_BASE+$23
ADC Offset Register 3—Address: ADC_BASE+$24
ADC Offset Register 4—Address: ADC_BASE+$25
ADC Offset Register 5—Address: ADC_BASE+$26
ADC Offset Register 6—Address: ADC_BASE+$27
ADC Offset Register 7—Address: ADC_BASE+$28

Reserved Bits

Application: _____ Date: _____

Programmer: _____

DEC

Decoder Control Register (DECCR)

arch.h: ArchIO.Decoder ***?***.ControlReg
registers.h: ArchIO_Decoder ***?***_ControlReg

Where ***?*** = 0 *or* 1

HOME Signal Transition Interrupt Request	HIRQ
No interrupt	0
Enable HOME interrupts	1

HOME Interrupt Enable	HIE
Disable HOME interrupts	0
Enable HOME interrupts	1

Enable HOME to Initialize Position Counters UPOS and LPOS	HIP
No action	0
Enable HOME signal	1

Use Negative Edge of Home Input	HNE
Use positive going edge-to-trigger initialization of position counters UPOS and LPOS	0
Use negative going edge-to-trigger initialization of position counters UPOS and LPOS	1

Software Triggered Initialization of Position Counters UPOS & LPOS	SWIP
No action	0
Initialize position counter	1

REV	Enable Reverse Direction Counting
0	Count normally
1	Count in the reverse direction

PH1	Enable Signal Phase Count Mode
0	Use standard quadrature decoder; PHASEA, PHASEB represent a two phase quadrature signal.
1	Bypass the quadrature decoder. A positive transition of the PHASE A input generates a count signal. PHASE B input and the DIR bit control the counter direction as indicated:

REV	PHASE B Input	Count Direction
0	0	Up
0	1	Down
1	0	Down
1	1	Up

XIRQ	Index Pulse Interrupt Request
0	No interrupt has occurred
1	Index pulse interrupt has occurred

XIE	Index Pulse Interrupt Enable
0	Index pulse interrupt is enabled
1	Index pulse interrupt is disabled

Decoder Control Register (DECCR) DEC_BASE+$0	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	HIRQ	HIE	HIP	HNE	SWIP	REV	PH1	XIRQ	XIE	XIP	XNE	DIRQ	DIE	WDE	MODE[1:0]	
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

See the following page for continuation of this register

Application: _____ Date: _____

Programmer: _____

DEC

Decoder Control Register (DECCR) Continued

arch.h: ArchIO.Decoder ***?***.ControlReg
registers.h: ArchIO_Decoder ***?***_ControlReg

Where ***?*** = 0 *or* 1

Use Negative Edge of Index Pulse	XNE
Use positive transition edge of INDEX pulse	0
Use negative transition edge of INDEX pulse	1

Watchdog Timeout Interrupt Request	DIRQ
Watchdog timeout interrupt has occurred	0
No interrupt has occurred	1

Watchdog Timeout Interrupt Enable	DIE
Watchdog timer interrupt is disabled	0
Watchdog timer interrupt has occurred	1

MODE	Switch Matrix Mode
00	Mode 0: Input captures are connected to the four input pins
01	Mode 1: Input captures are connected to the filtered versions of the four input pins
10	Mode 2: PHASEA input is connected to both channels 0 and 1 of the timer to allow capture of both rising and falling edges; Phase B input is connected to both channels 2 and 3 of the timer.
11	Mode 3: Reserved

WDE	Watchdog Enable
0	Watchdog timer is disabled
1	Watchdog timer is enabled

Decoder Control Register (DECCR) DEC_BASE+$0	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	HIRQ	HIE	HIP	HNE	SWIP	REV	PH1	XIRQ	XIE	XIP	XNE	DIRQ	DIE	WDE	MODE[1:0]	
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

DEC

Filter Delay Register (FIR)

Watchdog Timeout Register (WTR)

arch.h: ArchIO.Decoder ***?***.FilterIntervalReg
registers.h: ArchIO_Decoder ***?***_FilterIntervalReg

arch.h: ArchIO.Decoder ***?***.WatchdogTimeoutReg
registers.h: ArchIO_Decoder ***?***_WatchdogTimeoutReg

Where ***?*** = 0 *or* 1

DELAY	Filter Interval Delay
0	The Quadrature Decoder bypasses the digital filter for the PHASEA, PHASEB, INDEX, and HOME inputs.
1 – 255	Total Delay = [(DELAY[7:0] + 1) x 4] + 2 Where (DELAY[7:0] + 1) = Filter Interval Period (in increments of IPBus Clock period)

Filter Delay Register (FIR) DEC_BASE+$1	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	0	0	0	0	DELAY[7:0]							
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

CNT	Count
0 – 65535	Number of clock cycles that the Quadrature Decoder module watchdog timer will count before timing out and optionally generating an interrupt in the DIRQ bit in register DECCR Note: The Quadrature Decoder module watchdog timer is separate from the watchdog timer in the clock module.

Watchdog Timeout Register (WTR) DEC_BASE+$2	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	CNT[15:0]															
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reserved Bits

DEC

Position Difference Counter Register (POSD)

Position Difference Hold Register (POSDH)

arch.h: ArchIO.Decoder ***?***.PositionDifferenceReg
registers.h: ArchIO_Decoder ***?***_PositionDifferenceReg

arch.h: ArchIO.Decoder ***?***.PositionDifferenceHoldReg
registers.h: ArchIO_Decoder ***?***_PositionDifferenceHoldReg

Where ***?*** = 0 *or* 1

POSD	Position Difference Count
0 – 65535	Contains the change in position that occurs between each read of the position counter register The position difference count value is proportional to the change in position since the last time the position counter was read. Note: When any of the counter registers are read, the contents of all counter registers are copied as a snapshot into their respective hold registers.

Position Difference Counter Register (POSD) DEC_BASE+$3	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	POSD[15:0]															
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

POSDH	Position Difference Hold Value
0 – 65535	The Position Difference Hold Register contains a snapshot of the change in position value that occurs between each read of the position register The value stored in the Position Difference Hold Register (POSDH) is copied from the POSD register when the position register is read.

Position Difference Hold Register (POSDH) DEC_BASE+$4	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	POSDH[15:0]															
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

DEC

Revolution Counter Register (REV)

Revolution Hold Register (REVH)

arch.h: ArchIO.Decoder ***?***.RevolutionCounterReg
registers.h: ArchIO_Decoder ***?***_RevolutionCounterReg

arch.h: ArchIO.Decoder ***?***.RevolutionHoldReg
registers.h: ArchIO_Decoder ***?***_RevolutionHoldReg

Where ***?*** = 0 *or* 1

REV	Revolution Count
0 – 65535	The REV register contains the current value of the Revolution Counter. Note: When any of the counter registers are read, the contents of all counter registers are copied as a snapshot into their respective hold registers.

Revolution Counter Register (REV) DEC_BASE+$5	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	REV[15:0]															
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

REV	Revolution Count Hold Value
0 – 65535	The REVH register contains a snapshot of the current value of the REV Register. The value stored in the REVH register is copied from the REV register when the revolution count is read.

Revolution Hold Register (REVH) DEC_BASE+$6	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	REV[15:0]															
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

DEC

Upper Position Register (UPOS)

Lower Position Counter Register (LPOS)

arch.h: ArchIO.Decoder **?**.UpperPositionCounterReg
registers.h: ArchIO_Decoder **?**_UpperPositionCounterReg

arch.h: ArchIO.Decoder **?**.LowerPositionCounterReg
registers.h: ArchIO_Decoder **?**_LowerPositionCounterReg

Where **?** = 0 *or* 1

POS	Position Count – Most Significant Half
0 – 65535	The Upper Position Counter Register (UPOS) contains the upper (most significant) half of the current value of the Position Counter. The value in UPOS is the binary count from the position counter.

Upper Position Counter Register (UPOS) DEC_BASE+$7	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	POS[31:16]															
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

POS	Position Count – Least Significant Half
0 – 65535	The Lower Position Counter Register (LPOS) contains the lower (least significant) half of the current value of the Position Counter. The value in LPOS is the binary count from the position counter.

Lower Position Counter Register (LPOS) DEC_BASE+$8	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	POS[15:0]															
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

DEC

Upper Position Hold Register (UPOSH)

Lower Position Hold Register (LPOSH)

arch.h: ArchIO.Decoder **?**.UpperPositionHoldReg
registers.h: ArchIO_Decoder **?**_UpperPositionHoldReg

arch.h: ArchIO.Decoder **?**.LowerPositionHoldReg
registers.h: ArchIO_Decoder **?**_LowerPositionHoldReg

Where **?** = 0 *or* 1

POS	Position Counter
0 – 65535	The Position Counter is a *snapshot* of the upper (most significant) half of the current value of the Position Counter.

Upper Position Hold Register (UPOSH) DEC_BASE+$9	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	POS[31:16]															
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

POS	Position Counter
0 – 65535	The Position Counter is a *snapshot* of the lower (least significant) half of the current value of the Position Counter.

Lower Position Hold Register (LPOSH) DEC_BASE+$A	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	POS[15:0]															
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

DEC

Upper Initialization Register (UIR)

Lower Initialization Register (LIR)

arch.h: ArchIO.Decoder ***?***.UpperInitializationReg
registers.h: ArchIO_Decoder ***?***_UpperInitializationReg

arch.h: ArchIO.Decoder ***?***.LowerInitializationReg
registers.h: ArchIO_Decoder ***?***_LowerInitializationReg

Where ***?*** = 0 *or* 1

Initialization Value	Initialization Value
0 – 65535	The Initialization value initializes the upper half of the Position Counter (UPOS).

Upper Initialization Register (UIR) DEC_BASE+$B	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read / Write	INITIALIZATION VALUE[15:0]															
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Initialization Value	Initialization Value
0 – 65535	The Initialization value initializes the lower half of the Position Counter (LPOS).

Lower Initialization Register (LIR) DEC_BASE+$C	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read / Write	INITIALIZATION VALUE[15:0]															
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

DEC

Input Monitor Register (IMR)

arch.h: ArchIO.Decoder ***?***.InputMonitorReg
registers.h: ArchIO_Decoder ***?***_InputMonitorReg

Where ***?*** = 0 *or* 1

Bit Name	Bit Number	Input Description
HOME	0	Raw HOME input
INDEX	1	Raw INDEX input
PHB	2	Raw PHASEB input
PHA	3	Raw PHASEA input
FHOM	4	Filtered version of HOME input
FIND	5	Filtered version of INDEX input
FPHB	6	Filtered version of PHASEB input
FPHA	7	Filtered version of PHASEA input

Input Monitor Register (IMR) DEC_BASE+$D	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	0	0	0	0	FPHA	FPHB	FIND	FHOM	PHA	PHB	INDEX	HOME
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reserved Bits

PWM

PWM Control Register (PMCTL)

arch.h: ArchIO.Pwm *?*.ControlReg
registers.h: ArchIO_Pwm *?*_ControlReg
Where *?* = A *or* B

PWM Reload Frequency	LDFQ
Every PWM opportunity	0000
Every 2 PWM opportunities	0001
Every 3 PWM opportunities	0010
Every 4 PWM opportunities	0011
Every 5 PWM opportunities	0100
Every 6 PWM opportunities	0101
Every 7 PWM opportunities	0110
Every 8 PWM opportunities	0111
Every 9 PWM opportunities	1000
Every 10 PWM opportunities	1001
Every 11 PWM opportunities	1010
Every 12 PWM opportunities	1011
Every 13 PWM opportunities	1100
Every 14 PWM opportunities	1101
Every 15 PWM opportunities	1110
Every 16 PWM opportunities	1111

Half Cycle Reload	HALF
Half-cycle reloads disabled	0
Half-cycle reloads enabled	1

Current Polarity Bit 2	IPOL2
PWM value register 5 in next PWM cycle	0
PWM value register 4 in next PWM cycle	1

Current Polarity Bit 1	IPOL1
PWM value register 3 in next PWM cycle	0
PWM value register 2 in next PWM cycle	1

Current Polarity Bit 0	IPOL0
PWM value register 1 in next PWM cycle	0
PWM value register 0 in next PWM cycle	1

PRSC	PWM Clock Frequency
00	f_{IPBus}
01	$f_{IPBus/2}$
10	$f_{IPBus/4}$
11	$f_{IPBus/8}$

PWMRIE	PWM Reload Interrupt Enable Bit
0	**PWMF CPU interrupt requests disabled**
1	PWMF CPU interrupt requests enabled

PWMF	PWM Reload Flag
0	No new reload cycle since last PWMF clearing
1	New reload cycle since last PWMF clearing

ISENS	Current Status Bits—Correction Method
0X	Manual correction or no correction
10	Automatic current status sample correction on pins IS1, IS2, and IS3 during deadtime.
11	Automatic current status sample correction on pins IS1, IS2, and IS3: – At the half cycle in center-aligned operation – At the end of the cycle in edge-aligned operation

LDOK	Load Okay Bit
0	Do not load new modulus, prescaler, and PWM values
1	Load prescaler, modulus, and PWM values

PWMEN	PWM Enable Bit
0	PWM generator and PWM Pins disabled unless OUTCTL = 1
1	PWM generator and PWM pins enabled

PWM Control Register (PMCTL) PWM_BASE+$0	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	LDFQ[3:0]				HALF	IPOL2	IPOL1	IPOL0	PRSC		PWMRIE	PWMF	ISENS		LDOK	PWMEN
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

PWM

PWM Fault Control Registers (PMFCTL)

arch.h: ArchIO.Pwm ***?***.FaultControlReg
registers.h: ArchIO_Pwm ***?***_FaultControlReg

Where ***?*** = A *or* B

FAULTx Pin Interrupt Enable–Bits 7, 5, 3, 1	FIEx
FAULTx CPU interrupt requests disabled	0
FAULTx CPU interrupt requests enabled	1

FMODEx	FAULTx Pin Clearing Mode–Bits 6, 4, 2, 0
0	Manual fault clearing of FAULTx pin faults
1	Automatic fault clearing of FAULTx pin faults

PWM Fault Control Register (PMFCTL) PWM_BASE+$1	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	0	0	0	0	FIE3	FMODE3	FIE2	FMODE2	FIE1	FMODE1	FIE0	FMODE0
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reserved Bits

PWM

PWM Fault Status & Acknowledge Register (PMFSA)

arch.h: ArchIO.Pwm ***?***.FaultStatusReg
registers.h: ArchIO_Pwm ***?***_FaultStatusReg

Where ***?*** = A *or* B

FAULTx Pin–Bits 15, 13, 11, 9	FPINx
Logic 0 on the FAULTx pin	0
Logic 1 on the FAULTx pin	1

FAULTx Pin Flag–Bits 14, 12, 10, 8	FFLAGx
No fault on the FAULTx pin	0
Fault on the FAULTx pin	1

FAULTx Pin Acknowledge—Bits 6, 4, 2, 0	FTACKx
No effect Note: FTACKx always reads as logic 0.	0
Clears FFLAGx	1

Dead Time Bits, DTx (DTx bits are a timing marker indicating when to toggle between PWM value registers.)		ISx is sampled at the rising edge of the PWM cycle (the start of Dead Time) for the PWM0–5 pins.
DT0, DT1 Pair	Samples the IS0 current sense pin	PWM0
		PWM1
DT2, DT3 Pair	Samples the IS1 current sense pin	PWM2
		PWM3
DT4, DT5 Pair	Samples the IS2 current sense pin	PWM4
		PWM5
Note: To detect the current status, the voltage on each ISx pin is sampled twice in a PWM period at the end of each deadtime, and the ISx values are stored in the DTx bits.		

PWM Fault Status & Acknowledge Register (PMFSA) PWM_BASE+$2	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	FPIN3	FFLAG3	FPIN2	FFLAG2	FPIN1	FFLAG1	FPIN0	FFLAG0	0	0	DT5	DT4	DT3	DT2	DT1	DT0
	Write										FTACK3		FTACK2		FTACK1		FTACK0
	Reset	U	0	U	0	U	0	U	0	0	0	0	0	0	0	0	0

Reserved Bits

PWM

PWM Output Control Register (PMOUT)

arch.h: ArchIO.Pwm **?**.OutputControlReg
registers.h: ArchIO_Pwm **?**_OutputControlReg

Where ***?*** = A *or* B

Output Pad Enable	PAD_EN
Output pad disabled	0
Output pad enabled	1

Output Control Enable	OUTCTLx
Software control disabled	0
Software control enabled	1
Note: When an OUTCTLx bit is set, the OUTx bit activates and deactivataes the PWMx output.	

OUTx		Output Control	
		Complementary Channel Operation	**Independent Channel Operation**
OUT0	= 0	PWM0 is inactive	PWM0 is inactive
	= 1	PWM0 is active	PWM0 is Active
OUT1	= 0	PWM1 is inactive	PWM1 is inactive
	= 1	PWM1 is complement of PWM0	PWM1 is active
OUT2	= 0	PWM2 is inactive	PWM2 is inactive
	= 1	PWM2 is active	PWM2 is active
OUT3	= 0	PWM3 is inactive	PWM3 is inactive
	= 1	PWM3 is complement of PWM2	PWM3 is active
OUT4	= 0	PWM4 is inactive	PWM4 is inactive
	= 1	PWM4 is active	PWM4 is active
OUT5	= 0	PWM5 is inactive	PWM5 is inactive
	= 1	PWM5 is complement of PWM4	PWM5 is active
Note: When the corresponding OUTCTLx bit is set, the readable and writeable OUTx bits control the PWMx pins.			

PWM Output Control Register (PMOUT) PWM_BASE+$3	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	PAD_EN	0	OUTCTLx						0	0	OUTx					
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reserved Bits

PWM

PWM Counter Register (PMCNT)

PWM Counter Modulo Register (PWMCM)

arch.h: ArchIO.Pwm ***?***.CounterReg
registers.h: ArchIO_Pwm ***?***_CounterReg

arch.h: ArchIO.Pwm ***?***.CounterModuloReg
registers.h: ArchIO_Pwm ***?***_CounterModuloReg

Where ***?*** = A *or* B

Counter Register (CR)
Displays the state of the 15–bit PWM counter

PWM Counter Register (PMCNT) PWM_BASE+$4	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	CR[14:0]														
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

PWMCM	PWM Counter Modulo
0	Modulus = 0 is illegal **Note:** A modulus value of 0 results in waveforms inconsistent with other modulus waveforms. With the modulus = 0, the counter continually counts down from $7FFF.
1 – 32767	The modulus of the PWM counter Center-aligned operation: – PWM counter is an up/down counter – PWM Period = (PWM Modulus) x (PWM Clock Period) x 2 Edge-aligned operation: – PWM counter is an up counter – PWM Period = (PWM Modulus) x (PWM Clock Period)

PWM Counter Modulo Register (PWMCM) PWM_BASE+$5	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	PWMCM[14:0]														
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reserved Bits

PWM

PWM Value Registers (PWMVAL0–5)

arch.h: ArchIO.Pwm ***?***.ValueReg [***i***]
registers.h: ArchIO_Pwm ***?***_ValueReg+***i***
ArchIO_Pwm ***?***_ValueReg_***i***

Where ***?*** = A *or* B *and* ***i*** = 0, 1, 2, 3, 4, 5

PWM Value (PWMVAL[15:0])

The PWMVAL[15:0] signed 16-bit value is the pulse width in PWM clock periods of the PWM generator output.

$$(DutyCycle) = \left[\frac{PWM\ Value}{Modulus}\right] \times 100$$

Note:		
	PWMVAL[15:0] ≤ 0:	Deactivates the PWM output for the entire PWM period
	PWMVAL[15:0] ≥ Modulus:	Activates the PWM output for the entire PWM period

PWM Value Registers (PWMVAL0–5) PWM_BASE + $6–$B	**Bits**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	Read	PWMVAL[15:0]															
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

PWM Channel 0 Value Register (PWMVAL0)—Address: PWM_BASE + $6
PWM Channel 1 Value Register (PWMVAL1)—Address: PWM_BASE + $7
PWM Channel 2 Value Register (PWMVAL2)—Address: PWM_BASE + $8
PWM Channel 3 Value Register (PWMVAL3)—Address: PWM_BASE + $9
PWM Channel 4 Value Register (PWMVAL4)—Address: PWM_BASE + $A
PWM Channel 5 Value Register (PWMVAL5)—Address: PWM_BASE + $B

PWM

PWM Deadtime Register (PMDEADTM)

arch.h: ArchIO.Pwm ***?***.DeadtimeReg
registers.h: ArchIO_Pwm ***?***_DeadtimeReg

Where ***?*** = A *or* B

PWM Deadtime (PWMDT)
The number of PWM clock cycles in complementary channel operation DT = P x (PWMDT[7:0]) – 1 DT = Deadtime Duration P = Prescaler Value PWMDT[7:0] = PWM Deadtime **Note:** The PMDEADTM register is write protected after the WP bit in the PWM configuration register is set.

PWM Deadtime Register (PMDEADTM) PWM_BASE+$C	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	0	0	0	0	PWMDT[7:0]							
	Write																
	Reset	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Reserved Bits

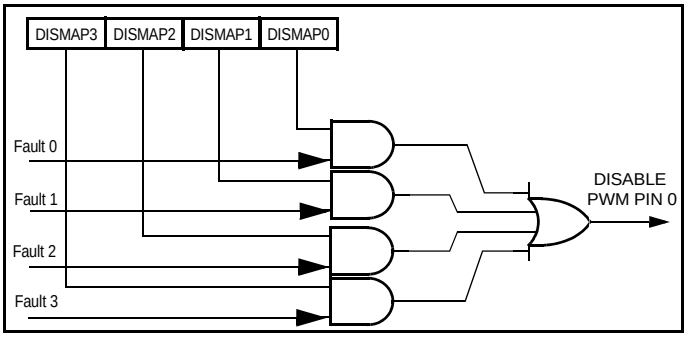
PWM

PWM Disable Mapping Registers 1 & 2 (PMDISMAP1–2)

arch.h: ArchIO.Pwm ?.DisableMapping***i***Reg
registers.h: ArchIO_Pwm ***?***_DisableMapping***i***Reg

Where ***?*** = A *or* B *and* ***i*** = 1 *or* 2

PWM Disable Mapping Register	DISMAP1 PMDISMAP2
In each four bit bank, each bit has teh effect indicated. For a complete discription of the effect of each four bit bank, refer to the figure below.	
Fault Protection Disabled	0
Fault Protection Enabled	1



PWM Disable Mapping Registers (PMDISMAP1–2)	
Each bank of four DISMAPx bits controls the mapping for a single PWM pin and determines which PWM pins are disabled by the fault protection inputs.	
Controlling Register Bits	Disabled PWM Pin
DISMAP3—DISMAP0	PWM0
DISMAP7—DISMAP4	PWM1
DISMAP11—DISMAP8	PWM2
DISMAP15—DISMAP12	PWM3
DISMAP19—DISMAP16	PWM4
DISMAP23—DISMAP20	PWM5
Note: The PMDISMAP1–2 registers are write protected after the WP bit in the PWM Config. Register is set.	

PWM Disable Mapping Register1 (PMDISMAP1) PWM_BASE+$D	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	DIS MAP15	DIS MAP14	DIS MAP13	DIS MAP12	DIS MAP11	DIS MAP10	DIS MAP9	DIS MAP8	DIS MAP7	DIS MAP6	DIS MA5	DIS MAP4	DIS MAP3	DIS MAP2	DIS MAP1	DIS MAP0
	Write																
	Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

PWM Disable Mapping Register2 (PMDISMAP2) PWM_BASE+$E	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	0	0	0	0	DIS MAP23	DIS MAP22	DIS MA21	DIS MAP20	DIS MAP19	DIS MAP18	DIS MAP17	DIS MAP16
	Write																
	Reset	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Reserved Bits

PWM

PWM Configuration Register (PMCFG)

arch.h: ArchIO.Pwm ?.ConfigReg
registers.h: ArchIO_Pwm ***?***_ConfigReg

Where ***?*** = A *or* B

Edge-Aligned or Center-Aligned PWM Channels	EDG
Center-aligned PWMs	0
Edge-aligned PWMs	1

Top-Side PWM Polarity Bit	TOPNEG
Positive top-side polarity	0
Negative top-side polarity	1
Note: Each PWM channel pair is configurable: Channel 0–1, Channel 2–3, Channel 4–5.	

Bottom-Side PWM Polarity Bit	BOTNEG
Positive bottom-side polarity	0
Negative bottom-side polarity	1
Note: Each PWM Channel Pair Is Configurable: Channel 0–1, Channel 2–3, Channel 4–5.	

INDEP	Independent or Complimentary Pair Operation
0	Complementary PWM pair
1	Independent PWM channels
Note: Each PWM Channel Pair Is Configurable: Channel 0–1, Channel 2–3, Channel 4–5.	

WP	Write Protect Bit
0	Write-protectable registers are writeable
1	Write-protectable registers are write-protected The write protectable registers are: PMDISMAP1–2, PMDEADTM, PMCFG, and PMCCR.
Note: Once set, the WP bit can only be cleared by a PMCFG system reset.	

PWM Config. Register (PMCFG) PWM_BASE+$F	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	EDG	0	TOP NEG 45	TOP NEG 23	TOP NEG 01	0	BOTNEG 45	BOTNEG 23	BOTNEG 01	INDEP 45	INDEP 23	INDEP 01	WP
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reserved Bits

PWM

PWM Channel Control Register (PMCCR)

arch.h: ArchIO.Pwm ?.ChannelControlReg
registers.h: ArchIO_Pwm ***?***_ChannelControlReg

Where ***?*** = A *or* B

Enable Hardware Acceleration	ENHA
Disable writing to VLMODE[1:0], SWP45, SWP23, SWP01 bits	0
Enable writing to VLMODE[1:0], SWP45, SWP23, SWP01 bits	1
Note: This bit can only be written if WP = 0.	

VLMODE	Value Register Load Mode
00	Each value register is accessed independently
01	Writing to value register 1 also writes to value registers 1–6
10	Writing to value register 1 also writes to value registers 1–4
11	Reserved

Mask		MSKx
The MSKx bits determine the mask for each of the corresponding PWM logical channels.		
Unmasked		0
Masked; channel is set to a value of 0% duty cycle at the PWM generator.		1
MSK5	PWM Channel 5	
MSK4	PWM Channel 4	
MSK3	PWM Channel 3	
MSK2	PWM Channel 2	
MSK1	PWM Channel 1	
MSK0	PWM Channel 0	

SWP45, SWP23, SWP01	Swap	
0	No swap	
1	Channels are swapped	
	SWP45	Channels 4 and 5 are swapped
	SWP23	Channels 2 and 3 are swapped
	SWP01	Channels 0 and 1 are swapped

PWM Channel Control Register (PMCCR) PWM_BASE+$10	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	ENHA	0	MSK5	MSK4	MSK3	MSK2	MSK1	MSK0	0	0	VLMODE[1:0]		0	SWP45	SWP23	SWP01
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reserved Bits

PWM

PWM Port Register (PMPORT)

arch.h: ArchIO.Pwm ?.PortReg
registers.h: ArchIO_Pwm ***?***_PortReg

Where ***?*** = A *or* B

PWM Port Register (PORT)						
The PMPORT register contains the values of the three current status inputs (IS0–IS2) and the values of the four fault inputs (FAULT0–FAULT3). These values are reclocked/synchronized by the IPBus clock.						
Current Status Inputs			**Fault Inputs**			
Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
IS2	IS1	IS0	FAULT3	FAULT2	FAULT1	FAULT0
Note: The PMPORT register may be read while the PWM is active.						

PWM Port Register (PMPORT) PWM_BASE+$11	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	0	0	0	0	0	PORT [6:0]						
	Write																
	Reset	0	0	0	0	0	0	0	0	0	U*	U	U	U	U	U	U

Reserved Bits

* U = Unaffected by Reset

Application: _____ Date: _____

Programmer: _____

SCI

SCI Baud Rate Regsiter (SCIBR)

arch.h: ArchIO.Sci ?.BaudRateReg
registers.h: ArchIO_Sci ***?***_BaudRateReg

Where ***?*** = 0 *or* 1

SBR	SCI Baud Rate
0	Baud rate generator disabled
1 – 8191	$\text{Baud Rate} = \frac{\text{SCI Module Clock}}{16 \times \text{SBR}}$
Note: The baud rate generator is disabled until the TE or RE bit in register SCICR is set for the first time after reset.	

SCI Baud Rate Register (SCIBR) SCI_BASE+$0	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	SBR												
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

Reserved Bits

SCI

SCI Control Regsiter (SCICR)

arch.h: ArchIO.Sci ?.ControlReg
registers.h: ArchIO_Sci ***?***_ControlReg

Where ***?*** = 0 *or* 1

Loop Select	LOOP
Normal operation enabled	0
Loop operation enabled. (See Notes 1 and 2.)	1

Note 1: To use the internal loop function, the transmitter and receiver must be enabled: RE = 1, TE = 1.

Note 2: When LOOP = 1, the RSRC bit determines the internal feedback path for the receiver. (See the Loop Functions table at the right for details.)

POL	Polarity Bit
0	Do not invert transmit and receive data bits (normal mode)
1	Invert transmit and receive data bits (inverted mode)

PE	Parity Enable Bit
0	Parity function disabled
1	Parity function enabled

Stop in Wait Mode	SWAI
SCI enabled in wait mode	0
SCI disabled in wait mode	1

PT	Parity Type Bit
0	Even parity
1	Odd parity

Receiver Source	RSRC
Receiver input internally Connected to transmitter output	0
Receiver Input connected to TXD pin	1

Data Format Mode	M
Eight data characters, one start bit, one stop bit	0
Nine data characters, one start bit, one stop bit	1

WAKE	Wake Up Condition
0	Idle line Wake up
1	Address mark Wake up

SCI Control Register (SCICR) SCI_BASE+$1	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	LOOP	SWAI	RSRC	M	WAKE	POL	PE	PT	TEIE	TIIE	RIE	REIE	TE	RE	RWU	SBK
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

LOOP	RSRC	Loop Functions
0	x	Normal Operation
1	0	Loop-Mode with Internal TXD Fed Back to RXD
1	1	Single-Wire Mode with TXD Output Fed Back to RXD

See the following page for continuation of this register

SCI

SCI Control Register (SCICR) Continued

arch.h: ArchIO.Sci ?.ControlReg
registers.h: ArchIO_Sci ***?***_ControlReg

Where ***?*** = 0 *or* 1

Transmitter Empty Interrupt Enable	TEIE
TDRE interrupt requests disabled	0
TDRE interrupt requests enabled	1

Transmitter Idle Interrupt Enable	TIIE
TI interrupt requests disabled	0
TI interrupt requests enabled	1

Receive Full Interrupt Enable	RIE
RDRF and OR interrupt requests disabled	0
RDRF and OR interrupt requests enabled	1

Receive Error Interrupt Enable	REIE
Error interrupt requests disabled	0
Error interrupt requests enabled	1

TE	Transmitter Enable
0	Transmitter disabled—TXD pin is in high-impedance state
1	Transmitter enabled

RE	Receiver Enable
0	Receiver disabled
1	Receiver enabled

RWU	Receiver Wake Up
0	Normal operation
1	Transmit break characters

SBK	Send Break
0	No break characters transmitted
1	Transmit break characters

SCI Control Register (SCICR) SCI_BASE+$1	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	LOOP	SWAI	RSRC	M	WAKE	POL	PE	PT	TEIE	TIIE	RIE	REIE	TE	RE	RWU	SBK
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

SCI

SCI Staus Regsiter (SCISR)

arch.h: ArchIO.Sci ?.StatusReg
registers.h: ArchIO_Sci **?**_StatusReg

Where ***?*** = 0 *or* 1

Transmit Data Register Empty Flag	TDRE
No character transferred to transmit shift register	0
Character transferred to transmit shift register; transmit data register empty	1

Transmitter Idle Flag	TIDLE
Transmission in progress	0
No transmission in progress	1

Receive Data Register Full Flag	RDRF
Data not available in SCI data register	0
Received data available in SCI data register	1

Receiver Idle Line Flag	RIDLE
Receiver input is either active now or has never become active since the RIDLE flag was last cleared	0
Receiver input has become idle (after receiving a valid frame)	1

OR	Overrun Flag
0	No overrun
1	Overrun

NF	Noise Flag
0	No Noise
1	Noise

FE	Framing Error Flag
0	No framing error
1	Framing error

PF	Parity Error Flag
0	No parity error
1	Parity error

RAF	Receiver Active Flag
0	No reception in progress
1	Reception in progress

SCI Status Register (SCISR) SCI_BASE+$2	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	TDRE	TIDLE	RDRF	RIDLE	OR	NF	FE	PF	0	0	0	0	0	0	0	RAF
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reserved Bits

SCI

SCI Data Regsiter (SCIDR)

arch.h: ArchIO.Sci ?.DataReg
registers.h: ArchIO_Sci ***?***_DataReg

Where ***?*** = 0 *or* 1

SCIDR Register
Nine bits of data in the SCIDR register can be read at any time, and can also be written to at any time.

Receive Data
During a read, 9 bits of received data may be accessed.

Transmit Data
During a write, 9 bits of data to be transmitted may be accessed.

SCI Data Register (SCIDR) SCI_BASE+$3	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	0	0	0	RECEIVE DATA								
	Write								TRANSMIT DATA								
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reserved Bits

Application: _____ Date: _____

_____ Programmer: _____

SPI

SCI Status and Control Regsiter (SPSCR)

arch.h: ArchIO.Spi.ControlReg
registers.h: ArchIO_Spi_ControlReg

Data Shift Order	DSO
MSB transmitted first (MSB ≥ LSB)	0
LSB transmitted first (LSB ≥ MSB)	1
Note: LSB is always at location 0, and MSB is at the correct bit position when reading/writing registers SPDRR, or SPDTR.	

SPI Receiver Full	SPRF
Receive Data Register not full	0
Receive Data Register full	1
Note 1: SPRF clears automatically after register SPDRR is read.	
Note 2: SPRF generates an interrupt request if bit SPRIE is set.	

Error Interrupt Enable	ERRIE
MODF and OVRF cannot generate interrupt requests	0
MODF and OVRF can generate interrupt requests	1

Overflow	OVRF
No overflow	0
Overflow	1

Mode Fault	MODF
$\overline{SS}$ pin at appropriate logic level	0
$\overline{SS}$ pin at inappropriate logic level	1
Slave SPI: MODF is set if the $\overline{SS}$ pin goes high during a transmission with MODFEN bit set.	
Master SPI: MODF is set if the $\overline{SS}$ pin goes low at any time with the MODFEN bit set.	

SPTE	SPI Transmitter Empty
0	Transmit data register not empty
1	Transmit data register empty
Note: SPTE generates an interrupt request if the SPTIE bit is set.	

MODFEN	Mode Fault Enable
0	Slave SPI: Prevents the MODF flag from being set for an enabled SPI Master SPI: The $\overline{SS}$ pin level does not affect the operation of an enabled SPI.
1	Allows the MODF flag to be set Clearing MODFEN does not clear the MODF flag.

SPI Baud Rate Select			
Slave Mode	SPRx has no effect		
Master Mode	SPRx bits select one of four baud rates listed in the table below		
	SPR1	SPR0	BD
	0	0	2
	0	1	8
	1	0	16
	1	1	32
Baud Rate = (clk / BD) where clk = IPBus Clock BD = Baud Rate Divisor			

SPI Status and Control Register (SPSCR) SPI_BASE+$0	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	DSO	SPRF	EERIE	OVRF	MODF	SPTE	MODFEN	SPR1	SPR0	SPRIE	SPMSTR	CPOL	CPHA	SPE	SPTIE
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reserved Bits

See the following page for continuation of this register

SPI

SCI Status and Control Regsiter (SPSCR) Continued

arch.h: ArchIO.Spi.ControlReg
registers.h: ArchIO_Spi_ControlReg

SPI Receiver Interrupt Enable	SPRIE
SPRF interrupt requests disabled	0
SPRF interrupt requests enabled	1
Note: Bit SPRF is set when a full data length transfers from the shift register to the SPDRR register.	

SPI Master Bit	SPMSTR
Slave mode operation selected	0
Master mode operation selected	1

Clock Polarity Bit	SPMSTR
Rising edge of SCLK starts transmission	0
Falling edge of SCLK starts transmission	1
Note: To transmit data between SPI modules, the SPI modules must have identical CPOL values.	

SPI Transmit Interrupt Enable	SPTIE
SPTE interrupt requests disabled	0
SPTE interrupt requests enabled	1
Note: SPTE is set when a full data length transfers from the SPDTR register to the shift register.	

CPHA	Clock Phase	
	CPHA controls the timing relationship between the serial clock and SPI data. The SPI modules must have identical CPHA values to transmit data.	
	SPI Configured as a Slave	**SPI Configured as a Master**
0	A falling edge on the $\overline{SS}$ pin starts the slave data transmission. The $\overline{SS}$ pin of the slave must be toggled high and back to low between each full length data transmission.	The SCLK signal remains inactive for the first half period.
1	The first SCLK edge starts a transmission. The $\overline{SS}$ pin can remain low between transmissions.	The first SCLK cycle begins with an edge on the SCLK line from its inactive to active level.

SPE	SPI Enable
0	SPI module disabled
1	SPI module enabled
Note: Clearing the SPE causes a partial reset of the SPI.	

SPI Status and Control Register (SPSCR) SPI_BASE+$0	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	DSO	SPRF	EERIE	OVRF	MODF	SPTE	MODFEN	SPR1	SPR0	SPRIE	SPMSTR	CPOL	CPHA	SPE	SPTIE
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reserved Bits

SPI

SPI Data Size Register (SPDSR)

arch.h: ArchIO.Spi.DataSizeReg
registers.h: ArchIO_Spi_DataSizeReg

DS3–DS0	Data Size— Data Length for Each Transmission
0000	Not Allowed
0001	2 bits
0010	3 bits
0011	4 bits
0100	5 bits
0101	6 bits
0110	7 bits
0111	8 bits
1000	9 bits
1001	10 bits
1010	11 bits
1011	12 bits
1100	13 bits
1101	14 bits
1110	15 bits
1111	16 bits

Note 1: The master and slave must transfer the same data length on each transmission.

Note 2: To cause a new value to take effect, disable and then enable the SPE bit in the SPSCR register.

SPI Data Size Register (SPDSR) SPI_BASE+$1	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	0	0	0	0	0	0	0	0	DS3	DS2	DS1	DS0
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

SPI

SPI Data Receive Register (SPDRR)

arch.h: ArchIO.Spi.DataRxReg
registers.h: ArchIO_Spi_DataRxReg

Receive (R15–R0)
Data read from SPDRR Data Receive Register shows the last full data received after a complete transmission.
Note 1: The SPRF bit in the SPI Status Control Register (SFSCR) clears automatically after reading SPDRR.
Note 2: The SPRF bit in the SPI Status Control Register (SPSCR) will set when new data has been transferred to this register.
Note 3: The SPI Transmitter Empty Bit (SPTE) in the SPSCR register indicates when the next write to register SPDRR can occur.

SPI Data Receive Register (SPDRR) SPI_BASE+$2	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	R15	R14	R13	R12	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reserved Bits

SPI

SPI Data Transmit Register (SPDTR)

arch.h: ArchIO.Spi.DataTxReg
registers.h: ArchIO_Spi_DataTxReg

Transmit (T15–T0)	
Data written to the SPDTR register is written to the transmit data buffer.	
Master Mode:	If new data is not written while in master mode, a new transaction will not begin until this register is written.
Slave Mode:	In slave mode, old data will be re-transmitted.
Note 1: Write all data with the LSB at bit 0.	
Note 2: Write new data to this register only when the SPTE bit in register SPSCR is set; otherwise, data may be lost.	
Note 3: Register SPDTR can only be written when the SPI is enabled, SPE = 1.	

SPI Data Transmit Register (SPDTR) SPI_BASE+$3	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read																
	Write	T15	T14	T13	T12	T11	T10	T9	T8	T7	T6	T5	T4	T3	T2	T1	T0
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reserved Bits

Application: _____ Date: _____

Programmer: _____

TMR

TMR Control Register (CTRL)

arch.h: ArchIO.Timer ***?***.Channel ***i***.ControlReg
registers.h: ArchIO_Timer ***?***_Channel ***i***_ControlReg
Where ? = A, B, C, or D *and* ***i*** = 0, 1, 2, or 3

Count Mode	Count Mode
000	No operation
001	Counts rising edges of primary source Note: Rising edges are counted only when IPS = 0. Falling edges are counted when IPS = 1.
010	Counts rising and falling edges of primary source
011	Counts rising edges of primary source while secondary input high active
100	Quadrature count mode; uses primary and secondary sources
101	Counts primary source rising edges; secondary source specifies direction Note: Rising Edges are counted only when IPS = 0. Falling edges are counted when IPS = 1.
110	Edge of secondary source triggers primary count until compare
111	Cascaded Counter mode (up/down)

TMR Control Register (CTRL)	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	COUNT MODE			PRIMARY COUNT SOURCE				SECONDARY SOURCE		ONCE	LENGTH	DIR	Co INIT	OUTPUT MODE		
	Write																
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TMRxn_BASE+$6, $E, $16, or $1E (x = A, B, C, or D; n = 0, 1, 2, 3)

TMRA0_CTRL (Timer A, Channel 0 Control)—Address: TMRA_BASE + $6
TMRA1_CTRL (Timer A, Channel 1 Control)—Address: TMRA_BASE + $E
TMRA2_CTRL (Timer A, Channel 2 Control)—Address: TMRA_BASE + $16
TMRA3_CTRL (Timer A, Channel 3 Control)—Address: TMRA_BASE + $1E

TMRB0_CTRL (Timer B, Channel 0 Control)—Address: TMRB_BASE + $6
TMRB1_CTRL (Timer B, Channel 1 Control)—Address: TMRB_BASE + $E
TMRB2_CTRL (Timer B, Channel 2 Control)—Address: TMRB_BASE + $16
TMRB3_CTRL (Timer B, Channel 3 Control)—Address: TMRB_BASE + $1E

TMRC0_CTRL (Timer C, Channel 0 Control)—Address: TMRC_BASE + $6
TMRC1_CTRL (Timer C, Channel 1 Control)—Address: TMRC_BASE + $E
TMRC2_CTRL (Timer C, Channel 2 Control)—Address: TMRC_BASE + $16
TMRC3_CTRL (Timer C, Channel 3 Control)—Address: TMRC_BASE + $1E

TMRD0_CTRL (Timer D, Channel 0 Control)—Address: TMRD_BASE + $6
TMRD1_CTRL (Timer D, Channel 1 Control)—Address: TMRD_BASE + $E
TMRD2_CTRL (Timer D, Channel 2 Control)—Address: TMRD_BASE + $16
TMRD3_CTRL (Timer D, Channel 3 Control)—Address: TMRD_BASE + $1E

See the following page for continuation of this register

TMR

TMR Control Register (CTRL) Continued

arch.h: ArchIO.Timer ***?***.Channel ***i***.ControlReg
registers.h: ArchIO_Timer ***?***_Channel ***i***_ControlReg
Where ? = A, B, C, or D *and* ***i*** = 0, 1, 2, or 3

Primary Count Source	Primary Count Source
The Primary Count Source bits select the primary count source.	
Counter #0 input pin	0000
Counter #1 input pin	0001
Counter #2 input pin	0010
Counter #3 input pin	0011
Counter #0 output	0100
Counter #1 output	0101
Counter #2 output	0110
Counter #3 output	0111
Prescaler (IPBus clock divide by 1)	1000
Prescaler (IPBus clock divide by 2)	1001
Prescaler (IPBus clock divide by 4)	1010
Prescaler (IPBus clock divide by 8)	1011
Prescaler (IPBus clock divide by 16)	1100
Prescaler (IPBus clock divide by 32)	1101
Prescaler (IPBus clock divide by 64)	1110
Prescaler (IPBus clock divide by 128)	1111
Note: A timer selecting its own output for input is not a legal choice. The result is no counting.	

Secondary Count Source	Secondary Count Source
Counter #0 input pin	00
Counter #1 input pin	01
Counter #2 input pin	10
Counter #3 input pin	11

ONCE	Count Once
0	**Count repeatedly**
1	Count until compare and then stop. If counting up, successful compare occurs when counter reaches CMP1 value. If counting down, successful compare occurs when counter reaches CMP2 value.

LENGTH	Count Length
0	Roll Over
1	Count until compare, then reinitialize. If counting up, successful compare occurs when counter reaches CMP1 value. If counting down, successful compare occurs when counter reaches CMP2 value. **Note:** When output mode $4 is used, alternating values of CMP1 and CMP2 are used to generate successful compares. For example, when output mode is $4, the counter counts until CMP1 value is reached, reinitializes, then counts until CMP2 value is reached, reinitializes, then counts until CMP1 value is reached, and so on.

TMR Control Register (CTRL)	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	COUNT MODE			PRIMARY COUNT SOURCE				SECONDARY SOURCE		ONCE	LENGTH	DIR	Co INIT	OUTPUT MODE		
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TMRxn_BASE+$6, $E, $16, or $1E (x = A, B, C, or D; n = 0, 1, 2, 3)

See the following page for continuation of this register

TMR

TMR Control Register (CTRL) Continued

arch.h: ArchIO.Timer ***?***.Channel ***i***.ControlReg
registers.h: ArchIO_Timer ***?***_Channel ***i***_ControlReg
Where ? = A, B, C, or D *and* ***i*** = 0, 1, 2, or 3

Count Direction	DIR
Count Up	0
Count Down	1

Co-Channel Initialization	CO INIT
Co-channel counter/timers can not force a reinitialization of this counter/timer	0
Co-channel counter/timers may force a reinitialization of this counter/timer	1

OUTPUT MODE	Output Mode
000	Asserted while counter is active
001	Clear OFLAG output on successful compare
010	Set OFLAG output on successful compare
011	Toggle OFLAG output on successful compare
100	Toggle OFLAG output using alternating compare registers
101	Set on compare, cleared on secondary source input edge
110	Set on compare, cleared on counter rollover
111	Enable Gated Clock output while counter is active **Note:** The Primary count source must be set to one of the counter outputs.

TMR Control Register (CTRL)	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read / Write	COUNT MODE			PRIMARY COUNT SOURCE				SECONDARY SOURCE		ONCE	LENGTH	DIR	Co INIT	OUTPUT MODE		
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TMRxn_BASE+$6, $E, $16, or $1E (x = A, B, C, or D; n = 0, 1, 2, 3)

TMR

TMR Status and Control Register (SCR)

arch.h: ArchIO.Timer ***?***.Channel ***i***.StatusControlReg
registers.h: ArchIO_Timer ***?***_Channel ***i***_StatusControlReg
Where ? = A, B, C, or D *and* ***i*** = 0, 1, 2, or 3

Timer Compare Flag	TCF
Compare has not yet occurred	0
A successful compare occurred	1

TOF	Timer Overflow Flag
0	Timer overflow has not occurred
1	The timer/counter rolled over to its maximum value $FFFF or $0000 (depending on count direction)

Timer Compare Flag Interrupt Enable	TCFIE
Timer compare interrupt is disabled	0
Interrupts enabled if the Timer Compare Flag, TCF, is set	1

TOFIE	Timer Overflow Flag Interrupt Enable
0	Timer overflow interrupt disabled
1	Timer overflow interrupt enabled if the Timer Overflow Flag, TOF, is also set.

TMR Status and Control Register (SCR)	Bits	15	14	13	12	11	10	9	8	7:	6	5	4	3	2	1	0
	Read	TCF	TCFIE	TOF	TOFIE	IEF	IEFIE	IPS	INPUT	CAPTURE MODE		MSTR	EEOF	VAL		OPS	OEN
	Write														FORCE		
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TMRxn_BASE+$7, $F, $17, or $1F (x = A, B, C, or D; n = 0, 1, 2, 3)

TMRA0_SCR (Timer A, Channel 0 Status and Control)—Address: TMRA_BASE + $7
TMRA1_SCR (Timer A, Channel 1 Status and Control)—Address: TMRA_BASE + $F
TMRA2_SCR (Timer A, Channel 2 Status and Control)—Address: TMRA_BASE + $17
TMRA3_SCR (Timer A, Channel 3 Status and Control)—Address: TMRA_BASE + $1F

TMRB0_SCR (Timer B, Channel 0 Status and Control)—Address: TMRB_BASE + $7
TMRB1_SCR (Timer B, Channel 1 Status and Control)—Address: TMRB_BASE + $F
TMRB2_SCR (Timer B, Channel 2 Status and Control)—Address: TMRB_BASE + $17
TMRB3_SCR (Timer B, Channel 3 Status and Control)—Address: TMRB_BASE + $1F

TMRC0_SCR (Timer C, Channel 0 Status and Control)—Address: TMRC_BASE + $7
TMRC1_SCR (Timer C, Channel 1 Status and Control)—Address: TMRC_BASE + $F
TMRC2_SCR (Timer C, Channel 2 Status and Control)—Address: TMRC_BASE + $17
TMRC3_SCR (Timer C, Channel 3 Status and Control)—Address: TMRC_BASE + $1F

TMRD0_SCR (Timer D, Channel 0 Status and Control)—Address: TMRD_BASE + $7
TMRD1_SCR (Timer D, Channel 1 Status and Control)—Address: TMRD_BASE + $F
TMRD2_SCR (Timer D, Channel 2 Status and Control)—Address: TMRD_BASE + $17
TMRD3_SCR (Timer D, Channel 3 Status and Control)—Address: TMRD_BASE + $1F

Reserved Bits

See the following page for continuation of this register

TMR

TMR Status and Control Register (SCR) Continued

arch.h: ArchIO.Timer ***?***.Channel ***i***.StatusControlReg
registers.h: ArchIO_Timer ***?***_Channel ***i***_StatusControlReg
Where ? = A, B, C, or D *and* ***i*** = 0, 1, 2, or 3

Input Edge Flag	IEF
An input edge transition has not occurred	0
A positive input edge transition occurred **Note 1:** Set the IPS bit to enable negative input edge transition detection. **Note 2:** The Secondary Count Source in the Control Register, CTRL, determines which external input pin is monitored.	1

Input Edge Flag Interrupt Enable	IEFIE
Input edge interrupts disabled	0
Input edge interrupts enabled when the IEF bit is set	1

Input Polarity Select	IPS
Input signal polarity is unchanged	0
Input signal polarity is inverted	1

INPUT	External Input Signal
0	External input pin polarity is unchanged
1	External input pin polarity is inverted

CAPTURE MODE	Input Capture Mode
Capture Mode specifies the operation of the Capture Register, CAP, and the Input Edge Flag, IEF.	
00	The Capture function is disabled, and the Input Edge Flag interrupts are disabled
01	Load the Capture register on a rising input edge
10	Load the Capture register on a falling input edge
11	Load the Capture register on both input edges

TMR Status and Control Register (SCR)	Bits	15	14	13	12	11	10	9	8	7:	6	5	4	3	2	1	0
	Read	TCF	TCFIE	TOF	TOFIE	IEF	IEFIE	IPS	INPUT	CAPTURE MODE		MSTR	EEOF	VAL		OPS	OEN
	Write														FORCE		
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TMRxn_BASE+$7, $F, $17, or $1F (x = A, B, C, or D; n = 0, 1, 2, 3)

Reserved Bits

See the following page for continuation of this register

TMR

TMR Status and Control Register (SCR) Continued

arch.h: ArchIO.Timer ***?***.Channel ***i***.StatusControlReg
registers.h: ArchIO_Timer ***?***_Channel ***i***_StatusControlReg

Where ? = A, B, C, or D *and* ***i*** = 0, 1, 2, or 3

Master Mode	MSTR
Disabled	0
Enabled	1

Enable External OFLAF Force	EEOF
Compare disabled	0
Enables a compare from another counter/timer	1

Forced OFLAG Value	VAL
VAL is disabled	0
VAL initiates a read of the OFLAG output signal value when a FORCE command is triggered by software or when a FORCE command is issued by a counter/timer set as a master.	1

FORCE	Force the OFLAG Output
0	This bit always reads as zero.
1	Forces the current value of the VAL bit to be written to the OFLAG output. (FORCE and VAL bits can be written in a single write operation.) **Note:** Write to the FORCE bit only if the counter is disabled. Setting this bit while the counter is enabled may yield unpredictable results.

OPS	Output Polarity Select
0	True polarity
1	Inverted polarity

OEN	Output Enable
0	OFLAG output signal is disabled
1	Enables the OFLAG output signal to be put on the external pin. Also connects a timer's output pin to its input.

TMR Status and Control Register (SCR)	Bits	15	14	13	12	11	10	9	8	7: 6	5	4	3	2	1	0
	Read	TCF	TCFIE	TOF	TOFIE	IEF	IEFIE	IPS	INPUT	CAPTURE MODE	MSTR	EEOF	VAL		OPS	OEN
	Write													FORCE		
	Reset	0	0	0	0	0	0	0	0	0 0	0	0	0	0	0	0

TMRxn_BASE+$7, $F, $17, or $1F (x = A, B, C, or D; n = 0, 1, 2, 3)

Reserved Bits

TMR

TMR Compare Register 1 (CMP1)

arch.h: ArchIO.Timer ***?***.Channel ***i***.CompareReg1
registers.h: ArchIO_Timer ***?***_Channel ***i***_CompareReg1

Where ? = A, B, C, or D *and* ***i*** = 0, 1, 2, or 3

Compare Register 1 (CMP1)
CMP1 stores the value used for comparison with the counter value.

TMR Compare Register #1 (CMP1)	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read / Write	COMPARISON VALUE[15:0]															
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TMRxn_BASE+$0, $8, $10, $18 (x = A, B, C, or D; n = 0, 1, 2, 3)

TMRA0_CMP1 (Timer A, Channel 0 Compare #1)—Address: TMRA_BASE + $0
TMRA1_CMP1 (Timer A, Channel 1 Compare #1)—Address: TMRA_BASE + $8
TMRA2_CMP1 (Timer A, Channel 2 Compare #1)—Address: TMRA_BASE + $10
TMRA3_CMP1 (Timer A, Channel 3 Compare #1)—Address: TMRA_BASE + $18

TMRB0_CMP1 (Timer B, Channel 0 Compare #1)—Address: TMRB_BASE + $0
TMRB1_CMP1 (Timer B, Channel 1 Compare #1)—Address: TMRB_BASE + $8
TMRB2_CMP1 (Timer B, Channel 2 Compare #1)—Address: TMRB_BASE + $10
TMRB3_CMP1 (Timer B, Channel 3 Compare #1)—Address: TMRB_BASE + $18

TMRC0_CMP1 (Timer C, Channel 0 Compare #1)—Address: TMRC_BASE + $0
TMRC1_CMP1 (Timer C, Channel 1 Compare #1)—Address: TMRC_BASE + $8
TMRC2_CMP1 (Timer C, Channel 2 Compare #1)—Address: TMRC_BASE + $10
TMRC3_CMP1 (Timer C, Channel 3 Compare #1)—Address: TMRC_BASE + $18

TMRD0_CMP1 (Timer D, Channel 0 Compare #1)—Address: TMRD_BASE + $0
TMRD1_CMP1 (Timer D, Channel 1 Compare #1)—Address: TMRD_BASE + $8
TMRD2_CMP1 (Timer D, Channel 2 Compare #1)—Address: TMRD_BASE + $10
TMRD3_CMP1 (Timer D, Channel 3 Compare #1)—Address: TMRD_BASE + $18

TMR

TMR Compare Register 2 (CMP2)

arch.h: ArchIO.Timer ***?***.Channel ***i***.CompareReg2
registers.h: ArchIO_Timer ***?***_Channel ***i***_CompareReg2

Where ? = A, B, C, or D *and* ***i*** = 0, 1, 2, or 3

Compare Register 2 (CMP2)
CMP2 stores the value used for comparison with the counter value.

TMR Compare Register #2 (CMP2)	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read / Write	COMPARISON VALUE[15:0]															
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TMRxn_BASE+$1, $9, $11, or $19 (x = A, B, C, or D; n = 0, 1, 2, 3)

TMRA0_CMP2 (Timer A, Channel 0 Compare #2)—Address: TMRA_BASE + $1
TMRA1_CMP2 (Timer A, Channel 1 Compare #2)—Address: TMRA_BASE + $9
TMRA2_CMP2 (Timer A, Channel 2 Compare #2)—Address: TMRA_BASE + $11
TMRA3_CMP2 (Timer A, Channel 3 Compare #2)—Address: TMRA_BASE + $19

TMRB0_CMP2 (Timer B, Channel 0 Compare #2)—Address: TMRB_BASE + $1
TMRB1_CMP2 (Timer B, Channel 1 Compare #2)—Address: TMRB_BASE + $9
TMRB2_CMP2 (Timer B, Channel 2 Compare #2)—Address: TMRB_BASE + $11
TMRB3_CMP2 (Timer B, Channel 3 Compare #2)—Address: TMRB_BASE + $19

TMRC0_CMP2 (Timer C, Channel 0 Compare #2)—Address: TMRC_BASE + $1
TMRC1_CMP2 (Timer C, Channel 1 Compare #2)—Address: TMRC_BASE + $9
TMRC2_CMP2 (Timer C, Channel 2 Compare #2)—Address: TMRC_BASE + $11
TMRC3_CMP2 (Timer C, Channel 3 Compare #2)—Address: TMRC_BASE + $19

TMRD0_CMP2 (Timer D, Channel 0 Compare #2)—Address: TMRD_BASE + $1
TMRD1_CMP2 (Timer D, Channel 1 Compare #2)—Address: TMRD_BASE + $9
TMRD2_CMP2 (Timer D, Channel 2 Compare #2)—Address: TMRD_BASE + $11
TMRD3_CMP2 (Timer D, Channel 3 Compare #2)—Address: TMRD_BASE + $19

TMR

TMR Capture Register (CAP)

arch.h: ArchIO.Timer ***?***.Channel ***i***.CaptureReg
registers.h: ArchIO_Timer ***?***_Channel ***i***_CaptureReg

Where ? = A, B, C, or D *and* ***i*** = 0, 1, 2, or 3

Capture Register (CAP)
CAP stores the value captured from the counter.

TMR Capture Register (CAP)	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	CAPTURE															
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TMRxn_BASE+$2, $A, $12 or $1A (x = A, B, C, or D; n = 0, 1, 2, 3)

TMRA0_CAP (Timer A, Channel 0 Capture)—Address: TMRA_BASE + $2
TMRA1_CAP (Timer A, Channel 1 Capture)—Address: TMRA_BASE + $A
TMRA2_CAP (Timer A, Channel 2 Capture)—Address: TMRA_BASE + $12
TMRA3_CAP (Timer A, Channel 3 Capture)—Address: TMRA_BASE + $1A

TMRB0_CAP (Timer B, Channel 0 Capture)—Address: TMRB_BASE + $2
TMRB1_CAP (Timer B, Channel 1 Capture)—Address: TMRB_BASE + $A
TMRB2_CAP (Timer B, Channel 2 Capture)—Address: TMRB_BASE + $12
TMRB3_CAP (Timer B, Channel 3 Capture)—Address: TMRB_BASE + $1A

TMRC0_CAP (Timer C, Channel 0 Capture)—Address: TMRC_BASE + $2
TMRC1_CAP (Timer C, Channel 1 Capture)—Address: TMRC_BASE + $A
TMRC2_CAP (Timer C, Channel 2 Capture)—Address: TMRC_BASE + $12
TMRC3_CAP (Timer C, Channel 3 Capture)—Address: TMRC_BASE + $1A

TMRD0_CAP (Timer D, Channel 0 Capture)—Address: TMRD_BASE + $2
TMRD1_CAP (Timer D, Channel 1 Capture)—Address: TMRD_BASE + $A
TMRD2_CAP (Timer D, Channel 2 Capture)—Address: TMRD_BASE + $12
TMRD3_CAP (Timer D, Channel 3 Capture)—Address: TMRD_BASE + $1A

TMR

TMR Load Register (LOAD)

arch.h: ArchIO.Timer ***?***.Channel ***i***.LoadReg
registers.h: ArchIO_Timer ***?***_Channel ***i***_LoadReg

Where ? = A, B, C, or D *and* ***i*** = 0, 1, 2, or 3

Load Register (LOAD)
LOAD stores the value used to initialize the counter/timer.

TMR Load Register (LOAD)	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	LOAD[15:0]															
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TMRxn_BASE+$3, $B, $13, or $1B (x = A, B, C, or D; n = 0, 1, 2, 3)

TMRA0_LOAD (Timer A, Channel 0 Load)—Address: TMRA_BASE + $3
TMRA1_LOAD (Timer A, Channel 1 Load)—Address: TMRA_BASE + $B
TMRA2_LOAD (Timer A, Channel 2 Load)—Address: TMRA_BASE + $13
TMRA3_LOAD (Timer A, Channel 3 Load)—Address: TMRA_BASE + $1B

TMRB0_LOAD (Timer B, Channel 0 Load)—Address: TMRB_BASE + $3
TMRB1_LOAD (Timer B, Channel 1 Load)—Address: TMRB_BASE + $B
TMRB2_LOAD (Timer B, Channel 2 Load)—Address: TMRB_BASE + $13
TMRB3_LOAD (Timer B, Channel 3 Load)—Address: TMRB_BASE + $1B

TMRC0_LOAD (Timer C, Channel 0 Load)—Address: TMRC_BASE + $3
TMRC1_LOAD (Timer C, Channel 1 Load)—Address: TMRC_BASE + $B
TMRC2_LOAD (Timer C, Channel 2 Load)—Address: TMRC_BASE + $13
TMRC3_LOAD (Timer C, Channel 3 Load)—Address: TMRC_BASE + $1B

TMRD0_LOAD (Timer D, Channel 0 Load)—Address: TMRD_BASE + $3
TMRD1_LOAD (Timer D, Channel 1 Load)—Address: TMRD_BASE + $B
TMRD2_LOAD (Timer D, Channel 2 Load)—Address: TMRD_BASE + $13
TMRD3_LOAD (Timer D, Channel 3 Load)—Address: TMRD_BASE + $1B

TMR

TMR Hold Register (HOLD)

arch.h: ArchIO.Timer ***?***.Channel ***i***.HoldReg
registers.h: ArchIO_Timer ***?***_Channel ***i***_HoldReg

Where ? = A, B, C, or D *and* ***i*** = 0, 1, 2, or 3

Hold Register (HOLD)
HOLD stores the specific channel's counter value when reading any of the four counters within a module.

TMR Hold Register (HOLD)	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	HOLD [15:0]															
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TMRxn_BASE+$4, $C, $14, or $1C (x = A, B, C, or D; n = 0, 1, 2, 3)

TMRA0_HOLD (Timer A, Channel 0 Hold)—Address: TMRA_BASE + $4
TMRA1_HOLD (Timer A, Channel 1 Hold)—Address: TMRA_BASE + $C
TMRA2_HOLD (Timer A, Channel 2 Hold)—Address: TMRA_BASE + $14
TMRA3_HOLD (Timer A, Channel 3 Hold)—Address: TMRA_BASE + $1C

TMRB0_HOLD (Timer B, Channel 0 Hold)—Address: TMRB_BASE + $4
TMRB1_HOLD (Timer B, Channel 1 Hold)—Address: TMRB_BASE + $C
TMRB2_HOLD (Timer B, Channel 2 Hold)—Address: TMRB_BASE + $14
TMRB3_HOLD (Timer B, Channel 3 Hold)—Address: TMRB_BASE + $1C

TMRC0_HOLD (Timer C, Channel 0 Hold)—Address: TMRC_BASE + $4
TMRC1_HOLD (Timer C, Channel 1 Hold)—Address: TMRC_BASE + $C
TMRC2_HOLD (Timer C, Channel 2 Hold)—Address: TMRC_BASE + $14
TMRC3_HOLD (Timer C, Channel 3 Hold)—Address: TMRC_BASE + $1C

TMRD0_HOLD (Timer D, Channel 0 Hold)—Address: TMRD_BASE + $4
TMRD1_HOLD (Timer D, Channel 1 Hold)—Address: TMRD_BASE + $C
TMRD2_HOLD (Timer D, Channel 2 Hold)—Address: TMRD_BASE + $14
TMRD3_HOLD (Timer D, Channel 3 Hold)—Address: TMRD_BASE + $1C

TMR

TMR Counter Register (CNTR)

arch.h: ArchIO.Timer ***?***.Channel ***i***.CounterReg
registers.h: ArchIO_Timer ***?***_Channel ***i***_CounterReg

Where ? = A, B, C, or D *and* ***i*** = 0, 1, 2, or 3

Counter Register (CNTR)
CNTR is the counter for the corresponding channel in a timer module.

TMR Counter Register (CNTR)	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	COUNTER[15:0]															
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TMRxn_BASE+$5, $d, $15, or $1D (x = A, B, C, or D; n = 0, 1, 2, 3)

TMRA0_CNTR (Timer A, Channel 0 Cntr)—Address: TMRA_BASE + $5
TMRA1_CNTR (Timer A, Channel 1 Cntr)—Address: TMRA_BASE + $D
TMRA2_CNTR (Timer A, Channel 2 Cntr)—Address: TMRA_BASE + $15
TMRA3_CNTR (Timer A, Channel 3 Cntr)—Address: TMRA_BASE + $1D

TMRB0_CNTR (Timer B, Channel 0 Cntr)—Address: TMRB_BASE + $5
TMRB1_CNTR (Timer B, Channel 1 Cntr)—Address: TMRB_BASE + $D
TMRB2_CNTR (Timer B, Channel 2 Cntr)—Address: TMRB_BASE + $15
TMRB3_CNTR (Timer B, Channel 3 Cntr)—Address: TMRB_BASE + $1D

TMRC0_CNTR (Timer C, Channel 0 Cntr)—Address: TMRC_BASE + $5
TMRC1_CNTR (Timer C, Channel 1 Cntr)—Address: TMRC_BASE + $D
TMRC2_CNTR (Timer C, Channel 2 Cntr)—Address: TMRC_BASE + $15
TMRC3_CNTR (Timer C, Channel 3 Cntr)—Address: TMRC_BASE + $1D

TMRD0_CNTR (Timer D, Channel 0 Cntr)—Address: TMRD_BASE + $5
TMRD1_CNTR (Timer D, Channel 1 Cntr)—Address: TMRD_BASE + $D
TMRD2_CNTR (Timer D, Channel 2 Cntr)—Address: TMRD_BASE + $15
TMRD3_CNTR (Timer D, Channel 3 Cntr)—Address: TMRD_BASE + $1D

OCCS

PLL Control Register (PLLCR)

arch.h: ArchIO.Pll.ControlReg
registers.h: ArchIO_Pll_ControlReg

PLL Interrupt Enable 1	PLLIE1
Disable interrupt	**00**
Enable interrupt on any rising edge of LCK1	01
Enable interrupt on falling edge of LCK1	10
Enable interrupt on any edge change of LCK1	11

PLL Interrupt Enable 0	PLLIE0
Disable interrupt	00
Enable interrupt on any rising edge of LCK0	01
Enable interrupt on falling edge of LCK0	10
Enable interrupt on any edge change of LCK0	11

Loss of Clock Interrupt Enable	LOCIE
Interrupt disabled	0
Interrupt enabled	1

Lock Detector On	LCKON
Lock detector disabled	0
Lock detector enabled	1

Charge Pump Tri-state	CHPMPTRI
Normal chip operation	0
In the event of *Loss of Reference* (Clock: FREF), set bit CHPMPTRI to 1. **Note:** Activating this bit renders the PLL inoperable and should not be done during standard chip operation.	1

PLLPD	PLL Power Down
0	PLL enabled
1	PLL powered down **Note:** When PLL is powered down, ZSRC[2:0] is switched to this value to prevent a loss of clock to the core.

PRECS*	Prescaler Clock Select *Applicable for 56F801 Only
0	Relaxation oscillator selected (reset value)
1	Crystal oscillator selected. (The crystal oscillator must be enabled in the GPIO to set this bit.)

*PRECS is not available on 56F803/805/807. It is Reserved on these registers.

ZSRC	ZCLOCK Source
01	Prescaler output **Note:** ZSRC is automatically set to this value when PLLPD is set or when PLL enters STOP_MODE preventing loss of clock to the core.
10	Postscaler output
00, 11	Reserved

PLL Control Register (PLLCR) CLKGEN_BASE+$0	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	PLLIE1		PLLIE0		LOCIE	0	0	0	LCKON	CHPMPTRI	0	PLLPD	0	PRECS*	ZSRC[1:0]	
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1

Reserved Bits

OCCS

PLL Divide-By Register (PLLDB)

arch.h: ArchIO.Pll.DivideReg
registers.h: ArchIO_Pll_DivideReg

Loss of Reference Timer Period	LORTP
t = LORTP x 10 x (PLL-Clock-Time-Period) where t = Time to Generate Loss of Reference Interrupt	0 – 15
Note: LORTP is not available in 56F805-A.	

PLL Clock Out Divide (Postscaler)	PLLCOD
Divide by 1	00
Divide by 2	01
Divide by 4	10
Divide by 8	11

PLL Clock In Divide (Prescaler)	PLLCID
Divide by 1	00
Divide by 2	01
Divide by 4	10
Divide by 8	11

PLLDB	PLL Divide-By
0 – 127	PLL Output Frequency = Fout fout = $\frac{FREF \times (n+1)}{2}$ where FREF = Reference Frequency = (XTAL Clock) / PLLCID[1:0] n + 1 = Scaling Value n = value of PLLDB[6:0] $0 \le n \le 127$

Note 1: Before changing the divide-by value, it is recommended that the core clock be switched to the Prescaler Clock.

Note 2: The value for n should be programmed so fout is in the normal operating range, 80 – 160 MHz .

PLL Divide-By Register (PLLDB) CLKGEN_BASE+$1	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	LORTP[3:0]				PLLCOD[1:0]		PLLCID[1:0]		0	PLLDB[6:0]						
	Write																
	Reset	0	0	1	0	0	0	0	0	0	0	0	1	0	0	1	1

Reserved Bits

OCCS

PLL Status Register (PLLSR)

arch.h: ArchIO.Pll.StatusReg
registers.h: ArchIO_Pll_StatusReg

PLL Loss of Lock Interrupt 1	LOLI1
No interrupt pending	0
Interrupt pending (See also PLLIE1)	1
Note: To clear this bit, write a 1 to LOLI1.	

PLL Loss of Lock Interrupt 0	LOLI0
No interrupt pending	0
Interrupt pending (See also PLLIE0)	1
Note: To clear this bit, write a 1 to LOLI0.	

Loss of Clock	LOCI
PLL reference clock normal (See also LOCKIE)	0
Lost PLL reference clock	1

Loss of Lock 1	LCK1
PLL is unlocked (fine)	0
PLL is locked (fine)	1

Loss of Lock 0	LCK0
PLL is unlocked (course)	0
PLL is locked (course)	1

PLLPDN	PLL Power Down
0	PLL not powered down
1	PLL powered down
Note: The PLL power down status is delayed by four IPBus Clocks from the PLLPD bit in the PLLCR.	

PRECSS*	Prescaler Clock Select Status— *Applicable for 56F801 Only
0	Relaxation oscillator clock
1	Crystal oscillator clock
Note: PRECSS takes more than one IPBus clock to indicate a changeover to a new clock.	

*PRECS is not available on 56F803/805/807. It is Reserved on these registers.

ZSRC	ZCLOCK Source
00	Synchronizing in progress
01	Prescaler output
10	Postscaler output
11	Synchronizing in progress
Note: ZSRC takes more than one IPBus clock to indicate a new selection.	

PLL Status Register (PLLSR) CLKGEN_BASE+$2	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	LOLI1	LOLI0	LOCI	0	0	0	0	0	0	LCK1	LCK0	PLLPDN	0	PRECSS*	ZSRC[1:0]	
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1

Reserved Bits

OCCS

PLL Select Register (CLKOSR)

arch.h: ArchIO.Pll.SelectReg
registers.h: ArchIO_Pll_SelectReg

CLKOSEL	CLKO Select
Selects clock to be “multiplexed out” on the CLKO pin.	
10000	No Clock
00000	ZCLK (Default)
00001	Reserved for factory test – t0
00010	Reserved for factory test – t1
00011	Reserved for factory test – t2
00100	Reserved for factory test – t3
00101	Reserved for factory test – phi0
00110	Reserved for factory test – phi1
00111	Reserved for factory test – ctzn
01000	Reserved for factory test – ct301en
01001	Reserved for factory test – 1PB clock
01010	Reserved for factory test – Feeback
01011	Reserved for factory test – Prescaler Clk
01100	Reserved for factory test – Fout
01101	Reserved for factory test – Fout/2
01110	Reserved for factory test – Postscaler Clk
01111	Reserved for factory test – Postscaler Clk

CLKO Select Register (CLKOSR) CLKGEN_BASE+$4	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	0	0	0	0	0	0	0	CLKOSEL[0:4]				
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reserved Bits

OCCS

PLL Internal Oscillator Control Register (IOSCTL)*

arch.h: ArchIO.Pll.InternalOscReg
registers.h: ArchIO_Pll_InternalOscReg

TRIM	Internal Relaxation Oscillator Trim Factor
0 – 255	TRIM adjusts the accuracy of the internal relaxation oscillator clock to 2% by changing the size of the capacitor used by the internal relaxation oscillator. To decrease the frequency by 0.23% increment TRIM by one. To increase the frequency by 0.23% decrement TRIM by one. Reset centers the adjustment range at $80.

Internal Oscillator Control Register (IOSCTL) CLKGEN_BASE+$5	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	0	0	0	0	TRIM[7:0]							
	Write																
	Reset	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

Reserved Bits

***IOSCTL** is available only on the 801 and 802 devices. It is Reserved and not available on 803/805/807.

Application: _____ Date: _____

Programmer: _____

COP

COP Control Register (COPCTL)

arch.h: ArchIO.Cop.ControlReg
registers.h: ArchIO_Cop_ControlReg

Stop Enable	CSEN
COP counter stops in Stop mode	0
COP counter runs in Stop mode	1
Note: This bit can only be changed when the CWP bit is set to zero. For the COP to run in Stop mode, the CEN bit must also be set.	

CEN	COP Enable
0	COP is disabled
1	COP is enabled
Note: This bit can only be changed when CWP is set to zero.	

COP Wait Enable	CWEN
COP counter will Stop in Wait mode	0
COP counter will run in Wait mode	1
Note: This bit can only be changed when the CWP bit is set to zero. For the COP to run in Stop mode, the CEN bit must also be set.	

CWP	COP Write Protect
0	COPCTL, COPTO registers are readable and writeable
1	COPCTL, COPTO registers are read-only
Note: Once this bit has been written to a 1, it cannot be changed back to 0 without resetting the COP module.	

COP Control Register (COPCTL) COP_BASE+$0	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	0	0	0	0	0	0	0	0	CSEN	CWEN	CEN	CWP
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

Reserved Bits

COP

COP Timeout (COPTO) & COP Service (COPSRV) Registers

arch.h: ArchIO.Cop.TimeoutReg
registers.h: ArchIO_Cop_TimeoutReg

arch.h: ArchIO.Cop.ServiceReg
registers.h: ArchIO_Cop_ServiceReg

CT	COP Timeout
0 – 4095	COP Timeout Period = [16384 x (CT[11:0] + 1) clock cycles Write the CT[11:0] value before the COP is enabled. (The COP is enabled in the COPCTL register.)

Note 1: Changing CT[11:0] while the COP is enabled will result in a timeout period that differs from the expected value given by the formula above.

Note 2: These bits can only be changed when the CWP bit in COPCTL is set to zero.

COP Timeout Register (COPTO) COP_BASE+$1	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	CT[11:0]											
	Write																
	Reset	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1

Service Sequence Write Values	COP Service Register
Write $5555 then Write $AAAA	COPSRV performs a periodic service sequence to clear the COP counter and prevent a reset. To perform a service sequence: 1. Write the value $5555 to COPSRV. An indefinite number of instructions may be executed before the second write. 2. Write the value $AAAA to COPSRV before the selected time-out period (as set in the COPTO register) expires.

Note: The writes to COPSRV must be performed in the correct order BEFORE the counter times out.

COP Service Register (COPSRV) COP_BASE+$2	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Write	COPSRV															
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reserved Bits

SIM

SIM Register (SYS_CNTL)

arch.h: ArchIO.Sim.ControlReg
registers.h: ArchIO_Sim_ControlReg

Timer I/O Pull-Up Disable	TMR PD
Pull ups for the timer I/O pins are enabled	0
Pull ups for the timer I/O pins are disabled	1

Control Signal Pull-Up Disable	CTRL PD
Pull ups for the $\overline{DS}$, $\overline{PS}$, $\overline{RD}$, and the $\overline{WR}$ I/O pins are enabled	0
Pull ups for the $\overline{DS}$, $\overline{PS}$, $\overline{RD}$, and the $\overline{WR}$ I/O pins are disabled	1

Address Bus [5:0] Pull-Up Disable	ADR PD
Pull ups for the lower address I/O pins are enabled	0
Pull ups for the lower address I/O pins are disabled	1
Note: The pull ups for the upper address pins [15:6] are controlled by the GPIO A and GPIO E modules.	

Data Bus I/O Pull-Up Disable	DATA PD
Pull ups for the data bus I/O pins are enabled	0
Pull ups for the data bus I/O pins are disabled	1

Bootmap (BOOTMAP_B)				
Mode No.	Bit Values			Description
	MA	MB	BOOTMAP_B	
0A	0	0	0	Boot Mode
0B	0	0	1	1st. 32K memory is internal 2nd. 32K memory is external
1	0	1	x	Reserved
2	1	0	x	Reserved
3	1	1	x	Development–64K Ext ROM
Note 1: Modes 0A and 0B are sub-modes of mode 0.				
Note 2: For details, see Table 3-38.				

LVIE27	2.7 V Low Voltage Interrupt Enable
0	2.7 V low voltage interrupt disabled
1	2.7 V low voltage interrupt enabled

LVIE22	2.2 V Low Voltage Interrupt Enable
0	2.2 V low voltage interrupt disabled
1	2.2 V low voltage interrupt enabled

PD	Permanent STOP/WAIT Disable
0	STOP and WAIT instructions enabled
1	STOP and WAIT instructions act as NOPs
Note: The part must be reset to clear this bit	

RPD	Re-Programmable STOP/WAIT
0	STOP and WAIT instructions enabled
1	STOP and WAIT instructions act as NOPs
Note: Software sets and clears this bit	

SIM Register (SYS_CNTL) SYS_BASE +$0	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	TMR PD	CTRL PD	ADR PD	DATA PD	0	0	0	BOOT MAP_B	LVIE27	LVIE22	PD	RPD
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reserved Bits

SIM

SIM Status Register (SYS_STS)

arch.h: ArchIO.Sim.StatusReg
registers.h: ArchIO_Sim_StatusReg

COP Reset	COPR
A COP timer generated system reset did not occur	0
A COP timer generated system reset has occurred	1
Note: A Power -n Reset will clear this bit. To clear this bit with software, write a one to this bit. (To set, this bit, write a zero to this bit. Setting this bit will not cause a reset operation.)	

POR	Power-On Reset
0	A Power-On Reset did not occur.
1	A Power-On Reset occurred some time in the past
Note: To clear this bit, write a one to this bit. (To set, this bit, write a zero to this bit. Setting this bit will not cause a reset operation.)	

LVIS27	2.7 V Low Voltage Interrupt Source
0	A Low Voltage Interrupt has not occurred.
1	A 2.7 V Low Voltage Interrupt is active, and the chip voltage has dropped below 2.7 V.
Note: To clear this bit, write a one to this bit. To set, this bit, write a zero to this bit. *Setting this bit will cause an interrupt if the corresponding interrupt enable bit is set.*	

External Reset	EXTR
An external system reset did not occur.	0
An external system reset has occurred. The previous system reset was caused by the external $\overline{\text{RESET}}$ pin being asserted low.	1
Note: A Power-On Reset will clear this bit. To clear this bit with software, write a one to this bit. (To set, this bit, write a zero to this bit. Setting this bit will not cause a reset operation.)	

LVIS22	2.2 Low Voltage Interrupt Source
0	A Low Voltage Interrupt has not occurred.
1	A 2.2 V Low Voltage Interrupt is active, and the chip voltage has dropped below 2.2 V.
Note: To clear this bit, write a one to this bit. To set, this bit, write a zero to this bit. *Setting this bit will cause an interrupt if the corresponding interrupt enable bit is set.*	

System Status Register (SYS_STS) SYS_BASE +$1	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	0	0	0	0	0	0	0	COPR	EXTR	POR	LVIS27	LVIS22
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reserved Bits

SIM

SIM Most Significant Half of JTAG ID Register (MSH_ID)

Least Significant Half of JTAG ID Register (LSH_ID)

Representative Values Shown In Registers
The representative values shown in the MSH_ID and LSH_ID registers are for the 56F805 chip, version 1.

Part Number[9:0]		
Binary Value	Decimal Equivalent	Part Number
1100100001	801	56F801
1100100011	803	56F803
1100100101	805	56F805
1100100111	807	56F807
Note: For version 0 of the 56F803 and 56F805 chips, registers MSH_ID, and LSH_ID are not available and always read zero.		

Most Significant Half of JTAG_ID (MSH_ID) SYS_BASE +$6	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	1	0	0	0	1	1	1	1	1	0	0	1	0
	Write																
	Reset	0	0	0	1	0	0	0	1	1	1	1	1	0	0	1	0

The MSH_ID register values are hardcoded and cannot be reset.

Least Significant Half of JTAG_ID (LSH_ID) SYS_BASE +$7	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	1	0	0	0	1	1	1	1	1	0	0	1	0
	Write																
	Reset	0	0	0	1	0	0	0	1	1	1	1	1	0	0	1	0

The LSH_ID register values are hard coded and cannot be reset.

Reserved Bits

SIM

SIM Operating Mode Register (OMR)

arch.h: OMR
registers.h: OMR

Nested Looping	NL
Nested Looping Not Allowed	0
Nested Looping Allowed	1

Looping Status		
DO Loop Status	LF in SR	NL in OMR
No DO Loops Active	0	0
Single DO Loop Active	1	0
Caution–Illegal combination. A hardware stack overflow interrupt will occur.	0	1
Two DO Loops Active	1	1

Condition Code	CC
Codes not set in CCR Register	0
Codes set in CCR Register	1

Stop Delay	SD
Stop Delay Enabled	0
Stop Delay Disabled	1

R	Rounding
0	Rounding Not Allowed
1	Rounding Allowed

SA	Saturation
0	Saturation Mode Disabled
1	Saturation Mode Enalbed

EX	External X Memory
0	External X Memory Disabled
1	External X Memory Disabled

MB	MA	Operating Mode
0	0	Mode 0 - Single Chip
0	1	Mode 1 - Not Supported
1	0	Mode 2 - Not Supported
1	1	Mode 3 - External Memory

MB	MA	Operating Mode
0	0	Mode 0 - Single Chip
0	1	Mode 1 - Not Supported
1	0	Mode 2 - Not Supported
1	1	Mode 3 - External Memory

Operating Mode Register (OMR) (CPU Register)	Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Read	0	0	0	0	0	0	0	CC	0	SD	R	SA	EX	0	MB	MA
	Write																
	Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reserved Bits

Glossary

This glossary is intended to reduce confusion that may be caused by the use of many acronyms and abbreviations throughout this manual.

ACIM	A/C Induction Motors
A/D	Analog-to-Digital
ADC	Analog to Digital Converter
ADCR	ADC Control Registe
ADDR	Address
ADHLMT	ADC High Limit Registers
ADLLMT	ADC Low Limit Registers
ADLST	ADC Channel List Registers
ADLSTAT	ADC Limit Status Register
ADM	Application Development Module
ADOFS	ADC Offset Registers
ADR PD	Address Bus Pull-up Disable
ADRSLT	ADC Result Registers
ADSDIS	ADC Sample Disable Register
ADSTAT	ADC Status Register
ADZCC	ADC Zero Crossing Control Register
ADZCSTAT	ADC Zero Crossing Status Register
AGU	Address Generation Unit
ALU	Arithmetic Logic Unit
API	Application Program Interface
Barrel Shifter	Part of the ALU that allows single cycle shifting and rotating of data word
BCR	Bus Control Register
BDC	Brush DC Motor
BE	Breakpoint Enable
BFIU	Boot Flash Interface Unit

BFLASH	Boot Flash
BK	Breakpoint Configuration Bit
BLDC	Brushless DC Motor
BOTNEG	Bottom-side PWM Polarity Bit
BS	Breakpoint Selection
BSDL	Boundary Scan Description Language
BSR	Boundary Scan Register
CAN	Controller Area Network
CC	Condition Codes
CAP	Capture
CEN	COP Enable Bit
CFG	Config
CGDB	Core Global Data Bus
CHCNF	Channel Configure
CID	Chip Identification Register
CKDIVISOR	Clock Divisor
CLKO	Clock Output pin
CLKOSEL	CLKO Select
CLKOSR	Clock Select Register
CMOS	Complementary metal oxide semiconductor. (A form of digital logic that is characterized by low power consumption, wide power suppply range, and high noise immunity.)
CMP	Compare
CNT	Count
CNTR	Counter
Codec	Coder/Decoder
COP	Computer Operating Properly
COP/RTI	Computer Operating Properly/Real Time Interface
COPCTL	COP Control
COPDIS	COP Timer Disable
COPR	COP Reset
COPSRV	COP Service
COPTO	COP Time Out
CPHA	Clock Phase

CPOL	Clock Polarity
CPU	Central Processing Unit
CRC	Cyclic Redundancy Code
CSEN	Cop Stop Enable
CTRL	Control
CTRL PD	Control signal Pull-up Disable
CWEN	COP Wait Enable Bit
CWP	COP Write Protect
DAC	Digital to Analog Converter
DAT	Data/Address Select
DATA ALU	Data Address Limit
DATA PD	Data bus I/O Pull-up Disable
DDA	Analog Power
DDR	Data Direction Register
DEC	Quadrature Decoder Module
DEE	Dumb Erase Enable
DFIU	Data Flash Interface Unit
DFLASH	Data Flash
DIE	Watchdog Time-Out Interrupt Enable
DIRQ	Watchdog Time-Out Interrupt Request
DPE	Dumb Programming Enable
DR	Data Register
DRV	Drive Control Bit
DSC	Digital Signal Controller
DSO	Data Shift Order
DSP	Digital Signal Processor
EDG	Edge-Aligned or Center-Aligned PWMs
EE	Erase Enable
EEOF	Enable External OFLAG Force
EM	Event Modifier
EN	Enable3
ENCR	Encoder Control Register

EOSI End of Scan Interrupt

EOSIE End of Scan Interrupt Enable

ERASE Erase Cycle

ERRIE Error Interrupt Enable

EX External X Memory

EXTBOOT External Boot

EXTR External Reset

FAULT Fault Input to PWM

FE Framing Error Flag

FFLAGx FAULTx Pin Flag

FH FIFO Halt

FIEx Faultx Pin Interrupt Enable

FIR Filter Interval Register

FIU Flash Interface Unit

Flash memory Nonvolatile memory, electronically erasable and programmable

FLOCI Force Loss of Clock

FLOLI Force Loss of Lock

FMODEx FAULTx Pin Clearing Mode

FPINx FAULTx Pin

FTACKx FAULTx Pin Acknowledge

GPIO General Purpose Input/Output

GPIO_IPR General Purpose Input/Output Interrupt Pending Register

GPR Group Priority Register

Harvard architecture - A microprocessor architecture that uses separate busses for program and data. This is typically used on DSPs to optimise the data throughput

HBO Hardware Breakpoint Occurrence

HLMTI High Limit Interrupt

HLMTIE High Limit Interrupt Enable

HOLD Hold Register

HOME Home Switch Input

IA Interrupt Assert

IC Integrated Circuit

IE Interrupt Enable

IEE	Intelligent Erase Enable
IEF	Input Edge Flag
IEFIE	Input Edge Flag Interrupt Enable
IENR	Interrupt Enable Register
IES	Interrupt Edge Sensitive
IFREN	Information Block Enable
IMR	Input Monitor Register
INDEP	Independent or Complimentary Pair Operation
INDEX	Index Input
INPUT	External Input Signal
INV	Invert
I/O	Input/Output
IP	Interrupt Pending
IPBus	Intellectual Properties Bus
IPE	Intelligent Program Enable
IPOL	Current Polarity
IPOLR	Interrupt Polarity Register
IPR	Interrupt Priority Register (in the Core)
IPS	Input Polarity Select
IRQ	Interrupt Request
IR	Instruction Register
IS	Interrupt Source
ITCN	Interrupt Controller
JTAG	Joint Test Action Group
JTAGBR	JTAG Bypass Register
JTAGIR	JTAG Instruction Register
LCD	Liquid Crystal Display
LCK	Loss of Lock
LDOK	Load OKay
LF	Loop Flag
LIR	Lower Initialization Register
LLMTI	Low Limit Interrupt

LLMTIE	Low Limit Interrupt Enable
LOAD	Load Register
LOCI	Loss of Clock
LOCIE	Los of Clock Interrupt Enable
LOLI	PLL Lock of Lock Interrupt
LOOP	Loop Select Bit
LPOS	Lower Position Counter Register
LPOSH	Lower Position Hold Register
LSB	Least Significant Bit
LSH_ID	Most Significant Half of JTAG_ID
LVD	Low Voltage Detect
LVIE	Low Voltage Interrupt Enable
LVIS	ow Voltage Interrupt Source
MA	Mode A
MAC	Multiply and Accumulate
MAS	Mass Cycle Erase
MB	Mode B
MCU	Microcontroller Unit
MHz	Megahertz
MIPS	Million Instructions Per Second
MISO	Master In/Slave Out
MODF	Mode Fault Error
MODFEN	Mode Fault Enable
MOSI	Master Out/Slave In
MPIO	Multi-Purpose Input/Output (A, B, C, D, E or F)
MSB	Most Significant Bit
MSH_ID	Most Significant Half of JTAG ID
MSTR	Master Mode
MUX	Multiplexer
NF	Noise Flag
NL	Nested Looping
NOP	No Operation

NOR	Inversion of the logical OR function
NVSTR	Non-volatile Store Cycle Definition
OBAR	OnCE Breakpoint Address Register
OBCTL	OnCE Breakpoint Control Register
OBMSK	OnCE Breakpont Mask Register
OCMDR	OnCE Command Register
OCCS	On-Chip Clock Synthesis
OCNTR	OnCE Count Register
OCR	OnCE Control Register
ODEC	OnCE Decoder
OEN	Output Enable
OMAC	OnCE Memory Address Comparator
OMAL	OnCE Memory Address Latch
OMR	Operating Mode Register
OnCE	On-Chip Emulation (unit)
OPABDR	OnCE Program Address Bus Decode Register
OPABER	OnCE Program Address Bus Execute Register
OPABFR	OnCE Program Address Bus Fetch Register
OPDBR	OnCE Program Data Bus Register
OPFIFO	OnCE PAB Change of Flow
OPGDBR	OnCE Program Global Data Bus Register
OPS	Output Polarity Select
OR	Overrun
OSHR	OnCE Shift Register
OS	OnCE Status bits OS1 and OS0
OSC	Oscillator
OSR	OnCE Status Register
OVRF	Overflow
PAB	Program Address Bus
PD	Permanent STOP/WAIT Disable
PDB	Program Data Bus
PE	Program Enable

PE	Parity Enable Bit
PER	Peripheral Enable Register
PF	Parity Error Flag
PFIU	Program Flash Interface Unit
PFLASH	Program Flash
PGDB	Peripheral Global Data Bus
PHASEA/B	Inputs
PLL	Phase Locked Loop
PLLCID	PLL Clock In Divide
PLLCOD	PLL Clock Out Divide
PLLDB	PLL Divide-by
PLLCR	PLL Control Register
PLLPDN	PLL Power Down
PLLSR	PLL Status Register
PLR	Priority Level Register
PMCCR	PWM Channel Control Register
PMCFG	PWM Configuration Register
PMCNT	PWM Counter Register
PMCTL	PWM Control Register
PMDEADTM	PWM Deadtime Register
PMDISMAP	PWM Disable Mapping Registers
PMFCTL	PWM Fault Control Register
PMFSA	PWM Fault Status Acknowledge
PMOUT	PWM Output Control Register
PMPORT	PWM Port Register
POL	Polarity
POR	Power on Reset
PRAM	Program RAM
PROG	Program Cycle
PSR	Processor Status Register
PT	Parity Type
PTM	Peripheral Test Mode

PUR	Pull-up Enable Register
PWD	Power Down Mode
PWM	Pulse Width Modulator
PWMEN	PWM Enable
PWMF	PWM Reload Flag
PWMRIE	PWM Reload Interrupt Enable
PWMVAL	PWM Value Registers
QE	Quadrature Encoder
QDN	Quadrature Decoder Negative Signal
RAF	Receiver Active Flag
RAM	Random Access Memory
RDRF	Receive Data Register Full
RE	Receiver Enable
REIE	Receive Error Interrupt Enable
REV	Revolution Counter Register
REVH	Revolution Hold Register
RFEN	Receive FIFO Enable
RIDLE	Receiver Idle Line
RIE	Receiver Full Interrupt Enable
ROM	Read Only Memory
RPD	Re-programmable STOP/WAIT Disable
RSRC	Receiver Source Bit
RT	Rate Tolerance
RWU	Receiver Wakeup
RXSR	Receive (data) Shift Register
SA	Saturation
SBK	Send Break
SBO	Software Breakpoint Occurrence
SBR	SCI Baud Rate
SCI	Serial Communications Interface3
SCIBR	SCI Baud Rate Register
SCICR	SCI Control Register

SCIDR	SCI Data Register
SCISR	SCI Status Register
SCLK	Serial Clock
SCR	Status and Control
SD	Stop Delay
SDK	Software Development Kit
SEXT	Sign Extend
SHFD	Shift Direction
SIM	System Integration Module
SMODE	Scan Mode
SPDRR	SPI Data Receive Register
SPDSR	SPI Data Size Register
SPDTR	SPI Data Transmit Register
SP	SPI Enable
SPI	Serial Peripheral Interface
SPMSTR	SPI Master
SPRF	SPI Receiver Full
SPRIE	SPI Receiver Interrupt Enable
SPSCR	SPI Status Control Register
SPTE	SPI Transmitter Empty
SPTIE	SPI Transmit Interrupt Enable
SR	Status Register
SRM	Switched Reluctance Motor
$\overline{\text{SS}}$	Slave Select
SSI	Synchronous Serial Interface
SWAI	Stop in Wait Mode
SWI	Software Interrupt
SYN	Sync
SYS_CNTL	System Control Register
SYS_STS	System Status Register
TAP	Test Access Port
TCSR	Text Control and Status Register

TCE	Test Counter Enable
TCF	Timer Compare Flag
TCFIE	Timer Compare Flag Interrupt Enable
TDRE	Transmit Date Register Empty
TE	Transmitter Enable
TEIE	Transmitter Empty Interrupt Enable
TEN	Test Mode Enable
TERASEL	Terase Limit
TESTR	Test Register
TFDBK	Test Feedback Clock
TFREF	Test Reference Frequency Clock
TIDLE	Transmitter Idle
TIIE	Transmitter Idle Interrupt Enable
TIRQ	Test Interrupt Request Register
TISR	Test Interrupt Source Register
TM	Test Mode bit
TMEL	Time Limit
TMODE	Test Mode bit
TMR	Quadrature Timer
TMR PD	Timer I/O Pull-up Disable
TNVHL	TNVH Limit
TNVSL	TNVS Limit
TO	Trace Occurrence
TOF	Timer Overflow Flag
TOFIE	Timer Oerflow Flag Interrupt Enable
TOPNEG	Top-side PWM Polarity Bit
TPROGL	Tprog Limit
TPGSL	TPGS Limit
TRCVL	TRCV Limit
TSTREG	Test Register
UIR	Upper Initialization Register
UPOS	Upper Position Hold Register

UPOSH	Upper Position Hold Register
VCO	Voltage Controlled Oscillator
V_{DD}	Power (Voltage Digital Drain)
V_{DDA}	Analog Power
VEL	Velocity Counter Register
VELH	Velocity Hold Register
VLMODE	Value Register Load Mode
VREF	Voltage Reference
VRM	Variable Reluctance Motor
V_{SS}	Ground
V_{SSA}	Analog Ground
WAKE	Wakeup Condition
WDE	Watchdog Enable
WP	Write Protect
WSPM	Wait State P Memory
WSX	Wait State Data Memory
WTR	Watchdog Timeout Register
WWW	World Wide Web
XDB2	X Data Bus
XE	X Address Enable
XIE	Index Pulse Interrupt Enable
XIRQ	Index Pulse Interrupt Request
XNE	Use Negative Edge of Index Pulse
XRAM	Data RAM
YE	Y Address Enable
ZCI	Zero Crossing Interrupt
ZCIE	Zero Crossing Interrupt Enable
ZCS	Zero Crossing Status
ZSRC	Zclock Source

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

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

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

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

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

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150

LDCForFreescaleSemiconductor@hibbertgroup.com

DSP56F801-7UM
Rev. 8, 03/2007