DSP56824
16-Bit Digital Signal Processor

User's Manual

DSP56824UM/D
Rev. 1.00, 01/2000

MOTOROLA

Freescale Semiconductor, Inc.
ARCHIVED BY FREESCALE SEMICONDUCTOR, INC. 2005

MFAX and OnCE™ are trademarks of Motorola, Inc.

This document contains information on a new product. Specifications and information herein are subject to change without notice.

How to reach us:

USA/EUROPE/Locations Not Listed: Motorola Literature Distribution; P.O. Box 5405, Denver, Colorado, 80217
1-303-675-2140 or 1-800-441-2447

JAPAN: Motorola Japan, Ltd.; SPS, Technical Information Center, 3-20-1, Minami-Azabu, Minato-ku,
Tokyo 106-8573 Japan. 81-3-3440-3569

ASIA/PACIFIC: Motorola Semiconductors H.K. Ltd., Silicon Harbour Centre, 2 Dai King Street,
Tai Po Industrial Estate, 2 Tai Po, N.T., Hong Kong. 852-26668334

Customer Focus Center: 1-800-521-6274

Mfax™: RMFAX0@email.sps.mot.com
–TOUCHTONE 1-602-244-6609
–US & Canada ONLY 1-800-774-184
–http://sps.motorola.com/mfax/

HOME PAGE: http://motorola.com/sps

Motorola DSP Products Home Page: http://www.motorola-dsp.com

Contents

Chapter 3
Memory Configuration and Operating Modes

Chapter 4
External Memory Interface

Chapter 5
Port B GPIO Functionality

Chapter 6
Port C GPIO Functionality

Chapter 7
Serial Peripheral Interface

Chapter 8
Synchronous Serial Interface

Chapter 9
Timers

Chapter 10
On-Chip Clock Synthesis

Chapter 11
COP and RTI Module

Chapter 12
OnCE™ Module

Appendix C
Programmer's Sheets

List of Figures

Freescale Semiconductor, Inc.
ARCHIVED BY FREESCALE SEMICONDUCTOR, INC. 2005

List of Tables

Freescale Semiconductor, Inc.
ARCHIVED BY FREESCALE SEMICONDUCTOR, INC. 2005

List of Examples

Freescale Semiconductor, Inc.
ARCHIVED BY FREESCALE SEMICONDUCTOR, INC. 2005

About This Book

This manual describes the DSP56824 16-bit digital signal processor (DSP), its memory and operating modes, and its peripheral modules. This manual is intended to be used with the *DSP56800 Family Manual* (DSP56800FM/AD), which describes the CPU, programming models, and instruction set details. The *DSP56824 Technical Data Sheet* (DSP56824/D) provides electrical specifications as well as timing, pinout, and packaging descriptions.

Audience

The information in this manual is intended to assist design and software engineers with integrating the DSP56824 digital signal processor into a design and with developing application software.

Organization

Information in this manual is organized into chapters by topic. The contents of the chapters are as follows:

Chapter 1, "DSP56824 Overview." This section provides a brief overview of the DSP56824.

Chapter 2, "Signal Descriptions." This section provides a description of the pins on the DSP56824 chip and how the pins are grouped into the various interfaces.

Chapter 3, "Memory Configuration and Operating Modes." This section describes the on-chip memory, structures, registers, and interfaces.

Chapter 4, "External Memory Interface." This section describes the DSP56824 external memory interface, which is also referred to as Port A.

Chapter 5, "Port B GPIO Functionality." This section describes the dedicated general-purpose input/output (GPIO) interface, which is also referred to as Port B.

Chapter 6, "Port C GPIO Functionality." This section describes the 16 dual-function pins that constitute Port C and defines their GPIO functions. Chapters 7–9 describe these pins' alternate functionality.

Chapter 7, "Serial Peripheral Interface." This section describes the serial peripheral interface (SPI), which is a part of Port C and which communicates with external devices, such as liquid crystal displays (LCDs) and microcontroller units (MCUs).

Chapter 8, "Synchronous Serial Interface." This section describes the synchronous serial interface (SSI), which is a part of Port C and which communicates with devices such as codecs, other DSPs, microprocessors, and peripherals to provide the primary data input path.

Chapter 9, "Timers." This section describes the three internal timer/counter devices that are a part of Port C.

Chapter 10, "On-Chip Clock Synthesis." This section describes the internal oscillator, phase lock loop (PLL), and timer distribution chain for the DSP56824.

Chapter 11, "COP and RTI Module." This section describes the on-chip watchdog timer and the real-time interrupt generator.

Chapter 12, "OnCE™ Module." This section describes the specifics of the DSP56824's On-Chip Emulation (OnCE™) module, which is accessed through the Joint Test Action Group (JTAG) port.

Chapter 13, "JTAG Port." This section describes the specifics of the DSP56824's JTAG port.

Appendix A, "Bootstrap Program." This section provides a sample listing of bootstrap code that can be used to start or reset the DSP56824.

Appendix B, "Boundary Scan Description Language Listing." This section provides the Boundary Scan Description Language (BSDL) listing for the DSP56824.

Appendix C, "Programmer's Sheets." This section provides programming references and master programming sheets used to program the DSP56824 registers.

Suggested Reading

A list of DSP-related books is included here as an aid for the engineer who is new to the field of DSP:

Advanced Topics in Signal Processing, Jae S. Lim and Alan V. Oppenheim (Prentice-Hall: 1988).

Applications of Digital Signal Processing, A. V. Oppenheim (Prentice-Hall: 1978).

Digital Processing of Signals: Theory and Practice, Maurice Bellanger (John Wiley and Sons: 1984).

Digital Signal Processing, Alan V. Oppenheim and Ronald W. Schafer (Prentice-Hall: 1975).

Digital Signal Processing: A System Design Approach, David J. DeFatta, Joseph G. Lucas, and William S. Hodgkiss (John Wiley and Sons: 1988).

Discrete-Time Signal Processing, A. V. Oppenheim and R.W. Schafer (Prentice-Hall: 1989).

Foundations of Digital Signal Processing and Data Analysis, J. A. Cadzow (Macmillan: 1987).

Handbook of Digital Signal Processing, D. F. Elliott (Academic Press: 1987).

Introduction to Digital Signal Processing, John G. Proakis and Dimitris G. Manolakis (Macmillan: 1988).

Multirate Digital Signal Processing, R. E. Crochiere and L. R. Rabiner (Prentice-Hall: 1983).

Signal Processing Algorithms, S. Stearns and R. Davis (Prentice-Hall: 1988).

Signal Processing Handbook, C. H. Chen (Marcel Dekker: 1988).

Signal Processing: The Modern Approach, James V. Candy (McGraw-Hill: 1988).

Theory and Application of Digital Signal Processing, Lawrence R. Rabiner and Bernard Gold (Prentice-Hall: 1975).

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).
- Bits within a register are formatted 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 that are asserted low (made active when pulled to ground) have an overbar over 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 monospaced font, as follows:

```
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 that are asserted low have a tilde in front of their names. In the previous example, line 3 refers to the $\overline{SS0}$ pin (shown as ~SS0).
- The word “reset” is used in three different contexts in this manual. There are a reset pin that is always written as $\overline{RESET}$, the processor state occurring when the $\overline{RESET}$ pin is asserted that is always written as Reset, and the word *reset* that refers to the reset function and is written in lowercase (without the italics that are used here for emphasis) or with a leading capital letter as style dictates, such as in headings and captions. The word “pin” is a generic term for any pin on the chip.
- The word *assert* means that a high true (active high) signal is pulled high to V_{CC} or that a low true (active low) signal is pulled low to ground. The word *deassert* means that a high true signal is pulled low to ground or that a low true signal is pulled high to V_{CC}.

Signal/Symbol	Logic State	Signal State	Voltage
$\overline{PIN}$	True	Asserted	Ground[1]
$\overline{PIN}$	False	Deasserted	V_{CC}[2]
PIN	True	Asserted	V_{CC}
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. VCC is an acceptable high voltage level. See the appropriate data sheet for the range of acceptable high voltage levels (typically a TTL logic high).

Definitions, Acronyms, and Abbreviations

The following terms appear frequently in this manual:

DSP	digital signal processor
JTAG	Joint Test Action Group
OnCE™	On-Chip Emulation
ALU	arithmetic logic unit
AGU	address generation unit

References

Please consult the following for more information on the function and characteristics of the DSP56824 digital signal processor.

- *DSP56800 Family Manual* (order number DSP56800FM/D)
- *DSP56824 Technical Data Sheet* (order number DSP56824/D)

These documents, as well as Motorola's DSP development tools, can be obtained through a local Motorola Semiconductor Sales Office or authorized distributor. To receive the latest information, access the Motorola DSP home page located at http://www.motorola-dsp.com.

Chapter 1
DSP56824 Overview

The DSP56824 is a member of the family of DSPs based on the DSP56800 core. This general-purpose DSP combines processing power with configuration flexibility, making it an excellent, cost-effective solution for signal processing and control functions. To achieve its design goals, the DSP56824 uses an efficient MPU-style general-purpose 16-bit DSP core, program and data memories, and support circuitry.

The CPU (the DSP56800 core) consists of three execution units operating in parallel, allowing as many as six operations during each instruction cycle. The MPU-style programming model and optimized instruction set allow straightforward generation of efficient, compact DSP and control code. The instruction set is also highly efficient for C Compilers.

The DSP56824 supports program execution from internal or external memories. Two data operands can be accessed per instruction cycle from the on-chip data RAM. The programmable peripherals and ports provide support for interfacing multiple external devices, such as codecs, microprocessors, or other DSPs. The DSP56824 also provides 16 to 32 general-purpose input/output (GPIO) lines, depending on how peripherals are configured, and 2 external dedicated interrupt lines. Because of its configuration flexibility, compact program code, and low cost, the DSP56800 core family is well suited for cost-sensitive applications, including digital wireless messaging, digital answering machines and feature phones, wireline and wireless modems, servo and AC motor control, and digital cameras.

1.1 DSP56824 Architecture Overview

The DSP56824 consists of the DSP56800 core, program and data memory, and peripherals useful for embedded control applications. Figure 1-1 on page 1-2 shows a block diagram of the DSP56824 chip.

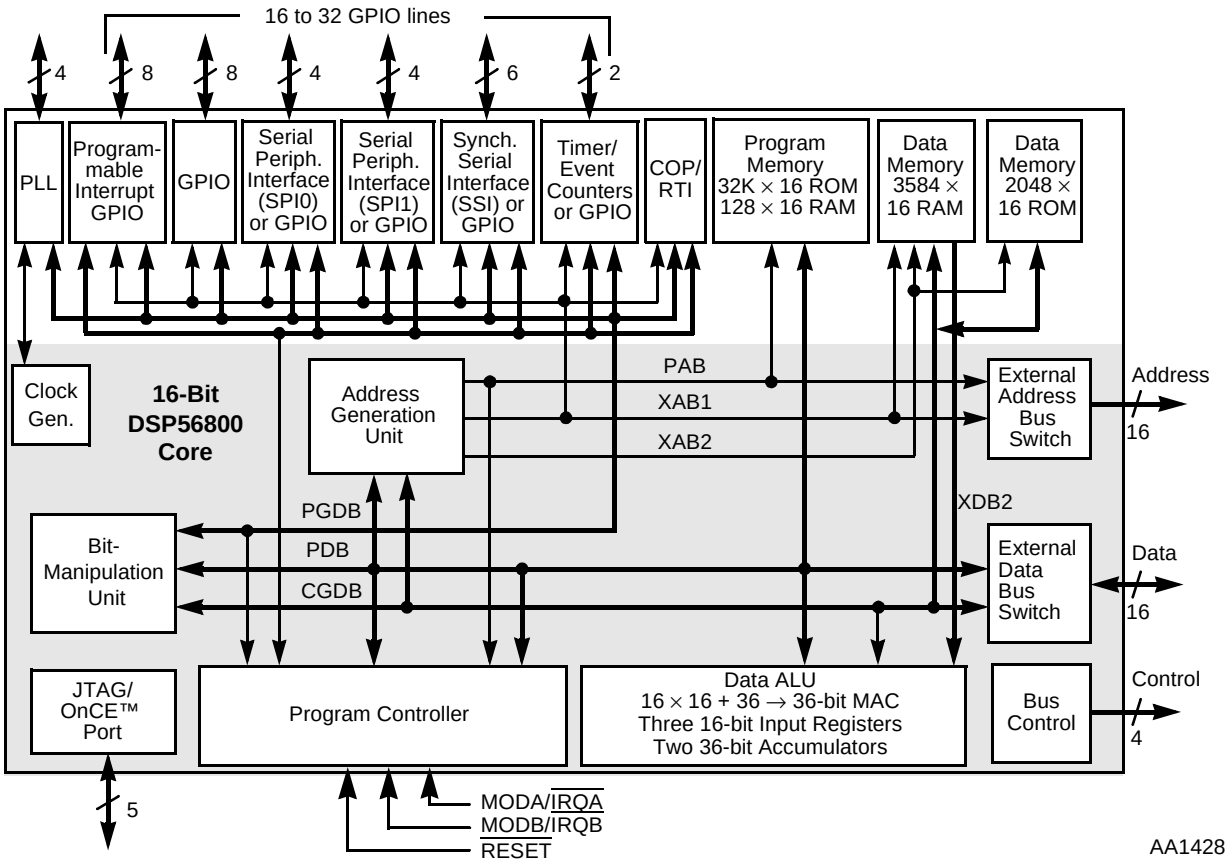


Figure 1-1. DSP56824 Functional Block Diagram

Features of the DSP56824 include the following:

- 35 million instructions per second (MIPS) with 70 MHz clock
- On-chip memory
- Off-chip memory expansion capability
  - — Off-chip expansion to 64K × 16 data memory
  - — Off-chip expansion to 64K × 16 program memory
- On-chip peripherals (listed in the following subsection)
- 2.7 V to 3.6 V power supply
- Very low-power complementary metal-oxide semiconductor (CMOS) design
- Power management circuitry with power-saving wait and multiple stop modes
- Fully static logic with operating frequencies down to DC

1.1.1 DSP56824 Peripheral Blocks

The DSP56824 provides the following peripheral blocks:

- On-chip memory
  - 32K × 16 program ROM
  - 128 × 16 program RAM
  - 3.5K × 16 RAM for data and applications
  - 2K × 16 data ROM
- External memory interface
- GPIO module
- Programmable I/O module
- Serial peripheral interface (SPI—two provided)
- Synchronous serial interface (SSI)
- General-purpose triple timer module
- On-chip clock synthesis block (phase lock loop or PLL)
- Computer Operating Properly (COP) and real-time interrupt (RTI) module
- JTAG/OnCE™ Port

1.1.1.1 On-Chip Memory

The DSP56824 uses a Harvard architecture, which provides independent data and program memory. On-chip ROM is provided for both the X data (2K × 16-bit) and program (P) memory (32K × 16-bit). In addition, on-chip RAM is provided for both the X data (3.5K × 16-bit) and P memory (128 × 16-bit). Both the program and data memories can be expanded off-chip.

1.1.1.2 External Memory Interface (Port A)

The DSP56824 provides an external memory interface, also known as Port A. This port provides a total of 36 pins—16 pins for an external address bus, 16 pins for an external data bus, and 4 pins for bus control.

1.1.1.3 General-Purpose Input/Output Port (Port B)

A dedicated general-purpose input/output (GPIO) port, also known as Port B, provides 16 programmable I/O pins. This port is configured so that it is possible to generate interrupts when a transition is detected on any of its lower eight pins.

1.1.1.4 Programmable I/O Port (Port C)

Port C provides 16 multiplexed general-purpose programmable I/O pins. Each pin can be individually selected as a GPIO pin or allocated to on-chip peripherals—the general-purpose timer module, two SPIs, and an SSI. Unlike the GPIO pins on Port B, the Port C pins cannot be configured to provide GPIO interrupts, but interrupts are available for the timer module, the two SPI ports, and the SSI port on Port C.

1.1.1.5 Serial Peripheral Interface (SPI)

The serial peripheral interface (SPI) is an independent serial communications subsystem that allows the DSP56824 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. In Slave mode, a transfer is initiated by the reception of a clock signal. Two separate SPIs are implemented on Port C.

1.1.1.6 Synchronous Serial Interface (SSI)

The synchronous serial interface (SSI) is a full-duplex serial port that allows the DSP56824 to communicate with a variety of multiple serial devices including industry-standard codecs, other DSPs, microprocessors, and peripherals that implement the Motorola SPI. It is typically used to transfer samples in a periodic manner. The SSI consists of independent transmitter and receiver sections with independent clock generation and frame synchronization. The SSI is implemented on Port C.

1.1.1.7 General-Purpose Timer Module

The timer module provides three independently programmable 16-bit timer/event counters. All three timer/counters can be clocked with clocks coming from one of three internal sources, including one of the other timers. In addition, the counters can be clocked with external clocking from the timer I/O pins (TIO01 or TIO2) on Port C to count external events when configured as inputs. The same pins can be used as a timer pulse or for timer clock generation when used as outputs. The timer/event counters can be used to either interrupt the DSP56824 or to signal an external device at periodic intervals. The timer I/O pins are implemented as part of Port C.

1.1.1.8 On-Chip Clock Synthesis Block

The clock synthesis module generates the clocking for the DSP56824. It generates three different clocks used by the DSP56800 core and DSP56824 peripherals. It contains a PLL that can multiply up the frequency or can be bypassed, and also contains a prescaler/divider used to distribute clocks to peripherals and to lower power consumption on the DSP56824. It also selects which clock, if any, is routed to the CLKO pin of the DSP56824.

1.1.1.9 COP/RTI Module

The Computer Operating Properly (COP) and real-time interrupt (RTI) module provides two separate functions: a watchdog timer and an interrupt generator. These two functions monitor processor activity and provide an automatic reset signal if a failure occurs. Both functions are contained in the same block because the input clock for both comes from a common clock divider.

1.1.1.10 JTAG/OnCE™ Port

The JTAG/OnCE port allows the user to insert the DSP56824 into a target system while retaining debug control. The JTAG port provides board-level testing capability that is compatible with the *IEEE Standard Test Access Port and Boundary-Scan Architecture* (IEEE 1149.1a-1993) specification defined by the Joint Test Action Group (JTAG). Five dedicated pins interface to a test access port (TAP) that contains a 16-state controller.

The On-Chip Emulation (OnCE) module allows the user to interact in a debug environment with the DSP56800 core and its peripherals non-intrusively. 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 DSP56800 core for sophisticated debugging and economical system development. The JTAG/OnCE port allows access to the OnCE module and through the DSP56824 to its target system, retaining debug control without sacrificing other user-accessible on-chip resources.

1.1.2 DSP56824 Peripheral Interrupts

The peripherals on the DSP56824 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) and can selectively be enabled or disabled via the interrupt priority level found on the DSP56800 core.

Chapter 3, “Memory Configuration and Operating Modes,” provides complete details on interrupt vectors.

1.2 DSP56800 Core Description

The DSP56800 core consists of functional units that operate in parallel to increase the throughput of the machine. Major features of the DSP56800 core include the following:

- Single-cycle 16-bit x 16-bit parallel multiply-accumulator (MAC)
- Two 36-bit accumulators including extension bits
- 16-bit bidirectional barrel shifter
- Highly parallel instruction set with unique DSP and controller addressing modes
- Nested hardware DO loops
- Software subroutine and interrupt stack with unlimited depth
- Instruction set that supports both DSP and controller functions for compact code
- Efficient C Compiler and local variable support

An overall block diagram of the DSP56800 core architecture is shown in Figure 1-2 on page 1-6. The DSP56800 core is fed by internal program and data memory, an external memory interface, 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
- Address buses
- Data buses

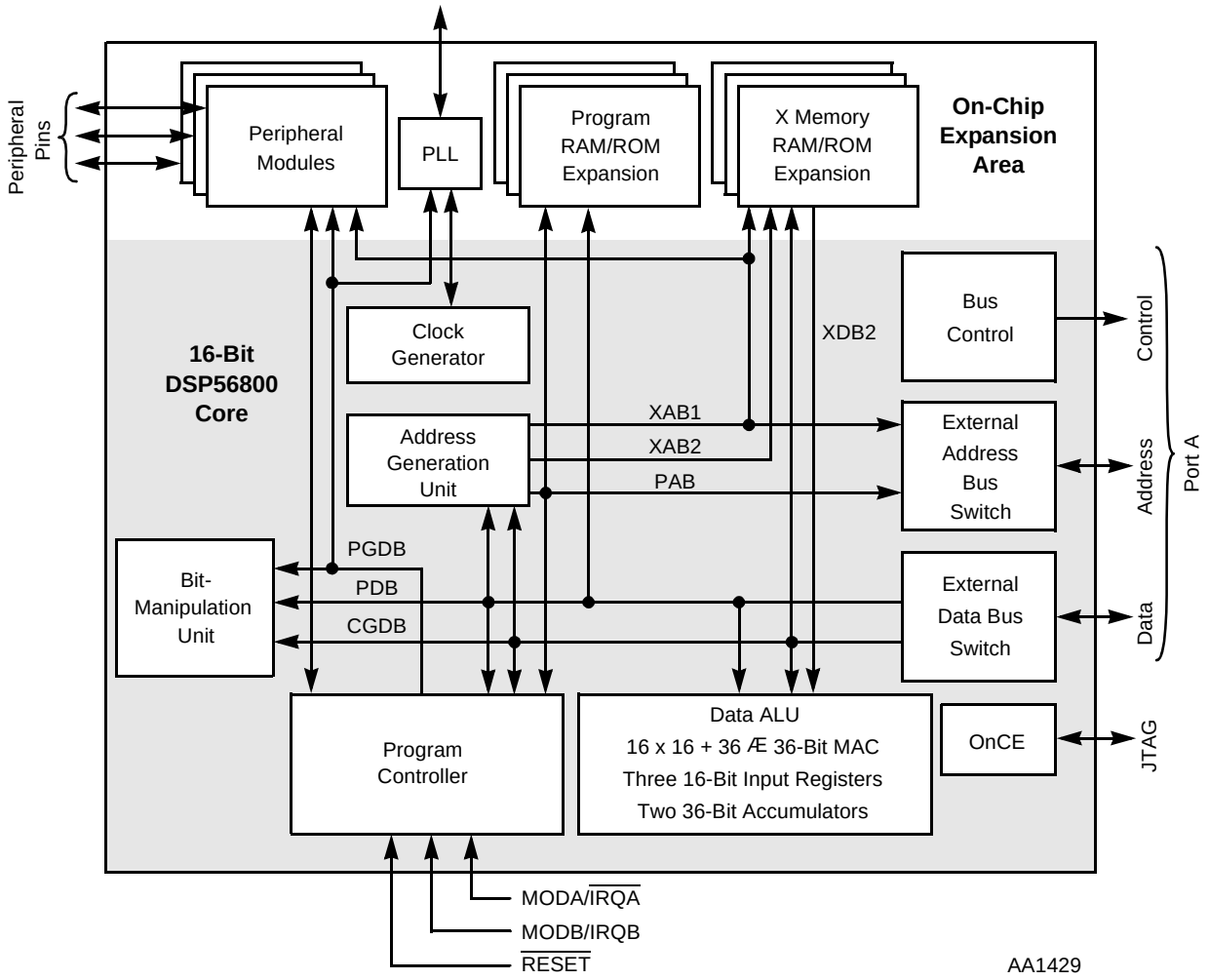


Figure 1-2. DSP56800 Core Block Diagram

The program controller, AGU, and data ALU each contain a discrete register set and control logic, so that 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, as shown in Figure 1-3 on page 1-7.

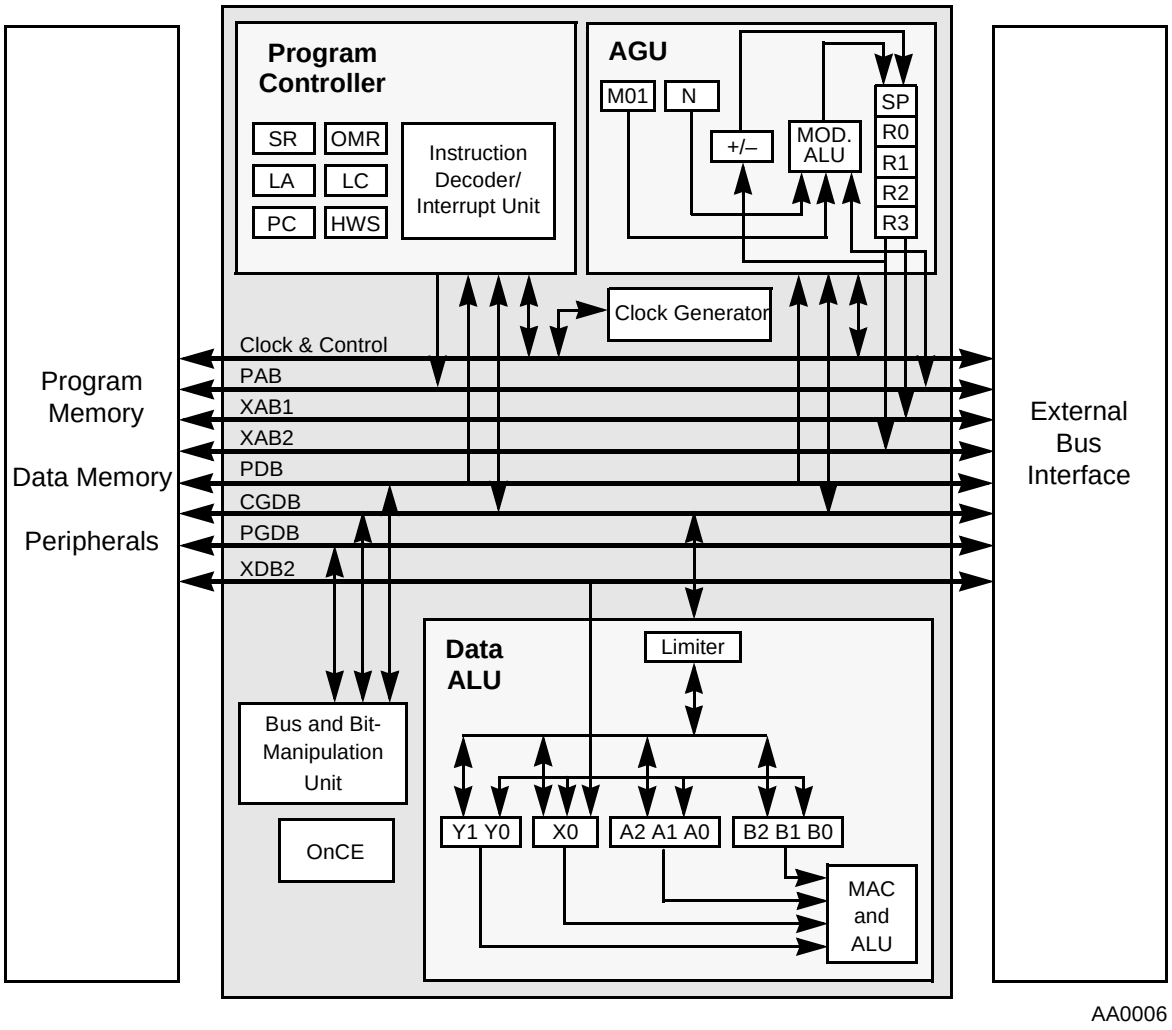


Figure 1-3. DSP56800 Bus Block Diagram

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.

1.2.1 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 the following:

- 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-accumulation with positive or negative accumulation
- Addition
- Subtraction
- Shifting
- Logical operations

Arithmetic operations are done 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 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 X0 register can also be written by the X data bus 2 (XDB2) with a 16-bit operand.

1.2.2 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 words of 16-bit data. Hooks are provided on the DSP56800 core to expand this address space. Its arithmetic unit can perform linear and modulo arithmetic.

1.2.3 Program Controller and Hardware Looping Unit

The program controller performs instruction prefetching, instruction decoding, hardware loop control, and 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 mode and interrupt control pins provide input to the program interrupt controller. The Mode Select A/External Interrupt Request A (MODA/$\overline{\text{IRQA}}$) pin and the Mode Select B/External Interrupt Request B (MODB/$\overline{\text{IRQB}}$) pin select the DSP56824 operating mode and receive interrupt requests from external sources.

The $\overline{\text{RESET}}$ pin resets the DSP56824. When it is asserted, it initializes the chip and places it in the Reset state. When the $\overline{\text{RESET}}$ pin is deasserted, the DSP56824 assumes the operating mode indicated by the MODA and MODB pins.

1.2.4 Bit-Manipulation Unit

The Bit Manipulation Unit performs bit-field 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-bit-field instructions, this unit tests bits on the upper or lower byte of a 16-bit word. In other words, the mask tests a maximum of 8 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 core global data bus (CGDB) bus and the peripheral register is connected to the peripheral global data bus (PGDB).

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

1.2.5 Address and Data Buses

Addresses are provided to the internal X data memory on two unidirectional 16-bit buses— X address bus 1 (XAB1) and X address bus 2 (XAB2). Program memory addresses are provided on the unidirectional 16-bit program address bus (PAB). Note that 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 DSP56824 occurs over three bidirectional 16-bit buses—the core global data bus (CGDB), the program data bus (PDB), and the peripheral global data bus (PGDB)—and also over one unidirectional 16-bit bus, the X data bus 2 (XDB2). Data transfer between the data ALU and the X data memory occurs over the CGDB when one memory access is performed, and over the CGDB and the XDB2 when two simultaneous memory reads are performed. All other data transfers to core blocks occur over the CGDB, and all transfers to and from peripherals occur 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 general register-to-register, register-to-memory, and memory-to-register transfers, and can transfer up to three 16-bit words in the same instruction cycle. Transfers between buses are accomplished in the bit-manipulation unit. Table 1-1 lists the address and data buses for the DSP56800 core.

Table 1-1. DSP56800 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

Table 1-1. DSP56800 Address and Data Buses (Continued)

Bus	Bus Name	Bus Width, Direction, and Use
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

1.2.6 On-Chip Emulation (OnCE) Module

The On-Chip Emulation (OnCE) module allows the user to interact in a debug environment with the DSP56800 core and its peripherals non-intrusively. 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 DSP56800 core for sophisticated debugging and economical system development. The JTAG port allows access to the OnCE module and through the DSP56824 to its target system, retaining debug control without sacrificing other user-accessible on-chip resources. This capability eliminates the costly cabling and the access to processor pins required by traditional emulator systems. The OnCE interface is fully described in **Chapter 12, "OnCE™ Module."**

1.3 DSP56800 Programming Model

The programming model for the registers in the DSP56800 core is shown in Figure 1-4 on page 1-11.

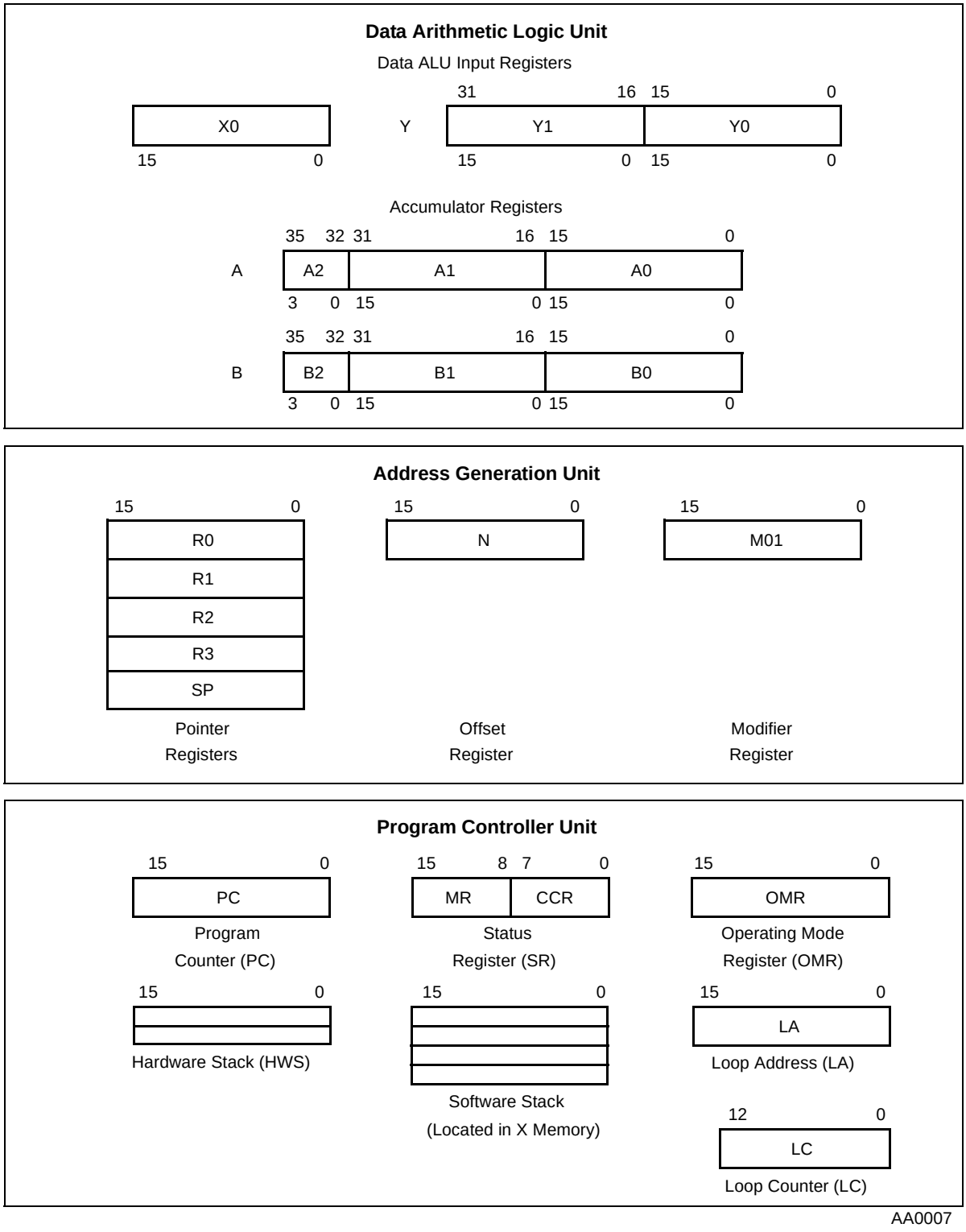


Figure 1-4. DSP56800 Core Programming Model

1.4 Code Development on the DSP56824

The DSP56824 instruction set, described in detail in the *DSP56800 Family Manual*, provides assembly-level programming for this product. This manual provides a number of samples of source code to demonstrate the programming of certain features. These examples are not comprehensive; they are only a sampling of what is possible. See the *DSP56800 Family Manual* for more information on development hardware and software products for the DSP56824, including an Application Development System (ADS) that allows access to most DSP56824 functions and peripherals.

Two mechanisms on the DSP56824 aid code development—full access by all instructions to the external data bus and OnCE module hooks. The first is useful when code is first being developed on a hardware platform using external program memory. This section describes this first option. The OnCE module is fully described in **Chapter 12, "OnCE™ Module."**

Instructions on the DSP56824 can be executed without regard for whether the instruction fetch is on-chip or off-chip and whether any data access is on-chip or off-chip. However, executing an instruction (including parallel moves) may require as many as three memory accesses. If more than one of these memory accesses occurs off-chip, an additional instruction cycle is required for every external access because only one access to external memory can occur at a time. This is shown in Example 1-1 and in the following discussion.

Example 1-1. On-Chip and Off-Chip Instruction Fetches

```
mac x0,y0,a x:(r0)+,y0 x:(r3)+,x0
```

The various permutations of memory accesses that may be represented in Example 1-1 are:

- Case 1—Instruction located on-chip, both x:() data accesses performed to on-chip memory

  In this case, because all memories are located on-chip, no external accesses are performed and the instruction runs in one instruction cycle, correctly performing all three accesses to on-chip memory.
- Case 2—Instruction located off-chip, both x:() data accesses performed to on-chip memory

  In this case, only one external memory access occurs off-chip. The instruction fetch occurs over the external bus, and the data accesses are made to on-chip memory. The instruction still runs in one instruction cycle, correctly performing all three accesses.
- Case 3—Instruction located on-chip, one x:() data access performed to off-chip memory

  In this case, only one external memory access occurs off-chip. The instruction fetch is done to on-chip memory, one data access occurs over the external bus, and one data access is done to on-chip memory. The instruction still runs in one instruction cycle, correctly performing all three accesses.
- Case 4—Instruction located off-chip, one x:() data access performed to off-chip memory

  In this case, two external memory accesses occur off-chip. The instruction fetch occurs over the external bus, followed by the external data access. A data access to internal memory also takes place. The instruction now runs in two instruction cycles, correctly performing all three accesses.

Case 4 is often used during code development to provide the best visibility. This feature allows a logic analyzer to be placed on the external bus during code development on target hardware so that all memory accesses are visible. Separate $\overline{PS}$ and $\overline{DS}$ pins are provided to indicate whether the access is to external program or data memory.

NOTE:

In this mode, accesses to on-chip peripherals are not visible because these memory-mapped registers are on-chip.

An example of a system where all program and memory accesses are visible is shown in Figure 1-5.

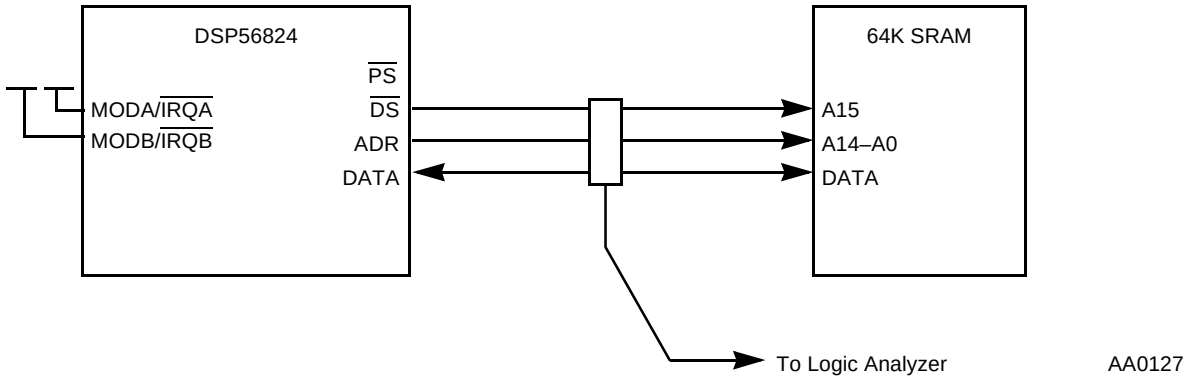


Figure 1-5. Code Development with Visibility on All Memory Accesses

In this example, the DSP56824 is programmed for operating mode 3 (development mode) in the operating mode register (OMR) to specify that all program accesses are performed externally. See Section 3.1, “DSP56824 Memory Map,” on page 3-1 for a detailed description of the OMR. Likewise, the EX bit (in the OMR) is set to specify that data accesses are performed externally. An exception to this is the second access on any instruction that performs two reads in a single instruction. In this case, the second read using the R3 pointer always occurs to on-chip memory. If this is an issue, the instruction performing two data memory reads can be replaced by two instructions, each performing one of the two data memory accesses.

Freescale Semiconductor, Inc.
ARCHIVED BY FREESCALE SEMICONDUCTOR, INC. 2005

Chapter 2
Signal Descriptions

The input and output signals of the DSP56824 are organized into functional groups, as shown in Table 2-1 on page 2-2 and as illustrated in Figure 2-1. In Table 2-2 on page 2-3 through Table 2-14 on page 2-12, each row describes the signal or signals present on an individual pin. Note that some pins can carry more than one signal, depending on chip configuration.

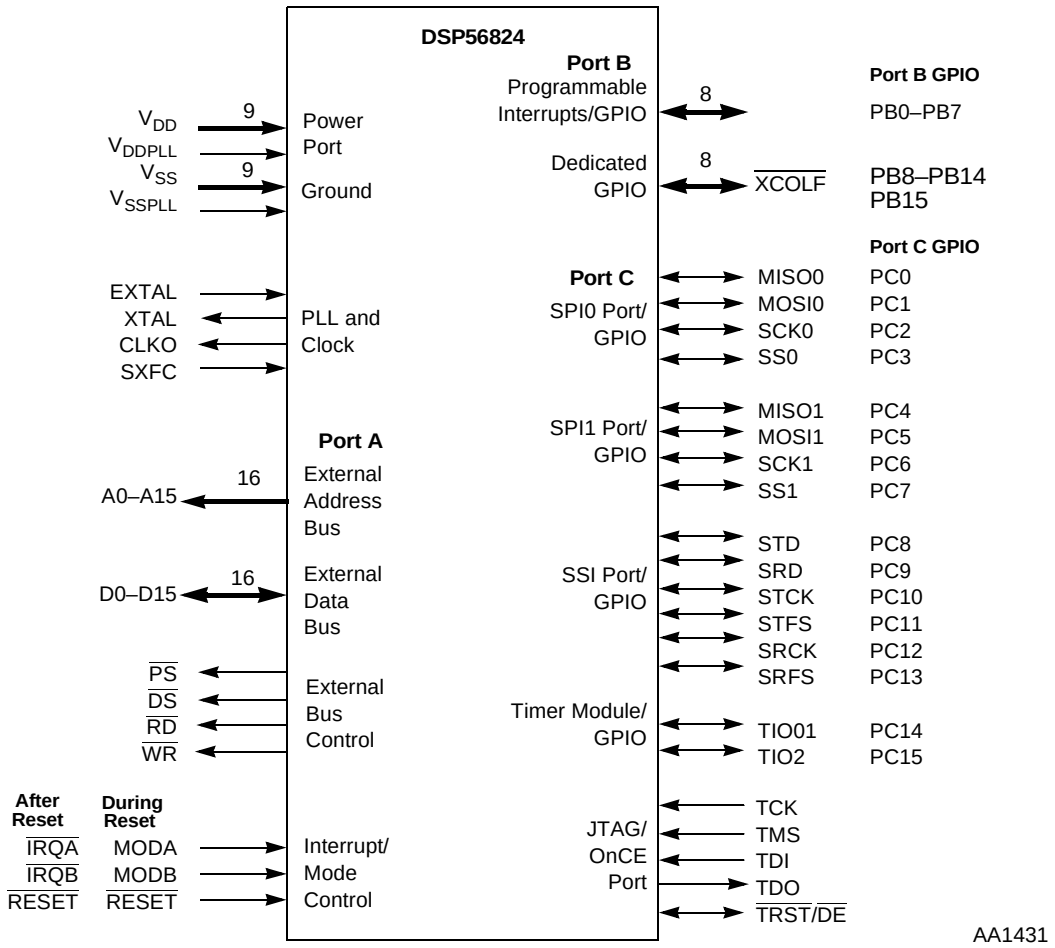


Figure 2-1. DSP56824 Functional Group Pin Allocations

The interface signals have the following general characteristics:

- The following pins are pulled high with a weak on-chip resistor: TDI, TMS, and $\overline{\text{TRST}}$/$\overline{\text{DE}}$.
- The following pins are pulled high by the DSP during hardware reset (assertion of $\overline{\text{RESET}}$): $\overline{\text{PS}}$, $\overline{\text{DS}}$, $\overline{\text{RD}}$, $\overline{\text{WR}}$, and $\overline{\text{XCOLF}}$/PB15.
- The following pins can be pulled high by the DSP during the JTAG `EXTEST_PULLUP` instruction: EXTAL, D0–D15, PB0–PB15, PC0–PC15, MODA/$\overline{\text{IRQA}}$, MODB/$\overline{\text{IRQB}}$, and $\overline{\text{RESET}}$.
- All unused port pins configured as inputs should be properly terminated through a pull-up resistor except for pins that are tied to an internal pull-up or pull-down resistor. All power and ground pins should be connected to the appropriate low-impedance power and ground paths.

The I/O signals are organized into the functional groups, as summarized in Table 2-1.

Table 2-1. Functional Group Pin Allocations

Functional Group	Number of Pins	Detailed Description
Power (V_{DD} or V_{DDPLL})	10	Table 2-2 on page 2-3
Ground (V_{SS} or V_{SSPLL})	10	Table 2-3 on page 2-3
Clock and phase lock loop (PLL)	4	Table 2-4 on page 2-3
Address bus	16	Table 2-5 on page 2-4
Data bus	16	Table 2-6 on page 2-4
Bus control	4	Table 2-7 on page 2-4
Interrupt and mode control	3	Table 2-8 on page 2-5
Programmable interrupt general-purpose input/output	8	Table 2-9 on page 2-6
Dedicated general-purpose input/output	8	Table 2-10 on page 2-7
Serial peripheral interface (SPI) ports *	8	Table 2-11 on page 2-7
Synchronous serial interface (SSI) port *	6	Table 2-12 on page 2-10
Timer module*	2	Table 2-13 on page 2-11
JTAG/OnCE	5	Table 2-14 on page 2-12
Note: *Alternately, general-purpose I/O pins		

2.1 Power and Ground Signals

Table 2-2. Power Inputs

Signal Name (Number of Pins)	Signal Description
V_{DD} (9)	**Power**—These pins provide power to the internal structures of the chip, and should all be attached to V_{DD}.
V_{DDPLL} (1)	**PLL Power**—This pin supplies a quiet power source to the VCO to provide greater frequency stability.

Table 2-3. Grounds

Signal Name (Number of Pins)	Signal Description
V_{SS} (9)	**GND**—These pins provide grounding for the internal structures of the chip and should all be attached to V_{SS}.
V_{SSPLL} (1)	**PLL Ground**—This pin supplies a quiet ground to the VCO to provide greater frequency stability.

2.2 Clock and Phase Lock Loop (PLL) Signals

Table 2-4. Clock and Phase Lock Loop (PLL) Signals

Signal Name	Signal Type	State During Reset	Signal Description
EXTAL	Input	Input	**External Clock/Crystal Input**—This input should be connected to an external clock or to an external oscillator. After being squared, the input clock can be selected to provide the clock directly to the DSP core. The minimum instruction time is two input clock periods broken up into four phases named T0, T1, T2, and T3. This input clock can also be selected as the input clock for the on-chip PLL. When the low frequency mode of $\overline{\text{XCOLF}}$ is selected, EXTAL is in phase with Phi1, T1, and T3. When the default mode is selected, EXTAL is in phase with CLKO, Phi0, T0, and T2.
XTAL	Output	Chip-driven	**Crystal Output**—This output connects the internal crystal oscillator output to an external crystal. If an external clock is used, XTAL should not be connected.
CLKO	Output	Chip-driven	**Clock Output**—This pin outputs a buffered clock signal. By programming the CS[1:0] bits in the PLL control register 1 (PCR1), the user can select between outputting a squared version of the signal applied to EXTAL and a version of the DSP master clock at the output of the PLL. The clock frequency on this pin can also be disabled by programming the CS[1:0] bits in PCR1.

Table 2-4. Clock and Phase Lock Loop (PLL) Signals (Continued)

Signal Name	Signal Type	State During Reset	Signal Description
SXFC	Input	Input	**External Filter Capacitor**—This pin is used to add an external filter circuit to the PLL.

2.3 External Memory Interface (Port A)

Table 2-5. Address Bus Signals

Signal Name	Signal Type	State During Reset	Signal Description
A0–A15	Output	Tri-stated	**Address Bus**—Signals A0–A15 change in T0 and specify the address for an external program or data memory access. The value of the DRV bit in the bus control register (BCR) causes the address bus to retain the last external address (DRV = 1) or to be tri-stated (DRV = 0) during an internal access or in stop or wait mode. See Section 4.2.1, “Bus Control Register (BCR),” on page 4-3.

Table 2-6. Data Bus Signals

Signal Name	Signal Type	State During Reset	Signal Description
D0–D15	Input/ output	Tri-stated	**Data Bus**—Read data is sampled in on the trailing edge of T2, while write data output is enabled on the leading edge of T2 and tri-stated on the leading edge of T0. D0–D15 are tri-stated when the external bus is inactive.

Table 2-7. Bus Control Signals

Signal Name	Signal Type	State During Reset	Signal Description
$\overline{PS}$	Output	Pulled high internally	**Program Memory Select**—$\overline{PS}$ is asserted low for external program memory access. If the external bus is not used during an instruction cycle (T0, T1, T2, or T3), $\overline{PS}$ is deasserted high in T0. During an internal access in stop or wait mode, the value of the DRV bit in the BCR determines whether the chip continues to drive $\overline{PS}$ (DRV = 1) or tri-states $\overline{PS}$ (DRV = 0).
$\overline{DS}$	Output	Pulled high internally	**Data Memory Select**—$\overline{DS}$ is asserted low during T0 for external data memory access. If the external bus is not accessed during an instruction cycle (T0, T1, T2, or T3), $\overline{DS}$ is deasserted high in T0. During an internal access in stop or wait mode, the value of the DRV bit in the BCR determines whether the chip continues to drive $\overline{DS}$ (DRV = 1) or tri-states $\overline{DS}$ (DRV = 0).

Table 2-7. Bus Control Signals (Continued)

Signal Name	Signal Type	State During Reset	Signal Description
$\overline{WR}$	Output	Pulled high internally	**Write Enable**—$\overline{WR}$ is asserted low during external memory write cycles. When $\overline{WR}$ is asserted low in T1, the data bus pins D0–D15 become outputs and the DSP puts data on the bus during the leading edge of T2. When $\overline{WR}$ is deasserted high in T3, the external data is latched inside the external device. When $\overline{WR}$ is asserted, it qualifies the A0–A15, $\overline{PS}$, and $\overline{DS}$ pins. $\overline{WR}$ can be connected directly to the $\overline{WE}$ pin of a Static RAM. During an internal access in stop or wait mode, the value of the DRV bit in the BCR determines whether the chip continues to drive $\overline{WR}$ (DRV = 1) or tri-states $\overline{WR}$ (DRV = 0).
$\overline{RD}$	Output	Pulled high internally	**Read Enable**—$\overline{RD}$ is asserted low during external memory read cycles. When $\overline{RD}$ is asserted low during late T0 or early T1, the data bus pins D0–D15 become inputs and an external device is enabled onto the DSP data bus. When $\overline{RD}$ is deasserted high in T3, the external data is latched in the DSP. When $\overline{RD}$ is asserted, it qualifies the A0–A15, $\overline{PS}$, and $\overline{DS}$ pins. $\overline{RD}$ can be connected directly to the $\overline{OE}$ pin of a Static RAM or ROM. During an internal access in stop or wait mode, the value of the DRV bit in the BCR determines whether the chip continues to drive $\overline{RD}$ (DRV = 1) or tri-states $\overline{RD}$ (DRV = 0).

2.4 Interrupt and Mode Control Signals

Table 2-8. Interrupt and Mode Control Signals

Signal Name	Signal Type	State During Reset	Signal Description
MODA	Input	Input	**Mode Select A**—During hardware reset, MODA and MODB select one of the four initial chip operating modes latched into the operating mode register (OMR). Several clock cycles (depending on PLL setup time) after leaving the Reset state, the MODA pin changes to external interrupt request $\overline{IRQA}$. The chip operating mode can be changed by software after reset.
$\overline{IRQA}$	Input		**External Interrupt Request A**—The $\overline{IRQA}$ input is an asynchronous external interrupt request that indicates that an external device is requesting service. It can be programmed to be level-sensitive or negative-edge triggered. If level-sensitive triggering is selected, an external pull-up resistor is required for Wired-OR operation. If the processor is in the stop state and $\overline{IRQA}$ is asserted, the processor will exit the stop state.

Table 2-8. Interrupt and Mode Control Signals (Continued)

Signal Name	Signal Type	State During Reset	Signal Description
MODB	Input	Input	**Mode Select B**—During hardware reset, MODA and MODB select one of the four initial chip operating modes latched into the OMR. Several clock cycles (depending on PLL setup time) after leaving the Reset state, the MODB pin changes to external interrupt request $\overline{\text{IRQB}}$. After reset, the chip operating mode can be changed by software.
$\overline{\text{IRQB}}$	Input		**External Interrupt Request B**—The $\overline{\text{IRQB}}$ input is an asynchronous external interrupt request that indicates that an external device is requesting service. It can be programmed to be level-sensitive or negative-edge triggered. If level-sensitive triggering is selected, an external pull-up resistor is required for Wired-OR operation.
$\overline{\text{RESET}}$	Input	Input	**Reset**—This input is a direct hardware reset on the processor. When $\overline{\text{RESET}}$ is asserted low, the DSP 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 MODA and MODB pins. The internal reset signal should be deasserted synchronously with the internal clocks. To ensure complete hardware reset, $\overline{\text{RESET}}$ and $\overline{\text{TRST}}$/$\overline{\text{DE}}$ should be asserted together. The only exception occurs in a debugging environment when a hardware DSP reset is required and it is necessary not to reset the JTAG/OnCE port. In this case, assert $\overline{\text{RESET}}$, but do not assert $\overline{\text{TRST}}$/$\overline{\text{DE}}$.

2.5 GPIO Signals

Table 2-9. Programmable Interrupt GPIO Signals

Signal Name	Signal Type	State During Reset	Signal Description
PB0–PB7	Input or output	Input	**Port B GPIO**—These eight pins can be programmed to generate an interrupt for any pin programmed as an input when there is a transition on that pin. Each pin can be configured individually to recognize a low-to-high or a high-to-low transition. In addition, these pins are dedicated GPIO pins that can individually be programmed as input or output pins. After reset, the default state is GPIO input.

Table 2-10. Dedicated GPIO Signals

Signal Name	Signal Type	State During Reset	Signal Description
PB8–PB14	Input or output	Input	**Port B GPIO**—These eight pins are dedicated GPIO pins that can individually be programmed as input or output pins. After reset, the default state is GPIO input.
$\overline{\text{XCOLF}}$	Input	Input, pulled high internally	**$\overline{\text{XCOLF}}$**—During reset, the External Crystal Oscillator Low Frequency ($\overline{\text{XCOLF}}$) function of this pin is active. PB15/$\overline{\text{XCOLF}}$ is tied to an on-chip pull-up transistor that is active during reset. When $\overline{\text{XCOLF}}$ is driven low during reset (or tied to a 10 kΩ pull-down resistor), the crystal oscillator amplifier is set to the low frequency mode. In this low frequency mode, only oscillator frequencies of 32 kHz and 38.4 kHz are supported. If $\overline{\text{XCOLF}}$ is not driven low during reset (or if a pull-down resistor is not used), the crystal oscillator amplifier operates in the default mode, and oscillator frequencies from 2 MHz to 10 MHz are supported. If an external clock is provided to the EXTAL pin, 70 MHz is the maximum frequency allowed. (In this case, do not connect a pull-down resistor or drive this pin low during reset.) When the low frequency mode is selected, EXTAL is in phase with Phi1, T1, and T3. When the default mode is selected, EXTAL is in phase with CLKO, Phi0, T0, and T2.
PB15	Input or output		**Port B GPIO**—This pin is a dedicated GPIO pin that can individually be programmed as an input or output pin. After reset, the default state is GPIO input. $\overline{\text{XCOLF}}$ is not resampled for Computer Operating Properly (COP) reset.

2.6 Serial Peripheral Interface (SPI) Signals

Table 2-11. Serial Peripheral Interface (SPI0 and SPI1) Signals

Signal Name	Signal Type	State During Reset	Signal Description
MISO0	Input/ output	Input	**SPI0 Master In/Slave Out (MISO0)**—This serial data pin is an input to a master device and an output from a slave device. The MISO0 line of a slave device is placed in the high-impedance state if the slave device is not selected. The driver on this pin can be configured as an open-drain driver by the SPI's Wired-OR mode (WOM) bit when this pin is configured for SPI operation.
PC0	Input or output		**Port C GPIO 0 (PC0)**—This pin is a GPIO pin called PC0 when the SPI MISO0 function is not being used. After reset, the default state is GPIO input.

Table 2-11. Serial Peripheral Interface (SPI0 and SPI1) Signals (Continued)

Signal Name	Signal Type	State During Reset	Signal Description
MOSI0	Input/ output	Input	**SPI0 Master Out/Slave In (MOSI0)**—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 MOSI0 line a half-cycle before the clock edge that the slave device uses to latch the data. The driver on this pin can be configured as an open-drain driver by the SPI's WOM bit when this pin is configured for SPI operation.
PC1	Input or output		**Port C GPIO 1 (PC1)**—This pin is a GPIO pin called PC1 when the SPI MOSI0 function is not being used. After reset, the default state is GPIO input.
SCK0	Input/ output	Input	**SPI0 Serial Clock (SCKO)**—This bidirectional pin provides a serial bit rate clock for the SPI. This gated clock signal is an input to a slave device and is generated as an output by a master device. Slave devices ignore the SCK signal unless the $\overline{SS}$ pin is active low. In both master and slave SPI devices, data is shifted on one edge of the SCK signal and is sampled on the opposite edge where data is stable. The driver on this pin can be configured as an open-drain driver by the SPI's WOM bit when this pin is configured for SPI operation. When using Wired-OR mode, the user must provide an external pull-up device.
PC2	Input or output		**Port C GPIO 2 (PC2)**—This pin is a GPIO pin called PC2 when the SPI SCK0 function is not being used. After reset, the default state is GPIO input.
$\overline{SS0}$	Input	Input	**SPI0 Slave Select ($\overline{SS0}$)**—This input pin selects a slave device before a master device can exchange data with the slave device. $\overline{SS}$ must be low before data transactions and must stay low for the duration of the transaction. The $\overline{SS}$ line of the master must be held high.
PC3	Input or output		**Port C GPIO 3 (PC3)**—This pin is a GPIO pin called PC3 when the SPI $\overline{SS0}$ function is not being used. After reset, the default state is GPIO input.
MISO1	Input/ output	Input	**SPI1 Master In/Slave Out (MISO1)**—This serial data pin is an input to a master device and an output from a slave device. The MISO1 line of a slave device is placed in the high-impedance state if the slave device is not selected. The driver on this pin can be configured as an open-drain driver by the SPI's WOM bit when this pin is configured for SPI operation.
PC4	Input or output		**Port C GPIO 4 (PC4)**—This pin is a GPIO pin called PC4 when the SPI MISO1 function is not being used. After reset, the default state is GPIO input.

Table 2-11. Serial Peripheral Interface (SPI0 and SPI1) Signals (Continued)

Signal Name	Signal Type	State During Reset	Signal Description
MOSI1	Input/ output	Input	**SPI1 Master Out/Slave In (MOSI1)**—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 MOSI0 line a half-cycle before the clock edge that the slave device uses to latch the data. The driver on this pin can be configured as an open-drain driver by the SPI's WOM bit when this pin is configured for SPI operation. When using Wired-OR mode, the user must provide an external pull-up device.
PC5	Input or output		**Port C GPIO5 (PC5)**—This pin is a GPIO pin called PC5 when the SPI MOSI1 function is not being used. After reset, the default state is GPIO input.
SCK1	Input/ output	Input	**SPI1 Serial Clock (SCK1)**—This bidirectional pin provides a serial bit rate clock for the SPI. This gated clock signal is an input to a slave device and is generated as an output by a master device. Slave devices ignore the SCK signal unless the $\overline{SS}$ pin is active low. In both master and slave SPI devices, data is shifted on one edge of the SCK signal and is sampled on the opposite edge where data is stable. The driver on this pin can be configured as an open-drain driver by the SPI's WOM bit when this pin is configured for SPI operation.
PC6	Input or output		**Port C GPIO 6 (PC6)**—This pin is a GPIO pin called PC6 when the SPI SCK1 function is not being used. After reset, the default state is GPIO input.
$\overline{SS1}$	Input	Input	**SPI1 Slave Select ($\overline{SS1}$)**—This input pin is used to select a slave device before a master device can exchange data with the slave device. $\overline{SS}$ must be low before data transactions and must stay low for the duration of the transaction. The $\overline{SS}$ line of the master must be held high.
PC7	Input or output		**Port C GPIO 7 (PC7)**—This pin is a GPIO pin called PC7 when the SPI $\overline{SS1}$ function is not being used. After reset, the default state is GPIO input.

2.7 Synchronous Serial Interface (SSI) Signals

Table 2-12. Synchronous Serial Interface (SSI) Signals

Signal Name	Signal Type	State During Reset	Signal Description
STD	Output	Input	**SSI Transmit Data (STD)**—This output pin transmits serial data from the SSI Transmitter Shift Register.
PC8	Input or output		**Port C GPIO 8 (PC8)**—This pin is a GPIO pin called PC8 when the SSI STD function is not being used. After reset, the default state is GPIO input.
SRD	Input	Input	**SSI Receive Data (SRD)**—This input pin receives serial data and transfers the data to the SSI Receive Shift Register.
PC9	Input or output		**Port C GPIO 9 (PC9)**—This pin is a GPIO pin called PC9 when the SSI SRD function is not being used. After reset, the default state is GPIO input.
STCK	Input/ output	Input	**SSI Serial Transmit Clock (STCK)**—This bidirectional pin provides the serial bit rate clock for the transmit section of the SSI. The clock signal can be continuous or gated and can be used by both the transmitter and receiver in synchronous mode.
PC10	Input or output		**Port C GPIO 10 (PC10)**—This pin is a GPIO pin called PC10 when the SSI STCK function is not being used. After reset, the default state is GPIO input.
STFS	Input/ output	Input	**SSI Serial Transmit Frame Sync (STFS)**—This bidirectional pin is used by the Transmit section of the SSI as frame sync I/O or flag I/O. The STFS can be used by both the transmitter and receiver in synchronous mode. It is used to synchronize data transfer and can be an input or an output.
PC11	Input or output		**Port C GPIO 11 (PC11)**—This pin is a GPIO pin called PC11 when the SSI STFS function is not being used. This pin is not required by the SSI in gated clock mode. After reset, the default state is input.
SRCK	Input/ output	Input	**SSI Serial Receive Clock (SRCK)**—This bidirectional pin provides the serial bit rate clock for the receive section of the SSI. The clock signal can be continuous or gated and can be used only by the receiver.
PC12	Input or output		**Port C GPIO 12 (PC12)**—This pin is a GPIO pin called PC12 when the SSI STD function is not being used. After reset, the default state is GPIO input.

Table 2-12. Synchronous Serial Interface (SSI) Signals (Continued)

Signal Name	Signal Type	State During Reset	Signal Description
SRFS	Input/ output	Input	**SSI Serial Receive Frame Sync (SRFS)**—This bidirectional pin is used by the receive section of the SSI as frame sync I/O or flag I/O. The STFS can be used only by the receiver. It is used to synchronize data transfer and can be an input or an output.
PC13	Input or output		**Port C GPIO 13 (PC13)**—This pin is a GPIO pin called PC13 when the SSI SRFS function is not being used. After reset, the default state is GPIO input.

2.8 Timer Module Signals

Table 2-13. Timer Module Signals

Signal Name	Signal Type	State During Reset	Signal Description
TIO01	Input/ output	Input	**Timer 0 and Timer 1 Input/Output (TIO01)**—This bidirectional pin receives external pulses to be counted by either the on-chip 16-bit Timer 0 or Timer 1 when configured as an input and external clocking is selected. The pulses are internally synchronized to the DSP core internal clock. When configured as an output, it generates pulses or toggles on a Timer 0 or Timer 1 overflow event. Selection of Timer 0 or Timer 1 is programmable through an internal register.
PC14	Input or output		**Port C GPIO 14 (PC14)**—This pin is a GPIO pin called PC14 when the Timer TIO01 function is not being used. After reset, the default state is GPIO input.
TIO2	Input/ output	Input	**Timer 2 Input/Output (TIO2)—**This bidirectional pin receives external pulses to be counted by the on-chip 16-bit Timer 2 when configured as an input and external clocking is selected. The pulses are internally synchronized to the DSP core internal clock. When configured as an output, it generates pulses or toggles on a Timer 2 overflow event.
PC15	Input or output		**Port C GPIO 15 (PC15)**—This pin is a GPIO pin called PC15 when the Timer TIO2 function is not being used. After reset, the default state is GPIO input.

2.9 JTAG/OnCE Port Signals

Table 2-14. JTAG/OnCE Port Signals

Signal Name	Signal Type	State During Reset	Signal Description
TCK	Input	Input, pulled high internally	**Test Clock Input (TCK)**—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.
TMS	Input	Input, pulled high internally	**Test Mode Select Input (TMS)**—This input pin is used to sequence the JTAG test access port (TAP) controller's state machine. It is sampled on the rising edge of TCK and has an on-chip pull-up resistor.
TDI	Input	Input, pulled high internally	**Test Data Input (TDI)**—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.
TDO	Output	Tri-stated	**Test Data Output (TDO)**—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.
$\overline{\text{TRST}}$ $\overline{\text{DE}}$	Input Output	Input, pulled high internally	**Test Reset ($\overline{\text{TRST}}$)**—As an input, a low signal on this pin provides a reset signal to the JTAG TAP controller. **Debug Event ($\overline{\text{DE}}$)**—When programmed within the OnCE port as an output, $\overline{\text{DE}}$ provides a low pulse on recognized debug events; when configured as an output signal, the $\overline{\text{TRST}}$ input is disabled. To ensure complete hardware reset, $\overline{\text{TRST}}$/$\overline{\text{DE}}$ should be asserted whenever $\overline{\text{RESET}}$ is asserted. The only exception occurs in a debugging environment when a hardware DSP 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 $\overline{\text{TRST}}$/$\overline{\text{DE}}$. This pin is connected internally to a pull-up resistor.

Chapter 3
Memory Configuration and Operating Modes

This section describes in detail the on-chip memories and the operating modes of the DSP56824. In addition, the interrupt vectors, interrupt priority register (IPR), and peripheral memory map are provided.

3.1 DSP56824 Memory Map

The DSP56824 chip uses a Harvard memory architecture in which two independent memory spaces, X data memory and program memory, are provided. RAM and ROM are used for the on-chip data and program memory. The DSP56824 has 3.5K words of on-chip data RAM, 2K words of on-chip data ROM, 128 words of on-chip program RAM, and 32K words of on-chip program ROM. Both the program and data memories can be expanded off-chip. These memory spaces are shown in Figure 3-1 on page 3-2.

The operating mode control bits (MA and MB) in the operating mode register (OMR) control the program memory map and select the reset vector address. The external X memory (EX) control bit in the OMR controls the data memory map.

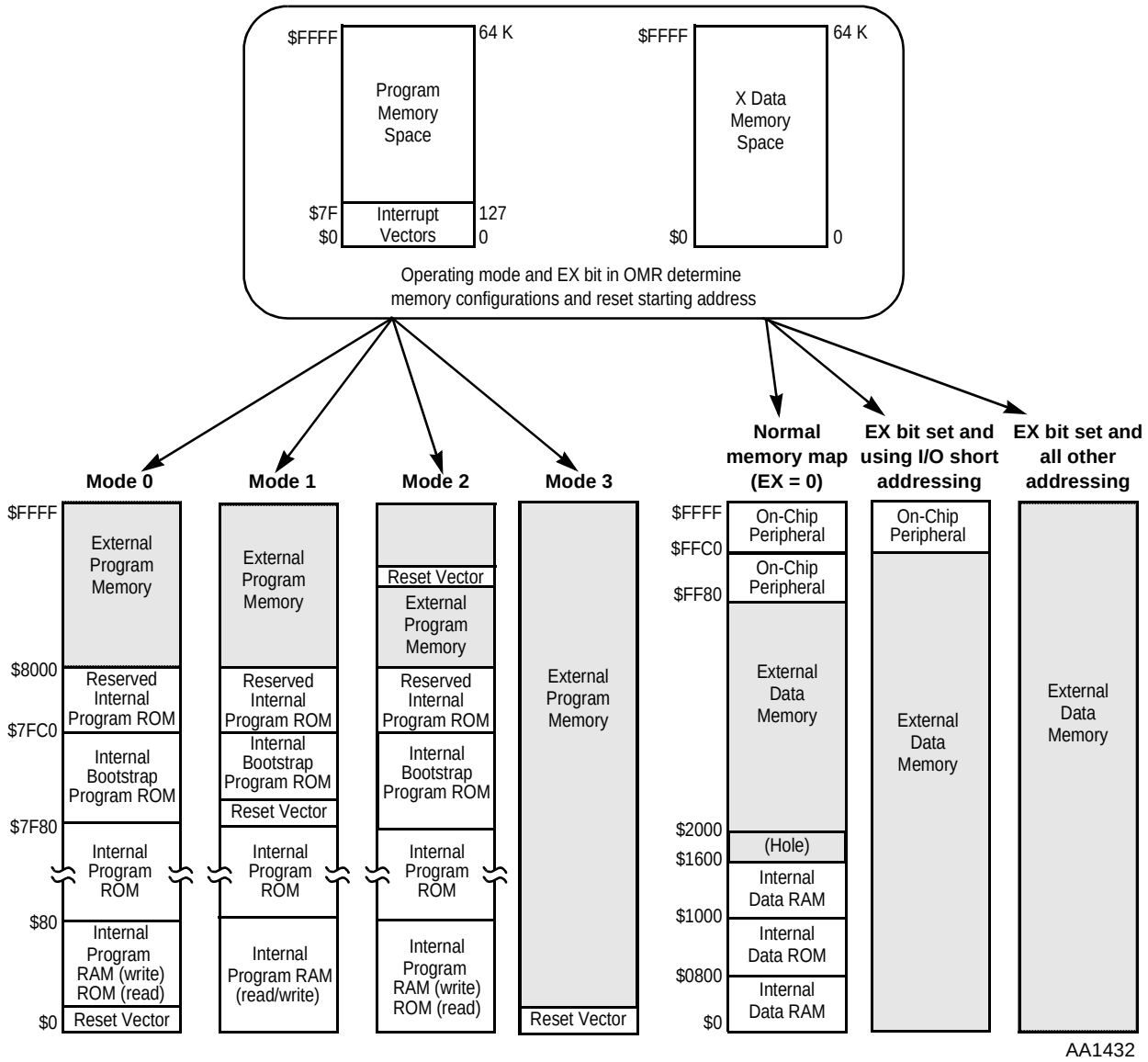


Figure 3-1. DSP56824 Memory Map

3.1.1 X Data Memory

The DSP56824 has 3.5K words of on-chip data RAM and 2K words of on-chip data ROM. Also, 128 additional data memory locations are reserved for on-chip peripheral registers (X:$FF80–$FFFF). The internal data memory map contains a non-accessible segment at X:$1600–$1FFF. When the EX bit is cleared, no memory can be accessed in these locations. When the EX bit is set, these locations are accessed as external memory. Any access to these addresses does not access external memory, except when the EX bit is set. These locations should be used by an application only when the EX bit is set exclusively.

For DSP56800 core instructions that perform two reads from the X data memory in a single instruction, the *second access* using the R3 pointer always occurs to *on-chip memory*. The internal X data memories decode only the upper 13 bits of XAB2, so the second read is Modulo 8192 to on-chip data memory x:$0, as follows:

```
        MOVE      #$A000, R3
        NOP
        MOVE       X:(R1)+,Y0      X:(R3)+, X0
```

The 2K on-chip X data ROM can only be accessed using the *second* access (that is, using the R3 pointer) of instructions that perform two parallel reads. On-chip X data ROM is not accessible on the first read or by instructions that perform single reads.

During development, the data that will finally be located in the X data ROM is often located off-chip. Data that will be mapped into on-chip ROM on a factory-programmed production part should be accessed from the external memory in the same manner as in the final target. Because the second read of a dual read instruction can never access off-chip memory, the **-o dbl** Assembler switch, which breaks a dual read into two equivalent instructions, must be used. This is shown in Example 3-1.

Example 3-1. Breaking a Read Instruction in Two

```
; Original instruction - X:(R3) accesses the internal ROM
      MOVE   X:(R0)+N,Y1    X:(R3)-,X0        ;X:(R3) cannot access off-chip memory

; Breaking the instruction into two equivalent
; instructions -- X:(R3) can access off-chip memory
      MOVE   X:(R0)+N,Y1
      MOVE                  X:(R3)-,X0        ;Now X:(R3) can access off-chip memory
```

The X memory can be expanded off-chip for a total of 65,536 addressable locations when the EX bit in the OMR is set. When the EX bit in the OMR is cleared, the X memory contains a total of 62,848 (65,536 – 2,688) addressable memory locations, including both on-chip and off-chip memory. The 2,560 locations of the memory hole and the 128 locations of the dedicated on-chip peripheral registers account for these 2,688 locations.

The external data memory bus access time is controlled by 4 bits of the bus control register (BCR) located at X:$FFF9. This register is shown in Figure 3-2 and described in detail in **Chapter 4, "External Memory Interface."**

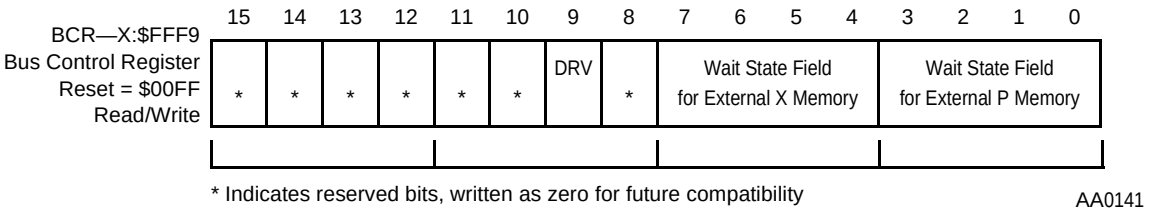


Figure 3-2. Bus Control Register Programming Model

Operation of the BCR is also controlled by the EX in the OMR. The EX bit determines the mapping of the X memory. Setting the EX bit completely disables the on-chip data memory and enables a full 64K *external* memory map. When the EX bit is set, the access time to any data memory is controlled by the BCR. The only exception to this rule is that if a MOVE or a bit-field instruction (such as BFSET, BFCLR, or BFCHG) is used with the I/O short addressing mode, the EX bit is ignored. This allows on-chip peripheral registers to be accessed when the EX bit is set.

NOTE:

When the EX bit is set, only the upper 64 memory-mapped peripheral registers (X:$FFC0 –$FFFF) are accessible with the I/O short addressing mode. The other 64 locations (X:$FF80–$FFBF) are not accessible in this case. An access to the lower 64 addresses results in an access to external X memory.

3.1.2 Operating Mode Register (OMR)

The operating mode register (OMR) is a 16-bit register that defines the current chip operating mode of the processor. The OMR bits are affected by processor reset, operations on the hardware stack (HWS), and instructions that directly reference the OMR. A nested DO loop also affects the OMR.

During processor reset, the chip operating mode bits (MA and MB) are loaded from the external mode select pins MODA and MODB, respectively. The OMR programming model is shown in Figure 3-3 and is described in the following subsections.

NOTE:

When a bit of the OMR is changed by an instruction, a delay of one instruction cycle is necessary before the new mode comes into effect.

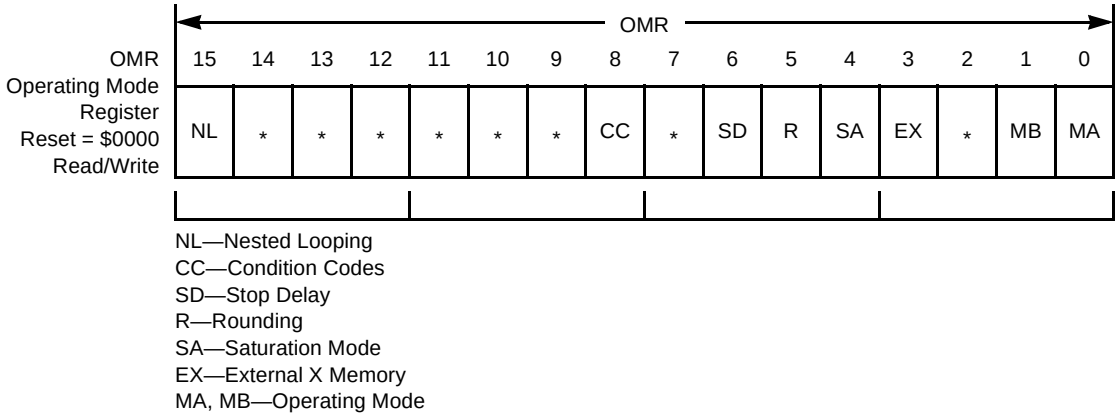


Figure 3-3. Operating Mode Register (OMR) Programming Model

3.1.2.1 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 that the program is currently in a nested DO loop (that is, two DO loops are active). When this bit is cleared, it indicates that 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 that the user never puts the processor in the reserved combination specified in Table 3-1. This can be avoided by ensuring that the LF bit is never cleared when the NL bit is set. The NL bit is cleared on processor reset.

Table 3-1. 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

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

The NL bit is also affected by any accesses to the HWS. Any MOVE instruction that writes this register copies the old contents of the LF bit into the NL bit and then sets the LF bit. Any reads of this register, such as from a MOVE or TSTW instruction, copy the NL bit into the LF bit and then clear the NL bit.

3.1.2.2 Reserved Bits—Bits 14–9

The OMR bits 14–9 are reserved. They are read as zero during DSP read operations and should be written with zero to ensure future compatibility.

3.1.2.3 Condition Codes (CC)—Bit 8

The condition code (CC) bit selects whether condition codes are generated using a 36-bit result from the multiply-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 31 of 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 are not affected by the CC bit. The CC bit is cleared by processor reset.

NOTE:

The unsigned condition tests used when 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.

3.1.2.4 Reserved Bit—Bit 7

The OMR bit 7 is reserved. It is read as zero during DSP read operations and should be written with zero to ensure future compatibility.

3.1.2.5 Stop Delay (SD)—Bit 6

The stop delay (SD) bit selects the delay that the DSP 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. Specific time intervals for stop delay are provided in the *DSP56824 Technical Data Sheet*. The SD bit is cleared by processor reset.

3.1.2.6 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 two rounding modes are discussed in detail in the *DSP56800 Family Manual*. The R bit is cleared by processor reset.

3.1.2.7 Saturation (SA)—Bit 4

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

The saturation logic operates by checking 3 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 3 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 3-2. The SA bit is cleared by processor reset.

Table 3-2. 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, 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.

Care should be used when consulting the N, C, and E condition codes after an operation when the SA bit is set. These condition codes are computed based on the value of the result *prior* to saturation. Thus, they may not have the expected values.

3.1.2.8 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, which uses 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. Refer to the *DSP56800 Family Manual* for a description of the dual read instructions.

NOTE:

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

The EX bit is cleared by processor reset.

3.1.2.9 Reserved Bit—Bit 2

The OMR bit 2 is reserved. It is read as zero during DSP read operations and should be written with zero to ensure future compatibility.

3.1.2.10 Operating Mode (MB, MA)—Bits 1–0

The chip operating mode (MB and MA) bits indicate the operating mode and memory maps of the DSP56824. These bits are loaded from the external mode select pins MODB and MODA on processor reset. After the DSP leaves the Reset state, MB and MA may be changed under program control. Operating modes for the DSP56824 are shown in Table 3-5 on page 3-12.

3.1.3 DSP56824 Status Register (SR)

The status register (SR) is a 16-bit register consisting of an 8-bit mode register (MR) and an 8-bit condition code register (CCR). The MR is the high-order 8 bits of the SR; the CCR is the low-order 8 bits. A full description of the SR is provided in the *DSP56800 Family Manual.* The programming model for the SR is shown in Figure 3-4.

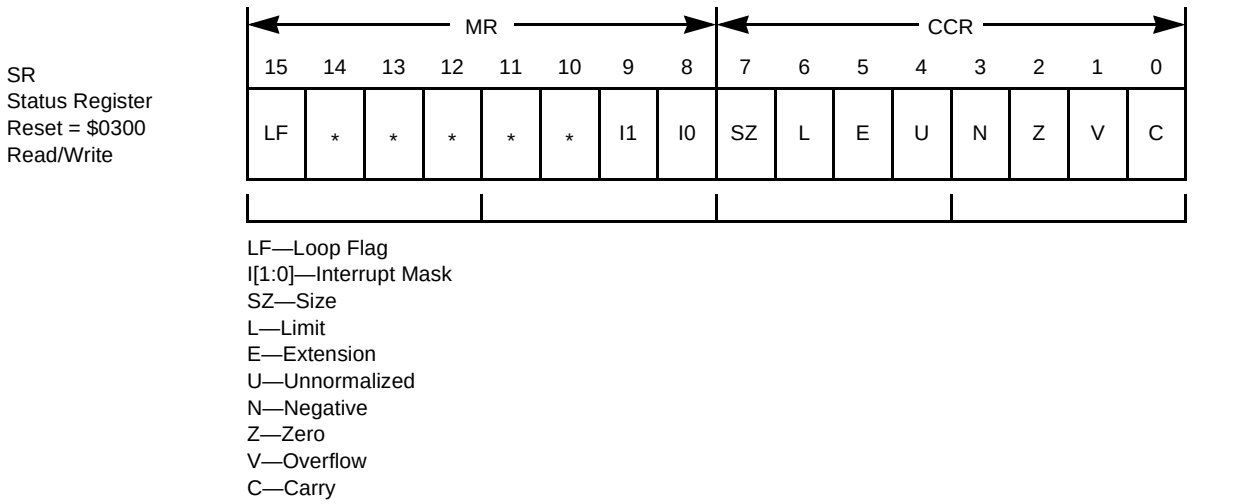


Figure 3-4. Status Register Programming Model

Within the SR, the MR is of special concern to DSP56824 users, as it allows masking or enabling interrupts for the on-chip peripherals. On reset, the SR is set to $0300, which enables interrupts having IPL 1 (listed in Table 3-7 on page 3-16) but masks interrupts with level IPL 0. All the on-chip peripheral

interrupts have IPL 0. To enable these interrupts, first selectively enable all desired interrupts for each peripheral using the IPR. After enabling the interrupts, set the I[1:0] bits in the SR to 01. This should be done for all applications that use the on-chip peripherals. See Section 3.3.1, “DSP56824 Interrupt Priority Register (IPR),” for more information on the IPR.

Table 3-3 shows the valid values to use for initializing the SR, the values for the interrupt mask bits, and the interrupt masking.

Table 3-3. Interrupt Mask Bit Definition

Value of SR	I[1:0]	Exceptions Permitted	Exceptions Masked
(Reserved)	00	(Reserved)	(Reserved)
$0100	01	IPL 0, 1	None
(Reserved)	10	(Reserved)	(Reserved)
$0300 (reset value)	11	IPL 1	IPL 0

NOTE:

Unless IPL 0 interrupts are enabled for on-chip peripheral interrupts in the SR, setting the IPR will have no effect.

For best results, use the following command to enable peripheral interrupts (IPL 0) in the SR:

```
BFCLR #$0200,SR
```

This command changes the I[1:0] bits from 11 to 01, clearing only the I1 bit and leaving all other bits in the SR unaffected. If it is necessary to temporarily disable peripheral interrupts, issue the following command:

```
BFSET #$0200,SR
```

This command changes the I[1:0] bits from 01 to 11, disabling all the peripheral interrupts. Afterwards, re-enable interrupts using the `BFCLR #$0200,SR` command.

3.1.4 On-Chip Peripheral Memory Map

Table 3-4 shows the on-chip memory-mapped I/O registers on the DSP56824.

Table 3-4. X I/O Registers

Address	Register
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 (Port A)

Table 3-4. X I/O Registers (Continued)

Address	Register
X:$FFF8	(Reserved) *
X:$FFF7	(Reserved) *
X:$FFF6	(Reserved) *
X:$FFF5	(Reserved) *
X:$FFF4	(Reserved) *
X:$FFF3	PCR1—PLL control register 1
X:$FFF2	PCR0—PLL control register 0
X:$FFF1	COPCTL—COP/RTI control register
X:$FFF0	COPCNT—COP/RTI count register (read-only) COPRST—COP reset register (write-only)
X:$FFEF	PCD—Port C data register
X:$FFEE	PCDDR—Port C data direction register
X:$FFED	PCC—Port C control register
X:$FFEC	PBD—Port B data register
X:$FFEB	PBDDR—Port B data direction register
X:$FFEA	PBINT—Port B interrupt register
X:$FFE9	(Reserved) *
X:$FFE8	(Reserved) *
X:$FFE7	(Reserved) *
X:$FFE6	SPCR1—SPI1 control register
X:$FFE5	SPSR1—SPI1 status register
X:$FFE4	SPDR1—SPI1 data register
X:$FFE3	(Reserved) *
X:$FFE2	SPCR0—SPI0 control register
X:$FFE1	SPSR0—SPI0 status register
X:$FFE0	SPDR0—SPI0 data register
X:$FFDF	TCR01—Timer 0 and 1 control register
X:$FFDE	TPR0—Timer 0 preload register
X:$FFDD	TCT0—Timer 0 count register

Table 3-4. X I/O Registers (Continued)

Address	Register
X:$FFDC	TPR1—Timer 1 preload register
X:$FFDB	TCT1—Timer 1 count register
X:$FFDA	TCR2—Timer 2 control register
X:$FFD9	TPR2—Timer 2 preload register
X:$FFD8	TCT2—Timer 2 count register
X:$FFD7	(Reserved) *
X:$FFD6	(Reserved) *
X:$FFD5	STSR—SSI time slot register
X:$FFD4	SCRRX—SSI receive control register
X:$FFD3	SCRTX—SSI transmit control register
X:$FFD2	SCR2—SSI control register 2
X:$FFD1	SCSR—SSI control/status register
X:$FFD0	SRX—SSI receive register (read-only) STX—SSI transmit register (write-only)
X:$FFCF	(Reserved) *
X:$FFCE	(Reserved) *
X:$FFCD	(Reserved) *
X:$FFCC	(Reserved) *
X:$FFCB	(Reserved) *
X:$FFCA	(Reserved) *
X:$FFC9	(Reserved) *
X:$FFC8	(Reserved) *
X:$FFC7	(Reserved) *
X:$FFC6	(Reserved) *
X:$FFC5	(Reserved) *
X:$FFC4	(Reserved) *
X:$FFC3	(Reserved) *
X:$FFC2	(Reserved) *

Table 3-4. X I/O Registers (Continued)

Address	Register
X:$FFC1	(Reserved) *
X:$FFC0	(Reserved) *
Note: *To ensure compatibility with other DSP56800 family members, these reserved addresses should not be used.	

3.1.5 Program Memory Map

The DSP56824 chip has 128 words of on-chip program RAM and 32K words of on-chip program ROM. The first 128 locations of program memory can be either ROM or RAM locations, depending on the operating mode selected and on whether the access is a read or a write. Program memory can be expanded off-chip to a maximum of 65,536 addressable locations. The external data bus access time is controlled by 4 bits of the BCR, shown in Figure 3-2 on page 3-6. On-chip program RAM can hold a combination of interrupt vectors and program code, and this memory space can be dynamically modified by the application being executed. In addition, this RAM can be used for program patching.

In the on-chip program ROM, locations P:$7F80–$7FBF are reserved for the bootstrap program. This program can be used to load internal or external program RAM from SPI0 (see Chapter 7, “Serial Peripheral Interface”**)** or from Port A **(see** Chapter 4, “External Memory Interface”**)**. After loading the RAM, the bootstrap program transfers program control to a user-defined starting address in RAM. See Appendix A, “Bootstrap Program,“ for more information. Locations P:$7FC0–$7FFF of on-chip program ROM are reserved for production test purposes and cannot be used.

Locations P:$8000–$FFFF are always addressed as external program memory, regardless of operating mode. When Mode 3 is selected, the entire P memory space is composed of external memory, and the reserved memory restrictions described in the previous paragraph do not apply. See Section 3.2, “DSP56824 Operating Modes.”

NOTE:

When using Mode 3, the user should ensure that applications targeted for other operating modes do not violate these P memory restrictions.

3.2 DSP56824 Operating Modes

The DSP56824 has four operating modes that determine the memory maps for program and data memories and the startup procedure when the chip leaves the Reset state. Operating modes can be selected either by applying the appropriate signals to the MODA and MODB pins during reset, or by writing to the OMR and changing the MA and MB bits, as shown in Table 3-5 on page 3-12.

Table 3-5. DSP56824 Program RAM Chip Operating Modes

MB or MODB Value	MA or MODA Value	Chip Operating Mode	Reset Vector	Program Memory Configuration	
				P:$7F–$0	P:$8000–$80
0	0	Mode 0 Single Chip Startup	Internal program ROM P:$0000 or P:$0002 (COP reset)	Read/fetch: internal program ROM Write: internal program RAM	All accesses: internal program ROM
0	1	Mode 1 Single Chip User	Internal program ROM P:$7F80 or P:$7F82 (COP reset)	All accesses: internal program RAM	All accesses: internal program ROM
1	0	Mode 2 Normal Expanded	External program memory P:$E000 or P:$E002 (COP reset)	Read/fetch: internal program ROM Write: internal program RAM	All accesses: internal program ROM
1	1	Mode 3 Development	External program memory P:$0000 or P:$0002 (COP reset)	All accesses: external program memory	All accesses: external program memory

The MODA and MODB pins are sampled as the DSP56824 leaves the Reset state, and the initial operating mode of the chip is set accordingly. After the Reset state is exited, the MODA and MODB pins become interrupt pins, $\overline{\text{IRQA}}$ and $\overline{\text{IRQB}}$. One of four initial operating modes is selected, based on the values detected on MODA and MODB:

- Single Chip mode (Mode 0 or Mode 1)
- Normal Expanded mode (Mode 2)
- Development mode (Mode 3)

Chip operating modes can also be changed by writing to the MB and MA bits in the OMR. Changing operating modes does not reset the DSP56824. To prevent an interrupt from going to the wrong memory location, interrupts should be disabled immediately before changing the OMR. Also, one no-operation (NOP) instruction should be included after changing the OMR to allow for remapping to occur.

NOTE:

On a Computer Operating Properly (COP) reset, the MA and MB bits (in the OMR) revert to the values originally latched from the MODA and MODB pins on deassertion of $\overline{\text{RESET}}$. These values determine the COP reset vector. For example, if the DSP56824 left hardware reset in Mode 2 and the mode bits in the OMR were later changed to specify Mode 3, a COP reset would use reset vector P:$E002 (for Mode 2) for its reset vector, and not P:$0002 (for Mode 3).

3.2.1 Single Chip Bootstrap Mode (Mode 0)

Mode 0 is the Single Chip bootstrap mode in which all the internal program and data memory space is enabled (see Figure 3-1 on page 3-2). Mode 0 can be entered by either pulling the MODA and MODB pins low before resetting the chip or by writing to the OMR and clearing the MA and MB bits. Writes to the lower 128 words of internal program space will write to the internal program RAM. The reset vector location in Mode 0 is P:$0000 in the internal program ROM (P:$0002 for COP timer reset).

Mode 0 is useful when exiting the Reset state for applications that execute primarily from internal program ROM. Write access to the internal program RAM allows an application to copy interrupt vectors and program code from program ROM to identically addressed locations in program RAM without changing operating modes.

3.2.2 Single Chip User (Mode 1)

Mode 1 is the Single Chip user mode in which 34,640 (32,768 – 128) words of internal program ROM are enabled for reads and fetches. All accesses to the lower 128 words of internal program space are to the internal program RAM. The reset vector location in Mode 1 is P:$7F80 in the internal program ROM (P:$7F82 for COP timer reset). The user should observe that these reset vectors are located in the area reserved for the bootstrap program.

Mode 1 is the ordinary user mode for applications that execute primarily from internal program ROM or for applications that must access the internal program RAM. The internal program RAM is typically loaded in Mode 0 or by the bootstrap program in Mode 1. See Appendix A, “Bootstrap Program,“ for more information on loading the internal program RAM in this manner.

3.2.3 Normal Expanded Mode (Mode 2)

In the Normal Expanded mode (Mode 2), all 32,768 words of internal program ROM are enabled for reads and fetches. Writes to the lower 128 words of internal program space will write to the internal program RAM. The reset vector location in Mode 2 is at P:$E000 in the external program RAM (P:$E002 for COP timer reset).

Mode 2 is identical to Mode 0 except that the reset vectors are in external memory. This feature provides additional flexibility for application development.

3.2.4 Development Mode (Mode 3)

Mode 3 is the Development mode, in which the entire 65,536 words of program memory space is external. No internal program memory space can be accessed. The reset vector location in Mode 3 is at P:$0000 in the external program memory space (P:$0002 for COP timer reset). Mode 3 is the primary mode for application development on the DSP56824.

3.3 DSP56824 Reset and Interrupt Vectors

The interrupt vector map specifies the address to which the processor jumps when it recognizes an interrupt or encounters a reset condition. The instruction located at this address must be a jump to subroutine (JSR) instruction for an interrupt or for a reset. The interrupt vector map for a given chip is specified by all possible interrupt sources on the DSP56800 core, as well as from the peripherals. No interrupt priority level (IPL) is specified for hardware reset (assertion of the $\overline{\text{RESET}}$ pin) or for COP reset because these conditions reset the chip, and a reset takes precedence over all other interrupts.

Table 3-6 on page 3-16 provides the interrupt priority structure for the DSP56824, including on-chip peripherals. Table 3-7 on page 3-16 lists the reset and interrupt vectors for the DSP56824. A full description of interrupts is provided in the *DSP56800 Family Manual.*

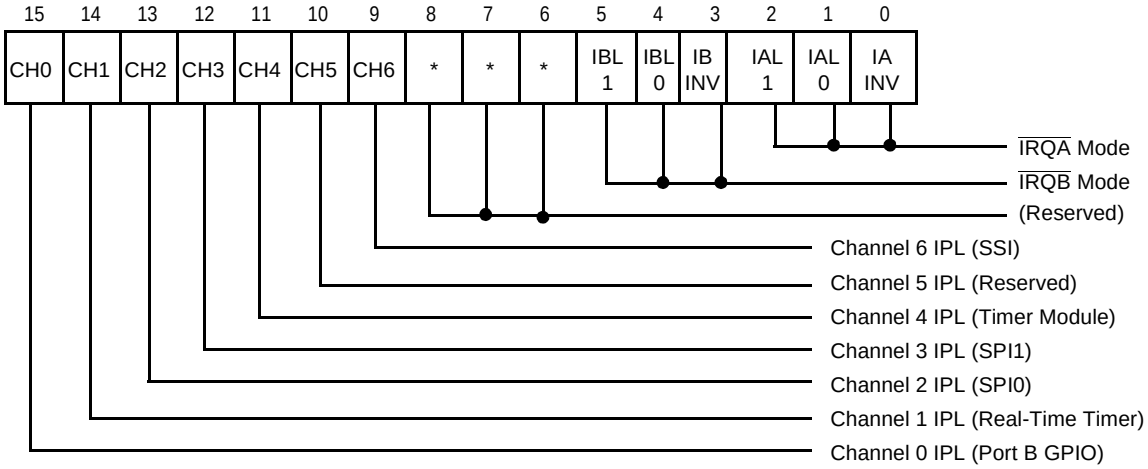
NOTE:

In Mode 2, the hardware and COP reset vectors are both located at address $E000 in external memory. In Mode 1, the hardware reset vector is located at address $7F80 and the COP reset vector is at $7F82. In Modes 0 and 3, the hardware reset vector is at $0000 and the COP reset vector is at $0002.

3.3.1 DSP56824 Interrupt Priority Register (IPR)

The 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 DSP56800 core contains seven interrupt channels for use by peripherals, in addition to the $\overline{\text{IRQ}}$ interrupts and the interrupts provided by the core. The IPL for each on-chip peripheral device (interrupt channels 0–6) and for each external source ($\overline{\text{IRQA}}$ and $\overline{\text{IRQB}}$) can be individually enabled or disabled under software control. In addition, 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 using the SR. Table 3-6 on page 3-16 shows the interrupt priority order. Figure 3-5 on page 3-15 shows the IPR, and Figure 3-6 on page 3-15 shows how the IPR bits are programmed. Unused bits are read as zero and should be written with zero to ensure future compatibility.



* Indicates reserved bits, read as zero and written with zero for future compatibility

AA1381

Figure 3-5. DSP56824 IPR Programming Model

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

AA1435

Figure 3-6. Interrupt Programming

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.

If the trigger mode is programmed to be edge sensitive in the IPR and the chip enters stop mode, one of two things could happen:

1. A valid level-sensitive value on the interrupt pin will bring the chip out of stop mode. The chip executes the next instruction following the STOP instruction; the interrupt service routine will not be executed.
2. A valid edge-sensitive interrupt will exit the chip from stop mode as well as service the interrupt by executing the interrupt service routine.

3.3.2 Interrupt Priority Structure

The following tables describe the programmable interrupt structure for the DSP56824. Table 3-6 on page 3-16 shows the interrupt priority structure, and Table 3-7 on page 3-16 shows the reset and interrupt vector map.

Table 3-6. Interrupt Priority Structure

Priority	Exception	IPR Bits
Level 1 (Non-maskable)		
Highest	Hardware $\overline{\text{RESET}}$	—
	COP timer RESET	—
	Illegal instruction trap	—
	Hardware stack overflow	—
	OnCE module instruction trap	—
Lower	SWI	—
Level 0 (Maskable)		
Higher	$\overline{\text{IRQA}}$ (external interrupt)	2, 1
	$\overline{\text{IRQB}}$ (external interrupt)	5, 4
	Channel 6 peripheral interrupt—SSI	9
	Channel 5 peripheral interrupt—Reserved	10
	Channel 4 peripheral interrupt—Timer module	11
	Channel 3 peripheral interrupt—SPI1	12
	Channel 2 peripheral interrupt—SPI0	13
	Channel 1 peripheral interrupt—Real-time timer	14
Lowest	Channel 0 peripheral interrupt—Port B GPIO	15

Table 3-7. Reset and Interrupt Vector Map

Interrupt Starting Address	IPL	Interrupt Source
\$0000/\$7F80[1]/\$E000[2]	—	Hardware $\overline{\text{RESET}}$
\$0000/\$7F82[1]/\$E002[2]	—	COP timer RESET
\$0004	—	(Reserved)
\$0006	1	Illegal instruction trap
\$0008	1	Software interrupt (SWI)
\$000A	1	Hardware stack overflow

Table 3-7. Reset and Interrupt Vector Map (Continued)

Interrupt Starting Address	IPL	Interrupt Source
$000C	1	OnCE module instruction trap
$000E	1	(Reserved)
$0010	0	$\overline{\text{IRQA}}$
$0012	0	$\overline{\text{IRQB}}$
$0014	0	Port B GPIO interrupt
$0016	0	Real-time interrupt
$0018	0	Timer 0 overflow
$001A	0	Timer 1 overflow
$001C	0	Timer 2 overflow
$001E	0	(Reserved)
$0020	0	SSI receive data with exception status
$0022	0	SSI receive data
$0024	0	SSI transmit data with exception status
$0026	0	SSI transmit data
$0028	0	SPI1 serial system
$002A	0	SPI0 serial system
$002C $007E	0 . . 0	Available for program code

1. Interrupt starting address when in Mode 1
2. Interrupt starting address when in Mode 2

Chapter 4
External Memory Interface

The DSP56824 provides a port for external memory interfacing. This section describes in detail the pins and programming specifics for this external memory interface, also known as Port A. This port provides 16 pins for an external address bus, 16 pins for an external data bus, and 4 pins for bus control. Together, these 36 pins comprise Port A.

4.1 External Memory Port Architecture

Figure 4-1 on page 4-2 shows the general block diagram of the DSP56824 I/O. It includes Ports A (described above), B, and C.

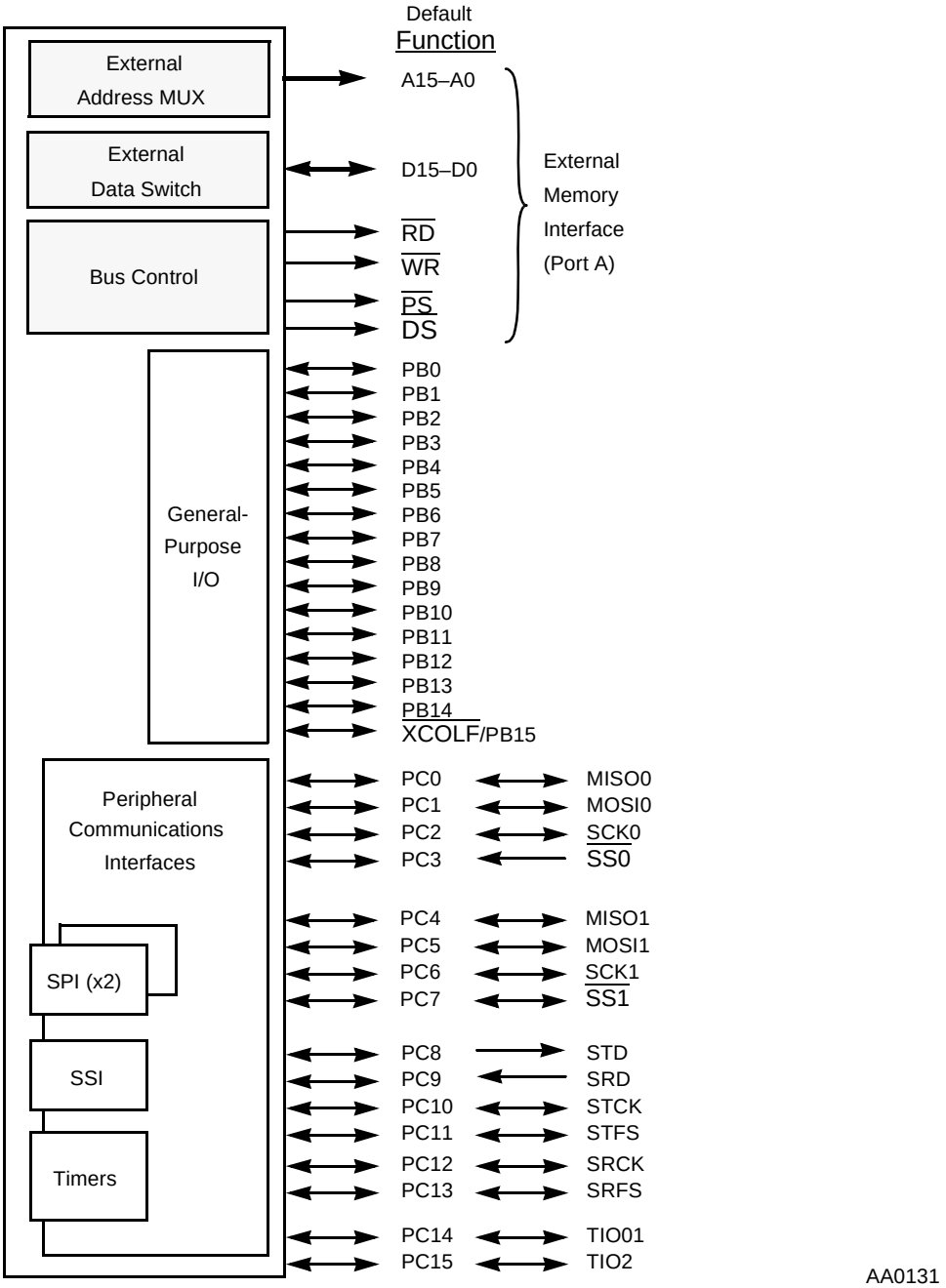


Figure 4-1. DSP56824 Input/Output Block Diagram

4.2 Port A Description

The external memory interface (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 external memory interface uses one programmable register for bus control, the bus control register (BCR).

External memory can be accessed at the maximum speed of the bus unit. In addition, software-controlled wait states can be introduced when accessing slower memories or peripherals. Wait states are programmable using registers. Figure 4-3 and Figure 4-4 on page 4-5 show examples of bus cycles with and without wait states.

4.2.1 Bus Control Register (BCR)

The bus control register (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 to 15 wait states. On processor reset, each wait state field is set to $F so that 15 wait states are inserted, allowing slower memory to be used immediately after reset. All other BCR bits are cleared on processor reset.

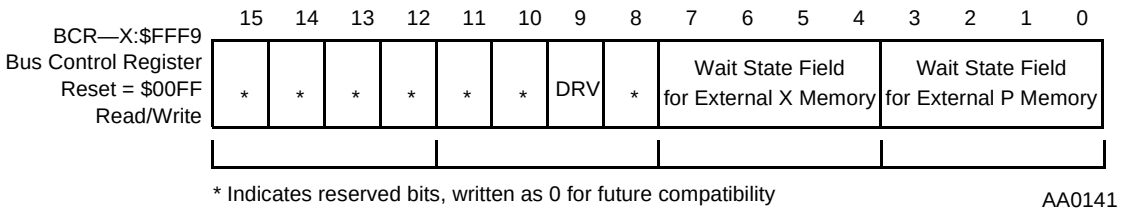


Figure 4-2. BCR Programming Model

4.2.1.1 Reserved Bits—Bits 15–10

Bits 15–10 are reserved and are read as zero during read operations. These bits should be written with zero to ensure future compatibility.

4.2.1.2 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. Table 4-2 on page 4-6 and Table 4-3 on page 4-7 summarize the action of the DRV bit. The DRV bit is cleared on hardware reset.

4.2.1.3 Reserved Bit—Bit 8

Bit 8 is reserved and is read as zero during read operations. This bit should be written with zero to ensure future compatibility.

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

The wait state X data memory (WSX[3:0]) control bits allow for the programming of the wait states for external X data memory. Table 4-1 shows 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 4-1. Programming WSP[3:0] and WSX[3:0] Bits for Wait States

Bit String	Hex Value	Number of Wait States
0000	$0	0
0001	$1	1
0010	$2	2
0011	$3	3
0100	$4	4
0101	$5	5
0110	$6	6
0111	$7	7
1000	$8	8
1001	$9	9
1010	$A	10
1011	$B	11
1100	$C	12
1101	$D	13
1110	$E	14
1111	$F	15

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

The wait state program memory (WSP[3:0]) control bits allow for the programming of the wait states for external program memory. These bits are programmed as shown in Table 4-1.

Figure 4-3 on page 4-5 shows an example of bus cycles without wait states, and Figure 4-4 on page 4-5 shows an example of bus cycles with wait states. For more information on wait states, see the *DSP56824 Technical Data Sheet*.

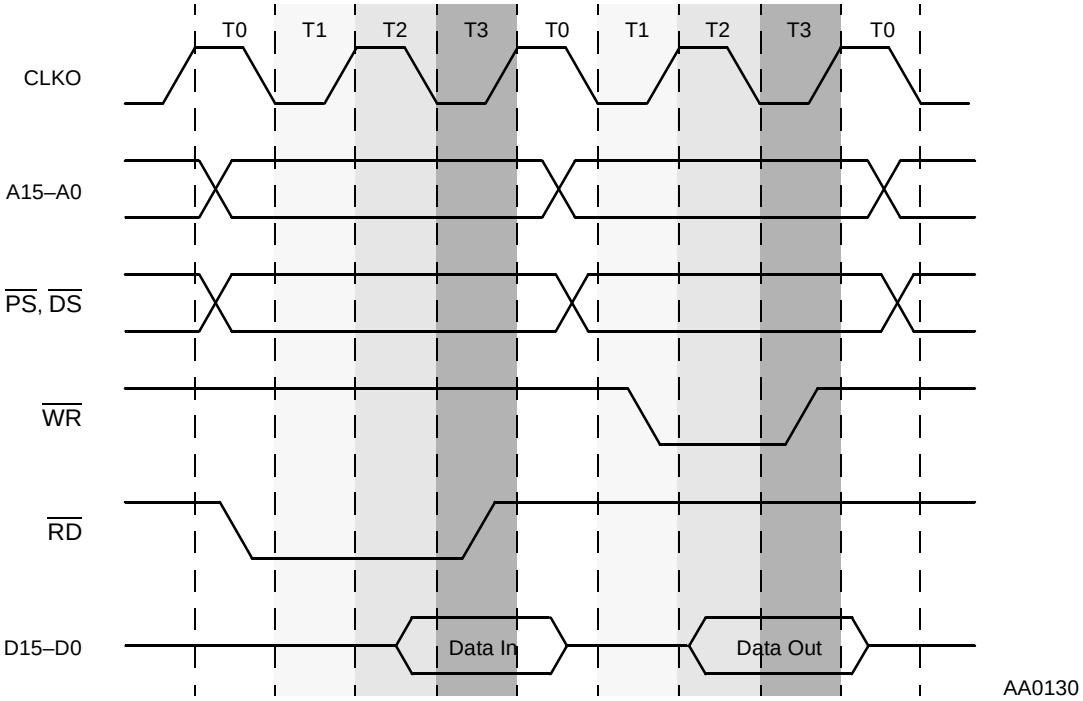


Figure 4-3. Bus Operation (Read/Write—Zero Wait States)

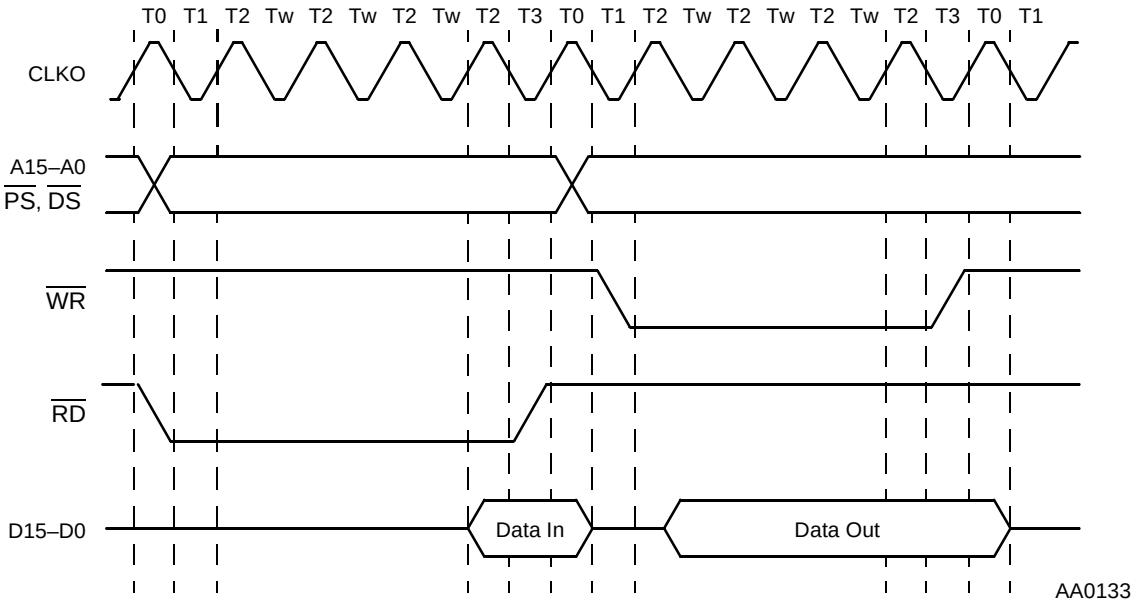


Figure 4-4. Bus Operation (Read/Write—Three Wait States)

4.2.2 Pins in Different Processing States

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

- Normal
- Exception
- Reset
- Wait
- Stop
- Debug

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

For the case where there is no external access on a particular instruction cycle, one of two things may occur. The external address bus and control pins can remain driven with their previous values, or they can be tri-stated.

The Reset mode always tri-states the external address bus and internally pulls control pins high. The 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 4-2 and Table 4-3 describe the operation of the external memory port in these different modes. The debug mode is a special state used to debug and test programming code. This state is discussed in detail in Chapter 9 of the *DSP56800 Family Manual*, but is not described in the following tables.

Table 4-2. 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 4-3. 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

The data lines are driven during normal mode, external fetch, only during an external write cycle.

Freescale Semiconductor, Inc.
ARCHIVED BY FREESCALE SEMICONDUCTOR, INC. 2005

Chapter 5
Port B GPIO Functionality

This section describes in detail the pins and programming specifics for Port B of the DSP56824. Port B is a dedicated general-purpose input/output (GPIO) port that provides 16 programmable I/O pins. Port B can be configured to generate an interrupt when a transition is detected on any of its lower eight pins, PB7–PB0, if they are configured as inputs. The upper eight pins, PB15–PB8, do not offer this interrupt functionality. During reset, the $\overline{\text{XCOLF}}$ functionality of PB15 allows selecting a low-frequency clock. For more information on $\overline{\text{XCOLF}}$, see the *DSP56824 Technical Data Sheet*. Figure 5-1 on page 5-2 shows a block diagram of the I/O of the DSP56824.

NOTE:

After reset, all Port B pins are inputs.

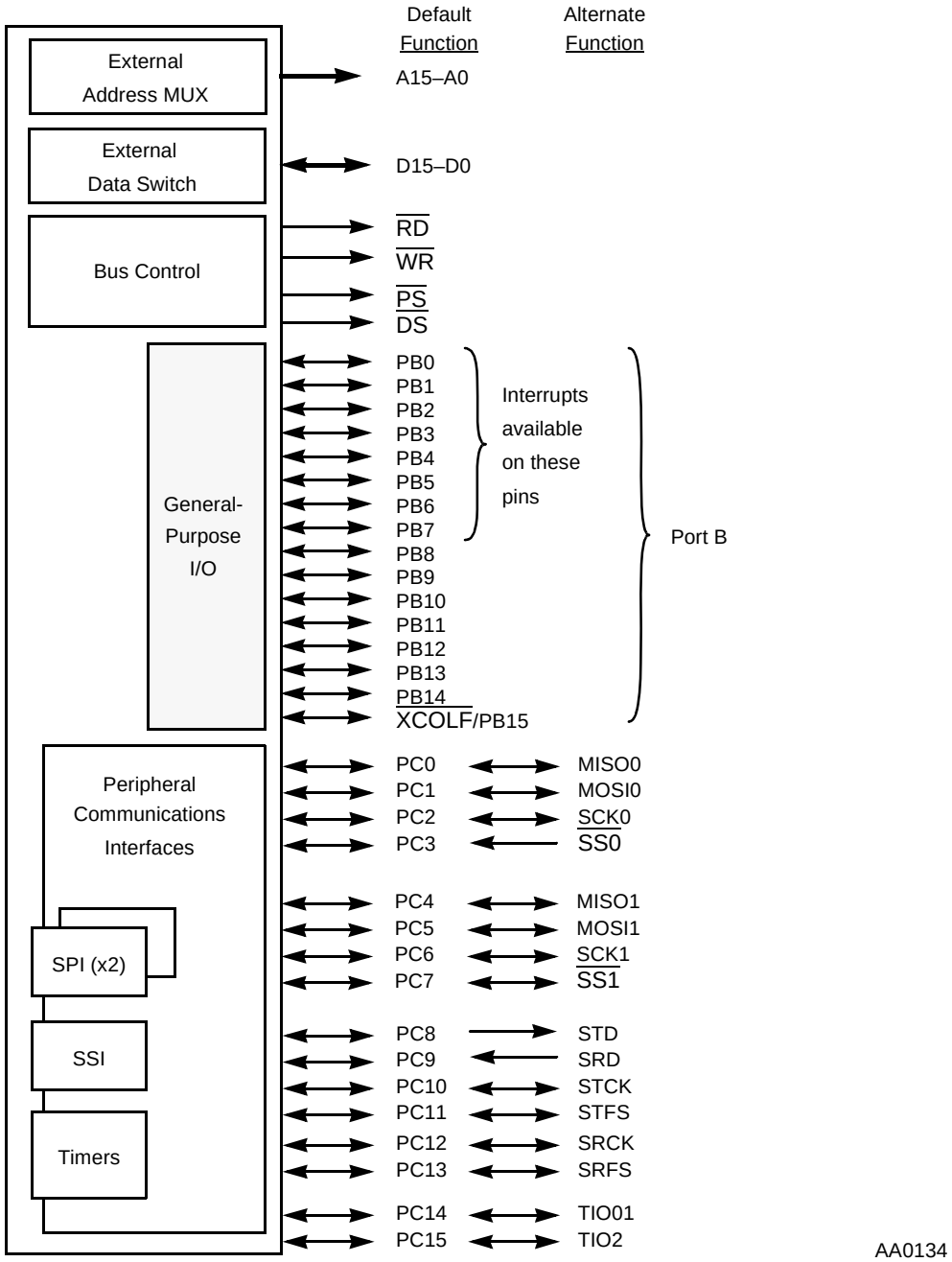


Figure 5-1. DSP56824 Input/Output Block Diagram

5.1 Port B Programming Model

Port B provides the following three read/write registers:

- Port B data direction register (PBDDR)
- Port B data (PBD) register
- Port B interrupt (PBINT) register

Figure 5-2 shows these the programming model for these registers. Bit-manipulation instructions can be used to access individual bits.

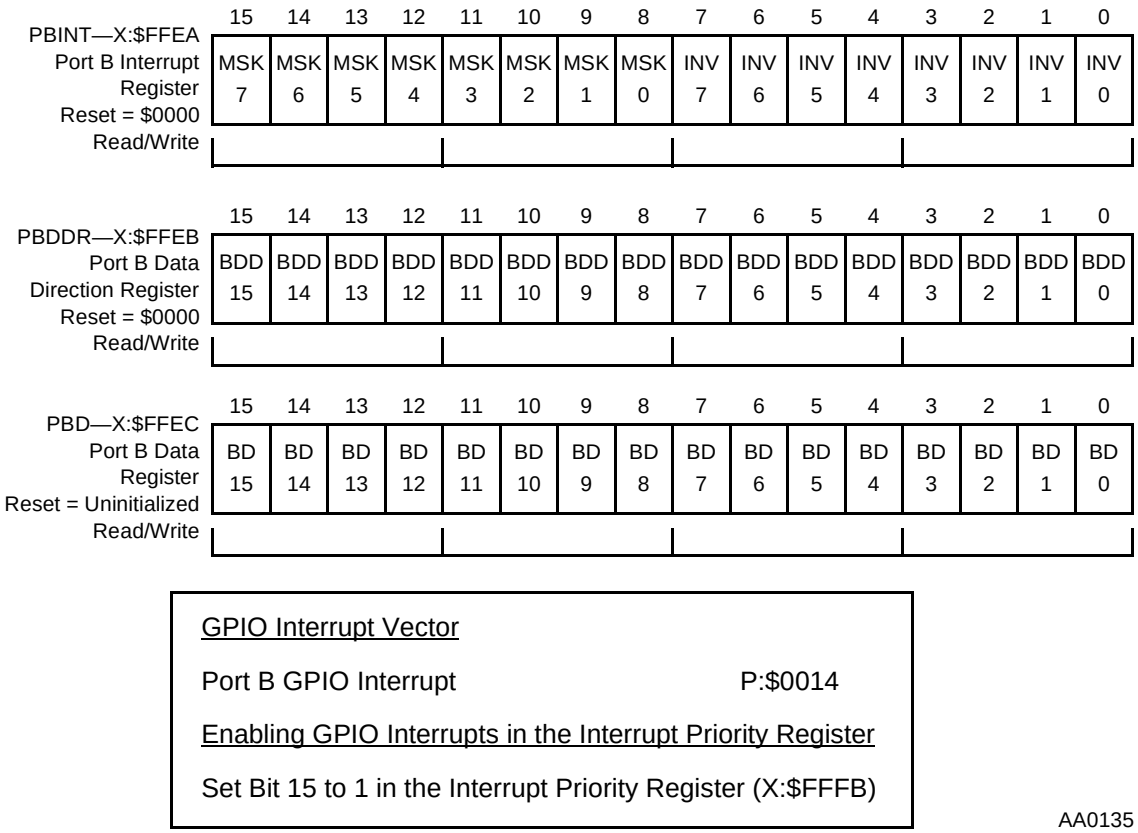


Figure 5-2. DSP56824 Port B Programming Model

5.1.1 Port B Data Direction Register (PBDDR)

The direction of each GPIO pin in Port B is determined by a corresponding control bit in the Port B data direction register (PBDDR). The port pin is configured as an input if the corresponding data direction register bit is cleared and is configured as an output if the corresponding data direction register bit is set. The PBDDR is cleared on processor reset, which configures all port pins as general-purpose input pins.

Table 5-1. PBDDR Bit Definition

BDD	Pin Direction
0	Input
1	Output

5.1.2 Port B Data (PBD) Register

The Port B data (PBD) register is a read/write register used to access the Port B GPIO pins. The value of each bit in the register is determined by the individual pin configuration. All 16 pins can be configured as general-purpose inputs (the default setting after reset) or as general-purpose outputs. When configured as inputs, the lowest eight pins can also be configured as interrupts.

5.1.2.1 PBD Bit Values for General-Purpose Inputs

If a Port B pin is configured as a general-purpose input, the corresponding bit value reflects the value driven into the pin when the register is read. Writing to the register transfers the written value through the register to the output latch, but no value is transferred to the pin.

5.1.2.2 PBD Bit Values for General-Purpose Outputs

If a Port B pin is configured as a general-purpose output, writing a value to the PBD register drives the value to the output latch for that pin and to the pin itself. Reading the PBD register returns the value latched to the pin.

5.1.2.3 PBD Bit Values for Interrupt Inputs

For pins PB7–PB0, if a pin is configured as an input, it can be configured as an interrupt. The method for programming a bit as an interrupt input is described in Section 5.1.3, “Port B Interrupt (PBINT) Register,” on page 5-5. If a pin is configured as an interrupt input, the respective bit in the PBD register reflects the state of the pin when an interrupt event was detected. Detection of a defined interrupt event after the last register read causes the values of all interrupt-enabled pins at the time the interrupt occurred to be locked into their respective bits in the PBD register. These values cannot be changed until a read of the register occurs again. Once the value is read, the bit values are unlocked and the pins operate like general-purpose input pins until the next interrupt occurs and locks the values for those pins into the register (until the next register read).

Table 5-2. Reading the PBD Register

BDD	MSK	Read of PBD Register Bit
0	0	Value on pin
0	1	Value in PBD latch
1	0	Value in PBD latch (and on pin)
1	1	Value in PBD latch (and on pin)

5.1.3 Port B Interrupt (PBINT) Register

The Port B interrupt (PBINT) register is a 16-bit read/write control register used to set up and control the capability of generating interrupts from the lower eight Port B GPIO pins. The bits of this register are defined in the following subsections.

NOTE:

The GPIO interrupt can be masked using the CH0 bit (bit 15) in the interrupt priority register (IPR). See Section 3.3.1, “DSP56824 Interrupt Priority Register (IPR),” on page 3-14 for information on the IPR.

5.1.3.1 Interrupt Mask (MSK[7:0])—Bits 15–8

The interrupt mask (MSK[7:0]) control bits are used to enable each of the lower eight Port B pins that can generate a GPIO interrupt. The programming for each of these 8 bits is described in Table 5-3 on page 5-5. The MSK[7:0] bits are cleared on hardware reset.

Table 5-3. MSK Bit Definition

MSK	Function
0	Pin masked from generating interrupt
1	Pin enabled for generating interrupt

NOTE:

Before any set MSK bit can enable a pin to generate an interrupt, the pin must be configured as an input pin in the PBDDR.

Port B programmable interrupts have the lowest priority of maskable interrupts. Table 3-6 on page 3-16 lists the interrupt priority order for the DSP56824.

5.1.3.2 Interrupt Invert (INV[7:0])—Bits 7–0

The interrupt invert (INV[7:0]) bits are used to individually program whether a rising or falling transition is detected on a pin. The programming for each of these 8 bits is described in Table 5-4. The INV[7:0] bits are cleared on hardware reset.

Table 5-4. INV Bit Definition

INV	Transition Detected
0	Rising Edge
1	Falling Edge

5.2 Port B Interrupt Generation

The following information describes the interrupt generation capabilities for the lower eight pins of Port B.

- Pins PB7–PB0 can be programmed to generate an interrupt by a transition on any of their input signals.
- Each pin can be programmed to detect a rising or a falling transition.
- Each pin can be individually enabled or masked.
- Each pin must be configured as an input pin to generate an interrupt.
- Any pin not used for generating an interrupt can be used as a GPIO pin.
- When the correct transition occurs on any pin enabled for interrupts, all pins enabled for interrupts are latched into the PBD register. Any pins not enabled for interrupt inputs are not latched.

As with all on-chip programmable peripheral interrupts for the DSP56824, the status register (SR) must first be set to enable maskable interrupts (interrupts of level IPL 1). See Section 3.1.3, “DSP56824 Status Register (SR),” on page 3-7 for more information about the SR. Next, the IPR must also be set to enable the interrupt. See Section 3.3.1, “DSP56824 Interrupt Priority Register (IPR),” on page 3-14 for more information about the IPR.

Set up the interrupt feature of Port B as follows:

1. Configure the SR to allow maskable interrupts.
2. Configure any pins desired for interrupt generation as inputs using the PBDDR.
3. Configure the associated INV bits for these pins, but leave associated MSK bits cleared (in the PBINT register).
4. Set the MSK bits for the associated pins.
5. Using the CH0 bit, enable the GPIO interrupt in the IPR. (See Section 3.3.1, “DSP56824 Interrupt Priority Register (IPR),” on page 3-14.)

Figure 5-3 shows the Port B interrupt block diagram.

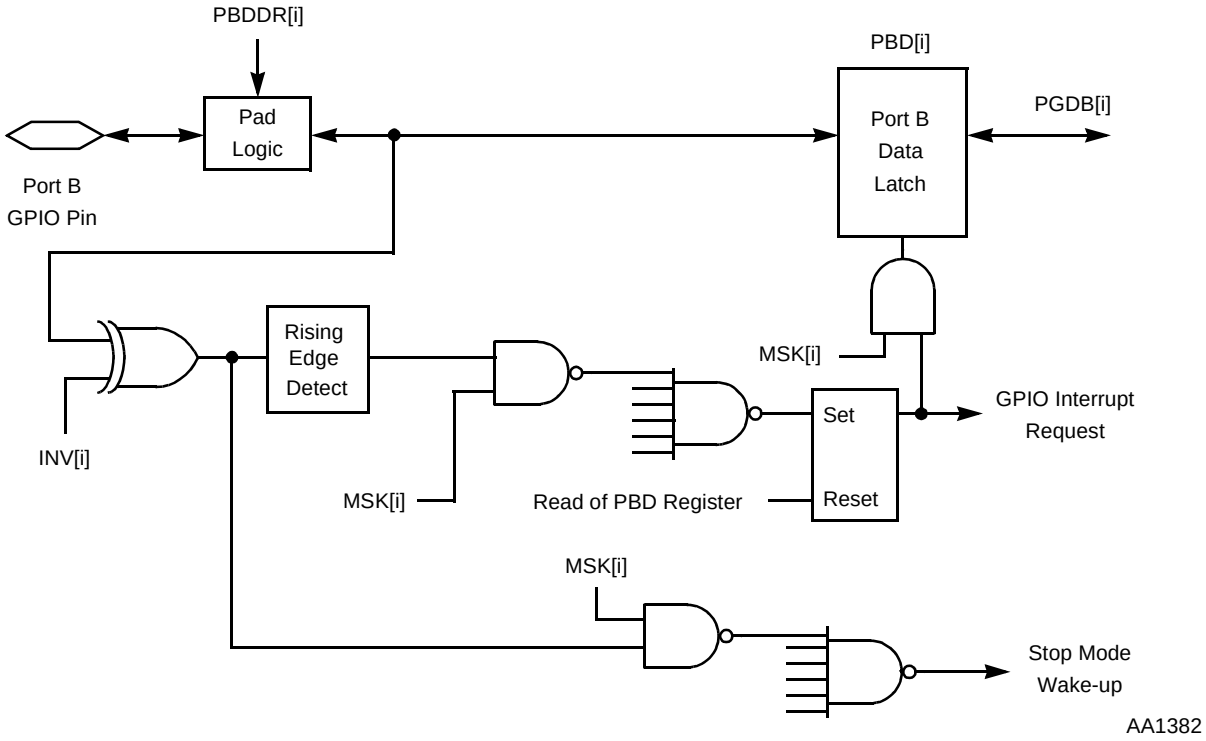


Figure 5-3. Port B Interrupt Block Diagram

5.3 Port B Programming Examples

Example 5-1 on page 5-8 through Example 5-4 on page 5-11 show how to use Port B pins for receiving data (configured for input), sending data (configured as output), and generating interrupts (configured as input).

5.3.1 Receiving Data on Port B

Example 5-1 shows how to configure Port B pins as inputs and use them for receiving data.

Example 5-1. Receiving Data on Port B

```
;**********************
;* INPUT example    *
;* for Port B       *
;* of DSP56824 chip *
;**********************
START               EQU     $0040       ; Start of program
BCR                 EQU     $FFF9       ; Bus Control Register
PBD                 EQU     $FFEC       ; Port B Data
PBDDR               EQU     $FFEB       ; Port B Data Direction Register
PBINT               EQU     $FFEA       ; Port B Interrupt Register
data_i              EQU     $0001       ; data input
;***************
;* Vector setup*
;***************
       ORG          P:$0000             ; Cold Boot
       JMP          START               ; also Hardware RESET vector (Mode 0, 1, 3)
       ORG          P:$E000             ; Warm Boot
       JMP          START               ; Hardware RESET vector (Mode 2)
       ORG          P:START             ; Start of program
;***************
;* General setup*
;***************
       MOVEP        #$0000,X:BCR        ; External Program memory has 0 wait states.
                                        ; External data memory has 0 wait states.
                                        ; Port A pins are tri-stated when no
                                        ; external access occurs.
;***************
;* Port B setup*
;***************
       MOVEP        #$0000,X:PBINT      ; Disable GPIO interrupt requests on
                                        ; all lower eight Port B pins (default).
       MOVEP        #$0000,X:PBDDR      ; Select pins PB0-PB15 as input (default).
;***************
;* Main routine*
;***************
                                        ;...
INPUT                                   ; Input Loop
                                        ; ...
       MOVEP        X:PBD,X0            ; Read PB0-PB15 into bits 0-15 of “data_i”.
       MOVE         X0,X:data_i         ; Memory to memory move requires two MOVEs.
                                        ; ...
       BRA          INPUT
```

5.3.2 Sending Data on Port B

Example 5-2 shows how to configure Port B pins as outputs and use them for sending data.

Example 5-2. Sending Data on Port B

```
;*********************
;* OUTPUT example   *
;* for Port B       *
;* of DSP56824 chip *
;*********************
START               EQU     $0040       ; Start of program
BCR                 EQU     $FFF9       ; Bus Control Register
PBD                 EQU     $FFEC       ; Port B Data
PBDDR               EQU     $FFEB       ; Port B Data Direction Register
PBINT               EQU     $FFEA       ; Port B Interrupt Register
data_o              EQU     $0001       ; data output
;***************
;* Vector setup*
;***************
       ORG          P:$0000             ; Cold Boot
       JMP          START               ; also Hardware RESET vector (Mode 0, 1, 3)
       ORG          P:$E000             ; Warm Boot
       JMP          START               ; Hardware RESET vector (Mode 2)
       ORG          P:START             ; Start of program
;***************
;* General setup*
;***************
       MOVEP        #$0000,X:BCR        ; External Program memory has 0 wait states.
                                        ; External data memory has 0 wait states.
                                        ; Port A pins are tri-stated when no
                                        ; external access occurs.
;***************
;* Port B setup*
;***************
       MOVEP        #$0000,X:PBINT      ; Disable GPIO interrupt requests on
                                        ; all lower eight Port B pins (default).
       MOVEP        #$FFFF,X:PBDDR      ; Select pins PB0-PB15 as input (default).
;***************
;* Main routine*
;***************
                                        ;...
OUTPUT                                  ; Output Loop
                                        ; ...
       MOVE         X:data_o,X0         ; Put bits 0-15 of “data_o” on pins PB0-PB15.
       MOVEP        X0,X:PBD            ; Memory to memory move requires two MOVEs.
                                        ; ...
       BRA          OUTPUT
```

5.3.3 Looping Data on Port B

Example 5-3 shows how to configure Port B pins to allow looping data from an output back to an input.

Example 5-3. Loop-Back Example

```
;*********************
;* LOOPBACK example*
;* for Port B       *
;* of DSP56824 chip *
;*********************
START               EQU     $0040       ; Start of program
BCR                 EQU     $FFF9       ; Bus Control Register
PBD                 EQU     $FFEC       ; Port B Data
PBDDR               EQU     $FFEB       ; Port B Data Direction Register
PBINT               EQU     $FFEA       ; Port B Interrupt Register
data_o              EQU     $0001       ; data output
data_i              EQU     $0001       ; data input
;***************
;* Vector setup*
;***************
        ORG         P:$0000             ; Cold Boot
        JMP         START               ; also Hardware RESET vector (Mode 0, 1, 3)
        ORG         P:$E000             ; Warm Boot
        JMP         START               ; Hardware RESET vector (Mode 2)
        ORG         P:START             ; Start of program
;***************
;* General setup*
;***************
        MOVEP       #$0000,X:BCR        ; External Program memory has 0 wait states.
                                        ; External data memory has 0 wait states.
                                        ; Port A pins are tri-stated when no
                                        ; external access occurs.
;***************
;* Port B setup*
;***************
        MOVEP        #$0000,X:PBINT     ; Disable GPIO interrupt requests on
                                        ; all lower eight Port B pins (default).
        MOVEP       #$FF00,X:PBDDR      ; Select pins PB0-PB7 as input and
                                        ; pins PB8-PB15 as output.
;***************
;* Main routine*
;***************
                                        ; ...
LOOPBACK                                ; Test Loop
        MOVE        X:data_o,X0         ; Put bits 8-15 of “data_o” on pins PB8-PB15.
        MOVEP       X0,X:PBD            ; Bits going to input pins are ignored.
                                        ; ...
       MOVEP        X:PBD,X0            ; Read PB0-PB7 into bits 0-7 of “data_i”.
       MOVE         X0,X:data_i         ; Bits 8-15 get values of PB8-PB15 as well.
                                        ; ...
       BRA          LOOPBACK
```

5.3.4 Generating Interrupts on Port B

Example 5-4 shows how to configure and use Port B for generating interrupts.

Example 5-4. Generating Interrupts on Port B

```
;*********************
;* INTERRUPT example*
;* for Port B       *
;* of DSP56824 chip *
;*********************
START               EQU     $0040       ; Start of program
BCR                 EQU     $FFF9       ; Bus Control Register
IPR                 EQU     $FFFB       ; Interrupt Priority Register
PBD                 EQU     $FFEC       ; Port B Data
PBDDR               EQU     $FFEB       ; Port B Data Direction Register
PBINT               EQU     $FFEA       ; Port B Interrupt Register
data_o              EQU     $0000       ; data output
data_i              EQU     $0001       ; data input
;***************
;* Vector setup*
;***************
       ORG          P:$0000             ; Cold Boot
       JMP          START               ; also Hardware RESET vector (Mode 0, 1, 3)
       ORG          P:$E000             ; Warm Boot
       JMP          START               ; Hardware RESET vector (Mode 2)
       ORG          P:$0014             ;
       JSR          GPIOISR             ; GPIO Interrupt vector
       ORG          P:START             ; Start of program
;***************
;* General setup*
;***************
       MOVEP        #$0000,X:BCR        ; External Program memory has 0 wait states.
                                        ; External data memory has 0 wait states.
                                        ; Port A pins are tri-stated when no
                                        ; external access occurs.
       BFCLR        #$0200,SR           ; Allow IPL (Interrupt Priority Level) 0.
                                        ; -- Enable maskable interrupts.
                                        ; -- (peripherals, and so on);**************
;* Port B setup*
;***************
       MOVEP         #$8000,X:PBINT     ; Enable GPIO interrupt requests for
                                        ; rising transitions on PB7.
       MOVEP        #$FF00,X:PBDDR      ; Select PB0-PB7 as input, PB8-PB15 as output
       BFSET        #$8000,X:IPR        ; Enable GPIO interrupts.

;***************
;* Main routine*
;***************
                                        ; ...
       MOVE         #$0000,X:data_o     ; Initialize “data_o”
TESTPAT                                 ; Test Pattern Loop
       MOVE         X:data_o,X0         ; Put bits 8-15 of “data_o” on pins PB8-PB15.
       MOVEP        X0,X:PBD            ; Bits going to input pins are ignored.
       ADD          #$0100,X0           ; Change output pattern: increment by 1.
       MOVE         X0,X:data_o         ; Increment upper byte by 1.
       MOVEP        X:PBD,X0            ; Read PB0-PB7 into bits 0-7 of “data_i”.
       MOVE         X0,X:data_i         ; Bits 8-15 get values of PB8-PB15 as well.
                                        ; ...
       BRA       TESTPAT
GPIOISR                                 ; GPIO Interrupt Service Routine
                                        ; interrupt code
       RTI
```

Freescale Semiconductor, Inc.
ARCHIVED BY FREESCALE SEMICONDUCTOR, INC. 2005

Chapter 6
Port C GPIO Functionality

The peripheral communications port (Port C) provides 16 multiplexed programmable I/O pins. These pins may be used as general-purpose input/output (GPIO) pins or allocated to on-chip peripherals—a timer module, two serial peripheral interfaces (SPIs), and a synchronous serial interface (SSI). (See Figure 6-1 on page 6-2.) Each pin is individually programmable.

This section describes how to program the pins for Port C and provides general information about their use as GPIO pins. Specifics for programming the various peripherals represented on Port C are provided in the appropriate sections, as follows:

- SPI module programming specifics are located in **Chapter 7, “Serial Peripheral Interface.”**
- SSI module programming specifics are located in **Chapter 8, “Synchronous Serial Interface.”**
- Timer module programming specifics are located in **Chapter 9, “Timers.”**

The Port C I/O interfaces are intended to minimize system chip count and “glue” logic in many DSP applications. Each I/O interface has its own control, status, and data registers that are treated as memory-mapped peripheral registers by the DSP56824. Each interface has several dedicated interrupt vector addresses and control bits to enable or disable interrupts. This minimizes the overhead associated with servicing the device, since each interrupt source may have its own service routine.

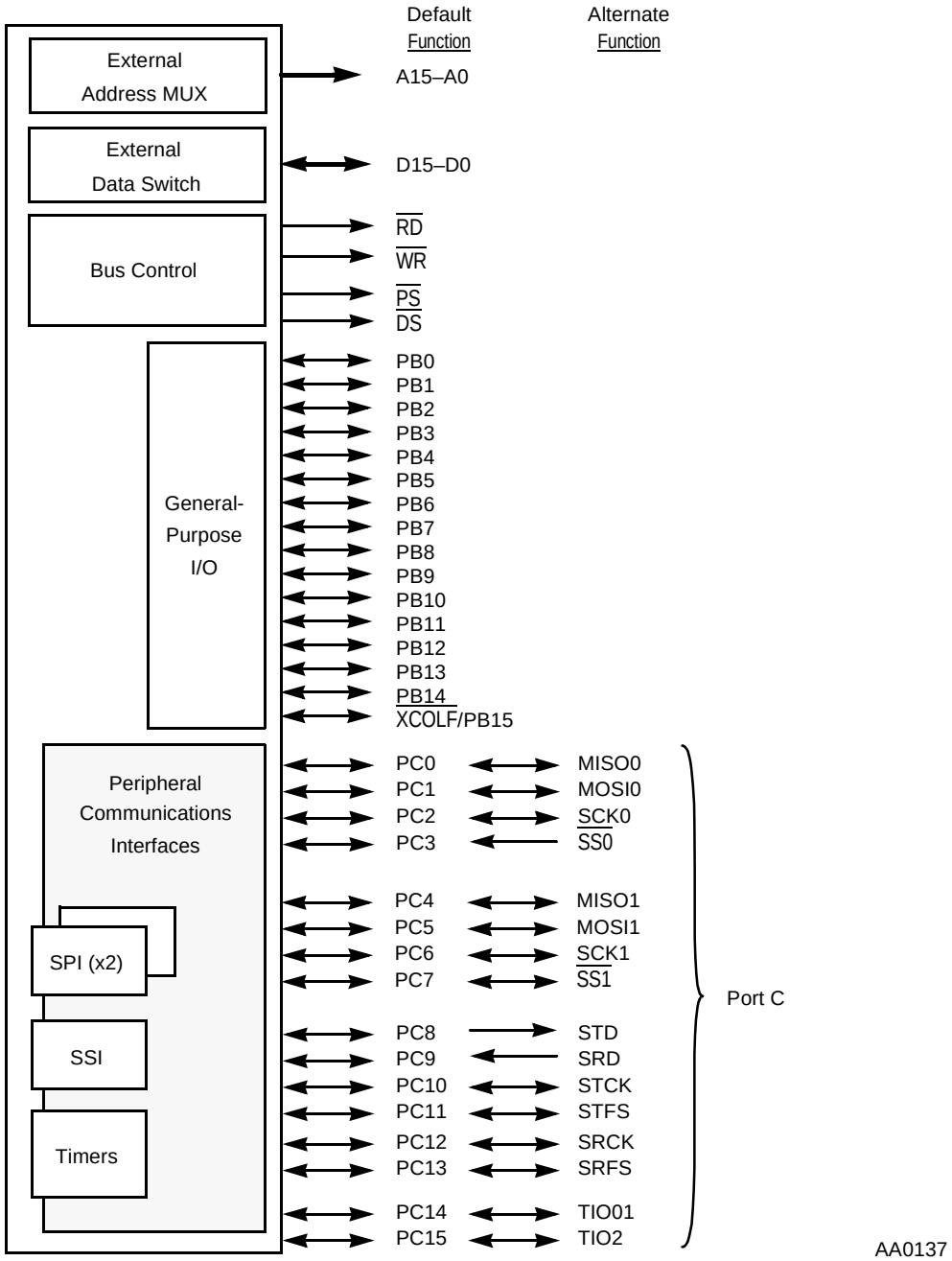


Figure 6-1. DSP56824 Input/Output Block Diagram

6.1 Port C Programming Model

Port C provides three read/write registers:

- Port C control (PCC) register
- Port C data direction register (PCDDR)
- Port C data (PCD) register

These registers are shown in Figure 6-2. The PCC register allows the programming of Port C pins as GPIO pins or as dedicated on-chip peripheral pins. The PCDDR specifies whether a pin programmed as a GPIO pin is an input or an output pin. The PCD register allows accessing data transmitted through a GPIO pin. Bit-manipulation instructions can be used to access individual bits.

Port C pins associated with the SPI can also be configured as open-drain drivers by the Wired-OR mode (WOM) control bit in the SPI control register (SPCR). More information on the SPCR is available in **Chapter 7, "Serial Peripheral Interface."**

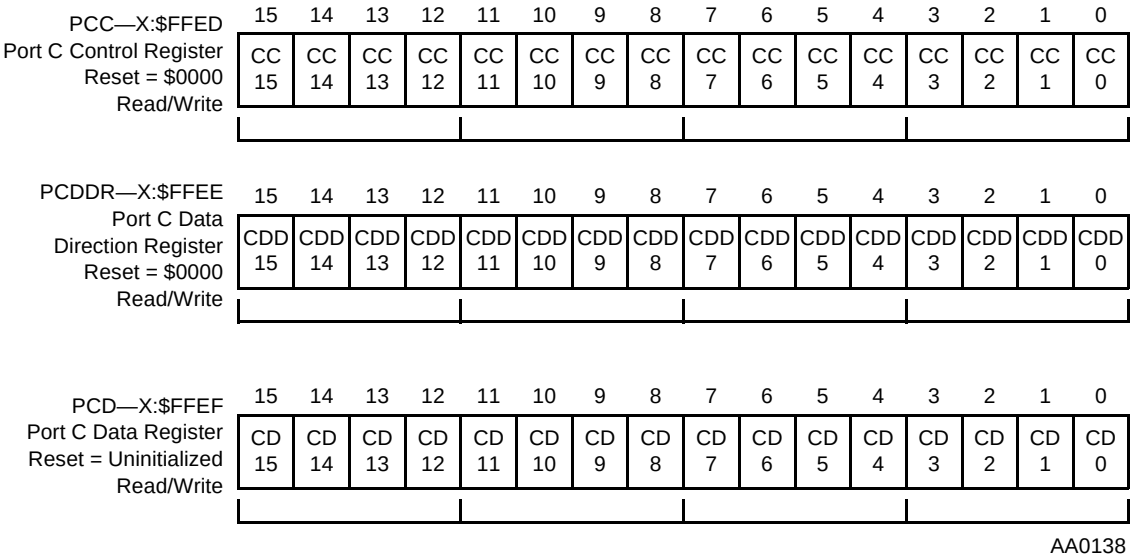


Figure 6-2. DSP56824 Port C Programming Model

6.1.1 Port C Control (PCC) Register

Port C pins can be programmed under software control as GPIO pins or as dedicated on-chip peripheral pins. The Port C control (PCC) register allows the port pins to be selected for one of these two functions. Each Port C pin is independently configured as a GPIO pin if the corresponding condition code (CC) bit is cleared, and is configured as an SPI, SSI, or timer pin if the corresponding PCC register bit is set. Table 6-1 shows how this bit is defined.

Table 6-1. PCC Bit Definition

CC	Pin Programmed As
0	General-purpose I/O pin
1	Dedicated peripheral pin

Unlike the pins on Port B, none of the GPIO pins on Port C can be configured to provide interrupts when configured as GPIO pins. When configured as peripheral pins, interrupts can be set within the peripheral. For example, when PC8–PC13 are configured to access the SSI functionality, level 0 maskable interrupts can be generated.

NOTE:

All CC bits are cleared on reset.

6.1.2 Port C Data Direction Register (PCDDR)

If a port pin is selected as a GPIO pin, the direction of that pin is determined by a corresponding bit in the PCDDR. The port pin is configured as an input if the corresponding CDD bit is cleared, and is configured as an output if the corresponding CDD bit is set. Table 6-2 shows how this bit is defined.

Table 6-2. PCDDR Bit Definition

CDD	Pin Direction
0	Input
1	Output

All PCC register bits and PCDDR bits are cleared on processor reset, configuring all Port C pins as general-purpose input pins. If the port pin is selected as an on-chip peripheral pin, the corresponding data direction bit is ignored and the direction of that pin is determined by the operating mode of the on-chip peripheral.

NOTE:

All CDD bits are cleared on reset.

6.1.3 Port C Data (PCD) Register

The Port C data (PCD) register allows accessing data through a port pin that has been configured as a GPIO pin. Data written to the PCD register is stored in an output latch. If the port pin is configured as an output, the output latch data is driven out on the port pin. When the PCD register is read, the logic value on the output port pin is read. If the port pin is configured as an input, data written to the PCD register is still stored in the output latch but is not gated to the port pin. When the PCD register is read, the state of the port pin is read. That is, reading the port data register will reflect the state of the pins regardless of how they were configured.

When a port pin is configured as a dedicated on-chip peripheral pin (the corresponding CC bit is set), the PCD register reads the state of the input pin or output driver.

6.2 Port C Programming Examples

Example 6-1 on page 6-5 through Example 6-3 on page 6-7 show how to configure Port C pins as GPIO pins to use them for receiving data (configured as input) and for sending data (configured as output).

6.2.1 Receiving Data on Port C GPIO Pins

Example 6-1 shows how to configure Port C pins as GPIO pins and how to use them for receiving data.

Example 6-1. Receiving Data on Port C GPIO Pins

```
;*********************
;* INPUT example     *
;* for Port C        *
;* of DSP56824 chip  *
;*********************
START               EQU     $0040       ; Start of program
BCR                 EQU     $FFF9       ; Bus Control Register
PCC                 EQU     $FFED       ; Port C Control Register
PCD                 EQU     $FFEF       ; Port C Data
PCDDR               EQU     $FFEE       ; Port C Data Direction Register
data_i              EQU     $0001       ; data input
;**************
;* Vector setup*
;**************
;+-----------------------------------------------------------------------+
;| Note: Bootstrap ROM configures OMR (operating mode register) to set   |
;|       chip operating mode for Mode 2 (Normal Expanded Mode), then     |
;|       jumps to first location of internal program RAM (P:$0000).      |
;+-----------------------------------------------------------------------+
        ORG           P:$0000           ; Cold Boot
        JMP           START             ; also Hardware RESET vector (Mode 0, 1, 3)
        ORG          P:$E000            ; Warm Boot
        JMP          START              ; Hardware RESET vector (Mode 2)
        ORG          P:START            ; Start of program
;**************
;* General setup*
;**************
        MOVEP        #$0000,X:BCR       ; External Program memory has 0 wait states.
                                        ; External data memory has 0 wait states.
                                        ; Port A pins are tri-stated when no
                                        ; external access occurs.
;**************
;* Port C setup*
;**************
       MOVEP         #$0000,X:PCC       ; Configures PC0-PC15 as GPIO pins (default)
       MOVEP         #$0000,X:PCDDR     ; Selects pins PC0-PC15 as input (default)
;**************
;* Main routine*
;**************
                                        ; ...
INPUT                                   ; Input Loop
                                        ; ...
       MOVEP         X:PCD,X0           ; Read PC0-PC15 into bits 0-15 of “data_i”
       MOVE          X0,X:data_i        ; (memory to memory move requires two moves)
                                        ; ...
       BRA        INPUT
```

6.2.2 Sending Data on Port C GPIO Pins

Example 6-2 shows how to configure Port C pins as GPIO pins and how to use them for sending data.

Example 6-2. Sending Data on Port C GPIO Pins

```
;*********************
;* OUTPUT example   *
;* for Port C       *
;* of DSP56824 chip *
;*********************
START               EQU     $0040       ; Start of program
BCR                 EQU     $FFF9       ; Bus Control Register
PCC                 EQU     $FFED       ; Port C Control Register
PCD                 EQU     $FFEF       ; Port C Data
PCDDR               EQU     $FFEE       ; Port C Data Direction Register
data_o              EQU     $0000       ; data output
;*****************
;* Vector setup *
;*****************
       ORG          P:$0000             ; Cold Boot
       JMP          START               ; also Hardware RESET vector (Mode 0, 1, 3)
       ORG          P:$E000             ; Warm Boot
       JMP          START               ; Hardware RESET vector (Mode 2)
       ORG          P:START             ; Start of program
;**************
;* General setup*
;**************
       MOVEP        #$0000,X:BCR        ; External Program memory has 0 wait states.
                                        ; External data memory has 0 wait states.
                                        ; Port A pins are tri-stated when no
                                        ; external access occurs.
;**************
;* Port C setup*
;**************
       MOVEP         #$0000,X:PCC       ; Configure PC0-PC15 as GPIO pins (default).
       MOVEP         #$FFFF,X:PCDDR     ; Select pins PC0-PC15 as output.
;**************
;* Main routine*
;**************
                                        ; ...
OUTPUT                                  ; Output Loop
                                        ; ...
       MOVE         X:data_o,X0         ; Put bits 0-15 of “data_o” on pins PC0-PC15.
       MOVEP        X0,X:PCD            ; Memory to memory move requires two MOVEs.
                                        ; ...
       BRA          OUTPUT
```

6.2.3 Looping Data on Port C GPIO Pins

Example 6-3 shows how to configure Port C GPIO pins and then use them to loop data back from an output to an input.

Example 6-3. Loop-Back Example

```
;********************
;* LOOPBACK example *
;* for Port C       *
;* of DSP56824 chip *
;********************
;
START               EQU     $0040       ; Start of program
BCR                 EQU     $FFF9       ; Bus Control Register
PCC                 EQU     $FFED       ; Port C Control Register
PCD                 EQU     $FFEF       ; Port C Data
PCDDR               EQU     $FFEE       ; Port C Data Direction Register
data_o              EQU     $0000       ; data output
data_i              EQU     $0001       ; data input
;*****************
;* Vector setup *
;*****************
;
      ORG           P:$0000             ; Cold Boot
      JMP           START               ; also Hardware RESET vector (Mode 0, 1, 3)
      ORG           P:$E000             ; Warm Boot
      JMP           START               ; Hardware RESET vector (Mode 2)
      ORG           P:START             ; Start of program
;******************
;* General setup *
;******************
;
      MOVEP         #$0000,X:BCR        ; External Program memory has 0 wait states.
                                        ; External data memory has 0 wait states.
                                        ; Port A pins are tri-stated when no
                                        ; external access occurs.
;*****************
;* Port C setup *
;*****************
;
      MOVEP          #$0000,X:PCC       ; Configure PC0-PC15 as GPIO pins (default).
      MOVEP          #$FF00,X:PCDDR     ; Select pins PC0-PC7 as input and
                                        ; pins PC8-PC15 as output.
;*****************
;* Main routine *
;*****************
;
                                        ; ...
LOOPBACK ; Test Loop
      MOVE          X:data_o,X0         ; Put bits 8-15 of “data_o” on pins PC8-PC15.
      MOVEP         X0,X:PCD            ; Bits going to input pins are ignored.
                                        ; ...
      MOVEP         X:PCD,X0            ; Read PC0-PC7 into bits 0-7 of “data_i”.
      MOVE          X0,X:data_i         ; Bits 8-15 get values of PC8-PC15 as well.
                                        ; ...
      BRA    LOOPBACK
```

ARCHIVED BY FREESCALE SEMICONDUCTOR, INC. 2005
Freescale Semiconductor, Inc.

Chapter 7
Serial Peripheral Interface

This section discusses the architecture of the serial peripheral interface (SPI) provided on Port C, its pins, and its programming model. The section includes information on SPI system errors, a discussion of overrun on the SPI, correct programming of Port C when using the SPI, and low-power operation with the SPI.

The SPI is an independent, serial communications subsystem that allows the DSP56824 to communicate synchronously with peripheral devices such as LCD drivers, A/D subsystems, and microprocessors. More sophisticated uses, such as interprocessor communication in a multiple master system, are also easy to implement. The SPI can be configured as either a master or a slave device with high data rates. In Master mode, a transfer is initiated when data is written to the SPI data register (SPDR). In Slave mode, a transfer is initiated by the reception of a clock signal.

Clock control logic allows a selection of clock polarity and a choice of two fundamentally different clocking protocols to accommodate most available synchronous serial peripheral devices. In some cases, the phase and polarity are changed between transfers to allow a master device to communicate with peripheral slaves having different requirements. When the SPI is configured as a master, software selects one of eight different bit rates for the clock.

Error-detection logic is included to support interprocessor communications. A write-collision detector indicates when an attempt is made to write data to the serial shift register while a transfer is in progress. A multiple-master mode-fault detector automatically disables SPI output drivers if more than one microcontroller unit (MCU) simultaneously attempts to become bus master.

The DSP56824 provides two identical SPIs—SPI0 and SPI1. Figure 7-1 on page 7-2 shows SPI0 and SPI1.

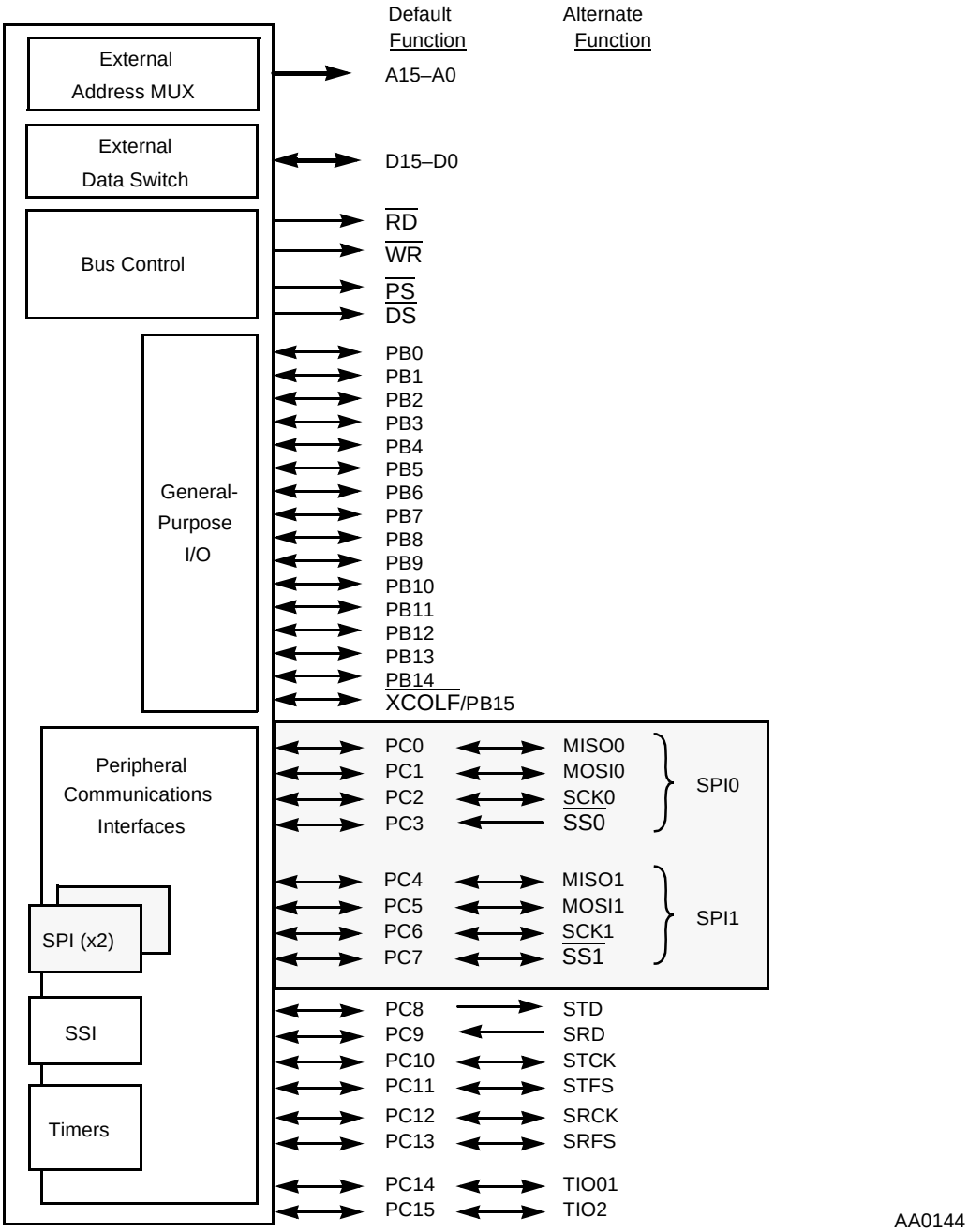


Figure 7-1. DSP56824 Input/Output Block Diagram

7.1 SPI Architecture

Figure 7-2 on page 7-3 shows a block diagram of the SPI subsystem. When an SPI transfer occurs, a byte is shifted out one data pin, while a different byte is simultaneously shifted in a second data pin. Another way to view this transfer is that an 8-bit shift register in the master device and another 8-bit shift register in the slave are connected as a circular 16-bit shift register. When a transfer occurs, this distributed shift register is shifted 8 bit positions; thus, the bytes in the master and slave are effectively exchanged.

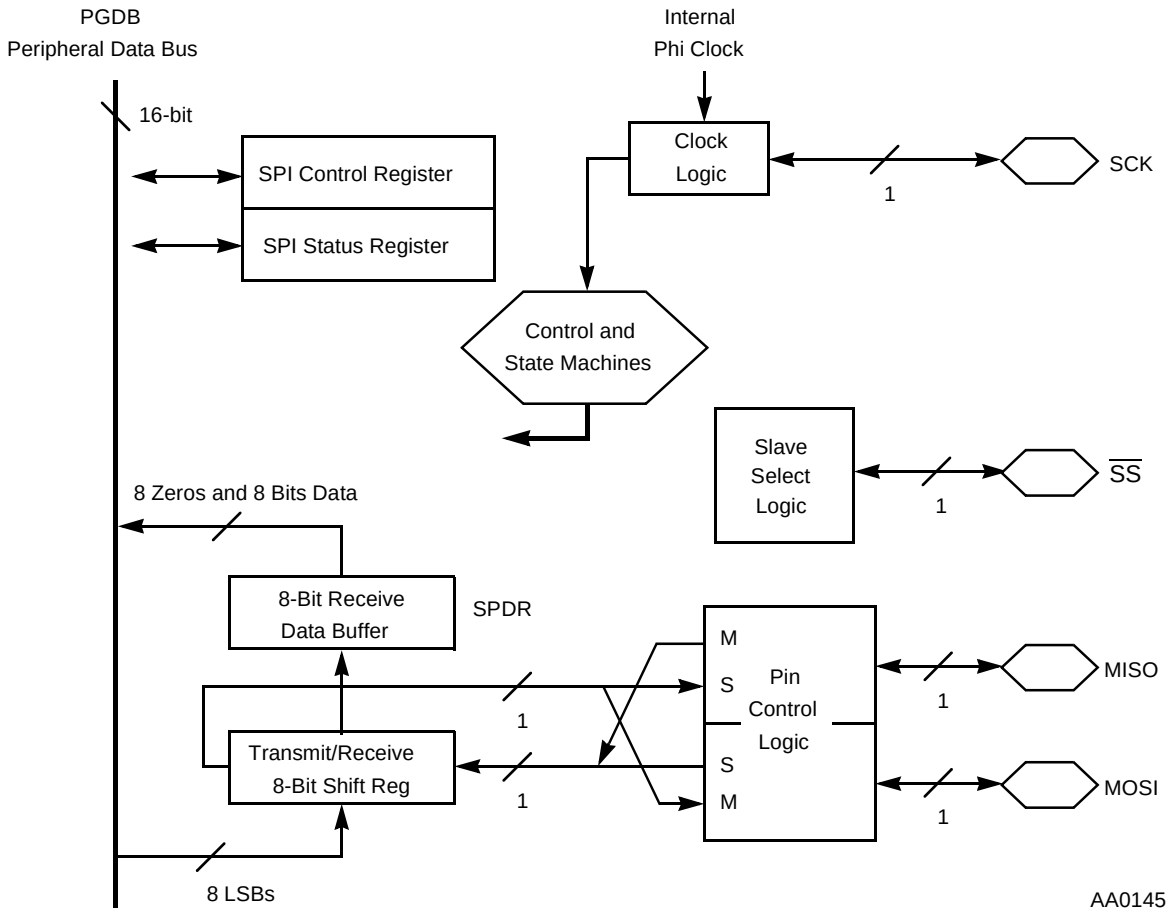


Figure 7-2. SPI Block Diagram

The central element in the SPI system is the block containing the shift register and receive data buffer. The system is single-buffered in the transmit direction and double-buffered in the receive direction. This means that new data for transmission cannot be written to the shifter until the previous transfer is complete. However, received data is transferred into a parallel read data buffer, so the shifter is free to accept a second serial byte. As long as the first byte is read out of the read data buffer before the next serial byte is ready to be transferred, no overrun condition occurs. A single memory address is used for reading data from the read buffer and writing data to the shifter.

The SPI status block represents the SPI status functions (transfer complete, write collision, and mode fault) located in the SPI status register (SPSR). The SPI control block contains the SPI control register (SPCR) used to set up and control the SPI system.

Figure 7-3 on page 7-4 and Figure 7-4 on page 7-4 illustrate the different transfer formats. SCK (the serial clock) is shown for each polarity (selected by CPL). Both master and slave timing are shown. Master in/slave out (MISO) and master out/slave in (MOSI) pins are connected between the master and slave. MISO is the slave output and MOSI is the master output. The slave select pin ($\overline{SS}$) shown is for the slave. The master's $\overline{SS}$ pin is held high but is not shown.

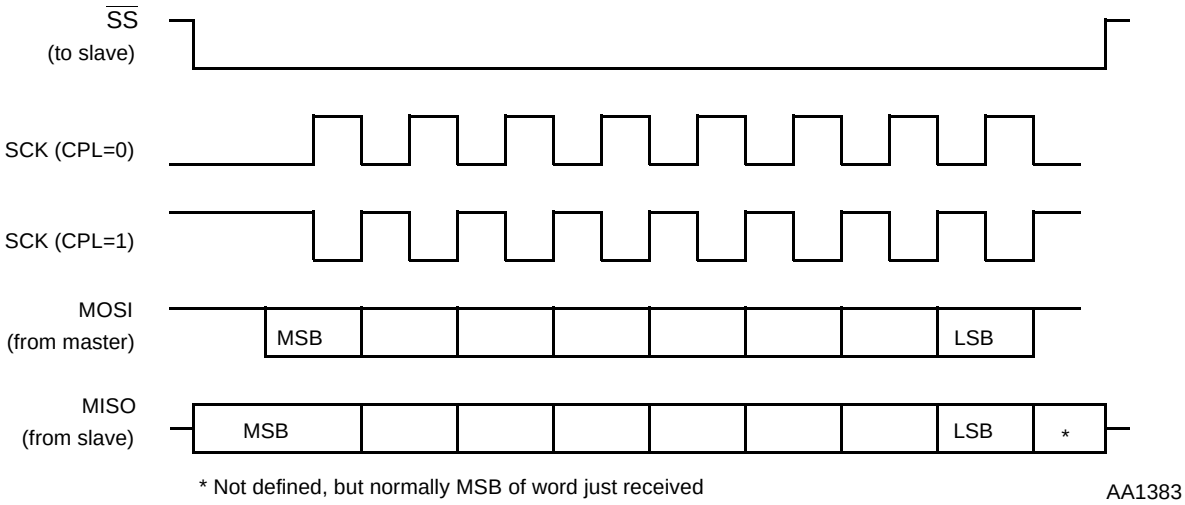


Figure 7-3. SPI Transfer with CPH = 0

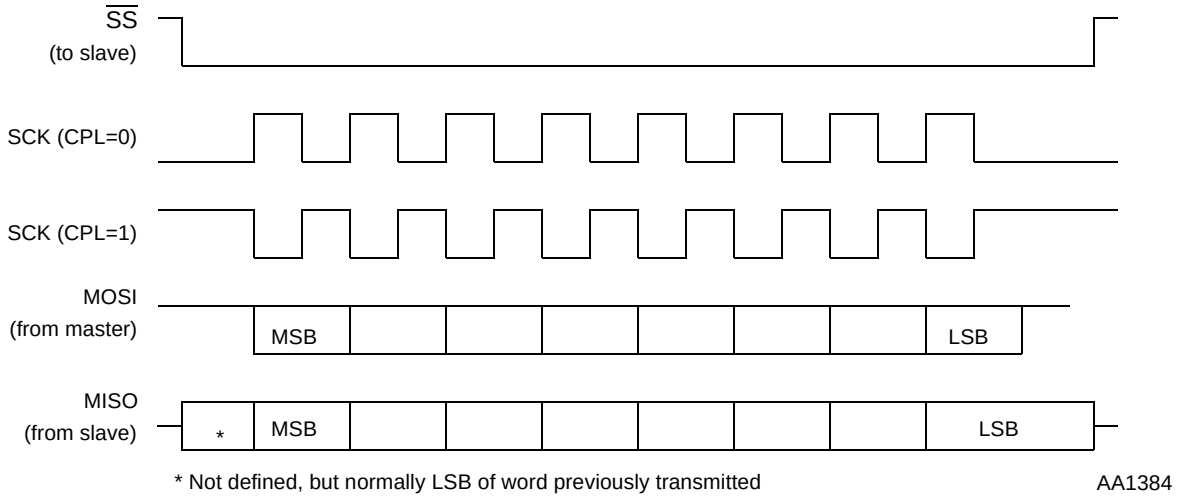


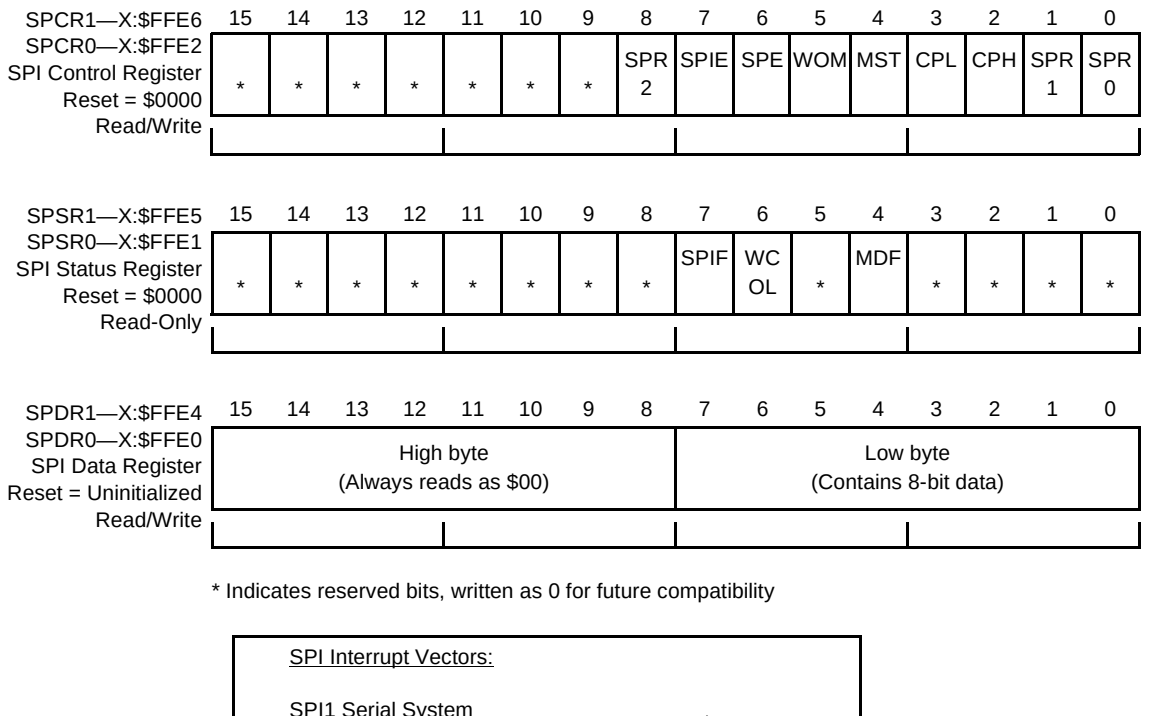
Figure 7-4. SPI Transfer with CPH = 1

7.2 SPI Programming Model

Each SPI peripheral provides the following registers:

- SPI control register (SPCR)
- SPI status register (SPSR)
- SPI data register (SPDR)

These registers are shown in Figure 7-5 on page 7-5. The descriptions of the registers in the following paragraphs apply equally to the corresponding registers on SPI0 and SPI1.



* Indicates reserved bits, written as 0 for future compatibility

SPI Interrupt Vectors:

SPI1 Serial System P:$0028
SPI0 Serial System P:$002A

Enabling SPI Interrupts in the IPR:

SPI1: Set bit 12 to 1 in the IPR (X:$FFFB).
SPI0: Set bit 13 to 1 in the IPR (X:$FFFB).

AA0147

Figure 7-5. SPI Programming Model

NOTE:

To use SPI0, the CC[3:0] bits in the PCC register must be correctly set. To use SPI1, the CC[7:4] bits in the PCC register must be correctly set. See Section 7.5, “Configuring Port C for SPI Functionality,” for more information.

7.2.1 SPI Control Registers (SPCR0 and SPCR1)

The SPI control registers (SPCR0 and SPCR1) are 16-bit, read/write registers used to set up and control the SPI0 and SPI1 peripherals, respectively. The bits within the registers are arranged and programmed identically. The only differences are that the two registers have different addresses and control different SPIs. Programming one register has no effect on the other register.

The WOM, MST, CPL, CPH, and the SPR[2:0] bits should not be changed when a transfer is taking place. Typically, these bits are only modified in Slave mode when the SPE bit is cleared, when the device is not selected, or in Master mode when no transfer is in progress.

NOTE:

When writing the SPCR0 or the SPCR1, it is necessary to first write the register with the SPE bit cleared. Then a second write is performed to the SPCR with the same value except that the SPE bit is set. The BFSET instruction can be used in place of this second write to set the SPE bit. This is true for all bits in the SPCR except the SPIE bit, which can be modified with a BFSET or BFCLR instruction at any time, even when the SPE bit is set.

The SPCR0 and SPCR1 are reset to $0000 on hardware reset. The bits of the SPCR0 and SPCR1 are described in Section 7.2.1.1, “Reserved Bits—Bits 15–9,” through Section 7.2.1.8, “Clock Phase (CPH)—Bit 2.”

7.2.1.1 Reserved Bits—Bits 15–9

Bits 15–9 of the SPCR0 and SPCR1 are reserved and are read as zero during read operations. These bits should be written with zero to ensure future compatibility.

7.2.1.2 SPI Clock Rate Select (SPR[2:0])—Bits 8, 1–0

The SPI clock rate select (SPR[2:0]) bits are used to program the divider of the Phi clock to generate a serial bit clock for the SPI peripheral when the SPI is configured as a master device. For an SPI configured as a slave, the serial clock is input from the master and these bits have no meaning. The SPR bits are cleared on hardware reset.

Table 7-1. SPR Divider Programming

SPR[2:0]	Divider
000	1
001	2
010	4
011	8
100	16
101	32
110	64
111	128

NOTE:

The maximum frequency of the serial bit clock is limited to 10 MHz. This means that for a 70 MHz DSP56824, the SPR bits must be programmed so that the clocking frequency of the serial bit clock is less than or equal to 10 MHz. Thus, setting the SPR[2:0] bits to 000 (divide by 1) can be used only when the frequency of the Phi clock is less than or equal to 10 MHz. Likewise, SPR[2:0] = 001 can be used only when the Phi clock is less than or equal to 20 MHz, and SPR[2:0] = 010 requires a Phi clock frequency of less than or equal to 40 MHz. All other combinations (divide by 8 through divide by 128) can be used with any Phi clock frequency up to the maximum frequency of 70 MHz.

7.2.1.3 SPI Interrupt Enable (SPIE)—Bit 7

The SPI interrupt enable (SPIE) control bit is used to enable interrupts from the SPI port. When interrupts are disabled (the SPIE bit is cleared), any pending interrupt is cleared, and polling is used to sense the SPI interrupt complete flag (SPIF) and mode fault (MDF) bits. When interrupts are enabled (the SPIE bit is set), an SPI interrupt is requested if either the SPIF or the MDF bit is set. The SPIE bit is cleared on hardware reset.

As with all on-chip peripheral interrupts for the DSP56824, the status register (SR—bits I[1:0] = 01) must first be set to enable maskable interrupts (interrupts of level IPL 0). Next, the IPR must also be set to enable the interrupt. Table 7-2 lists the appropriate bits to set in the interrupt priority register (IPR) and the corresponding interrupt vector.

Table 7-2. SPI Interrupt

SPI Port (Register)	Bit in IPR	Interrupt Vector	Interrupt Priority
SPI0 (SPCR0)	Bit 13 (CH2)	$002A	0
SPI1 (SPCR1)	Bit 12 (CH3)	$0028	0

Table 3-6 on page 3-16 lists the interrupt priority order for the DSP56824. Finally, SPI interrupts are configured within the SPI itself, using the SPIE bit.

7.2.1.4 SPI Enable (SPE)—Bit 6

The SPI enable (SPE) control bit is used to enable the SPI port functionality. Clearing the SPE bit disables the SPI peripheral and the shuts off the Phi clock signal provided to the SPI port to reduce power consumption. The SPE bit is cleared on hardware reset.

7.2.1.5 Wired-OR Mode (WOM)—Bit 5

The Wired-OR mode (WOM) control bit is used to select the nature of the SPI pins. When enabled (the WOM bit is set), the SPI pins in Port C are configured as open-drain drivers with the P-channel pull-ups disabled. When disabled (the WOM bit is cleared), the SPI pins are configured as push-pull drivers. The WOM bit is cleared on hardware reset.

7.2.1.6 Master Mode Select (MST)—Bit 4

The Master mode select (MST) control bit is used to select Master or Slave mode. The MST bit is cleared on hardware reset. Table 7-3 shows how this mode is set.

Table 7-3. SPI Mode Programming

MST	SPI Mode
0	Slave
1	Master

7.2.1.7 Clock Polarity (CPL)—Bit 3

The clock polarity (CPL) control bit is used to define the polarity of the serial bit clock. When data is not being transferred and this bit is cleared, the SCK pin of the master device idles as a logic low. When the CPL bit is set, the SCK pin idles as a logic high. The CPL bit is cleared on hardware reset.

7.2.1.8 Clock Phase (CPH)—Bit 2

The clock phase (CPH) control bit is used in conjunction with the CPL bit to control the clock-data relationship between the master and slave. This bit selects one of two different clocking protocols. The CPH bit is cleared on hardware reset.

7.2.2 SPI Status Register (SPSR0 and SPSR1)

The SPI status registers (SPSR0 and SPSR1) are 16-bit read-only registers used to indicate the status of the SPI0 and SPI1 peripherals, respectively. The bits within these registers are arranged and interpreted identically. The only differences are that the two registers have different addresses and represent the status of different SPIs. The value of one register is not affected by the value of the other register. The bits of these registers are defined in Section 7.2.2.1, “Reserved Bits—Bits 15–8,” through Section 7.2.2.6, “Reserved Bits—Bits 3–0.”

7.2.2.1 Reserved Bits—Bits 15–8

Bits 15–8 of SPSR0 and SPSR1 are reserved and are read as zero during read operations. These bits should be written with zero to ensure future compatibility.

7.2.2.2 SPI Interrupt Complete Flag (SPIF)—Bit 7

The SPI interrupt complete flag (SPIF) bit is set upon completion of data transfer between the processor and the external device. If the SPIF bit goes high and the SPIE bit is set, an interrupt is generated. To clear the SPIF bit, read the SPSR with the SPIF bit set, then access the SPDR. Unless the SPSR is read first (with the SPIF bit set), attempts to write to the SPDR are not permitted. The SPIF bit is cleared on hardware reset.

7.2.2.3 Write Collision (WCOL)—Bit 6

The write collision (WCOL) flag bit is set when a write collision condition is detected. This bit is cleared by reading the SPSR (with MDF set) followed by an access of the SPDR. The WCOL flag is cleared on hardware reset.

7.2.2.4 Reserved Bit—Bit 5

Bit 5 of SPSR0 and SPSR1 is reserved and is read as zero during read operations. This bit should be written with zero to ensure future compatibility.

7.2.2.5 Mode Fault (MDF)—Bit 4

The mode fault (MDF) flag is set when a mode fault has occurred. This bit is cleared by reading the SPSR (with MDF set) followed by a write to the SPCR. The MDF flag is cleared on hardware reset.

7.2.2.6 Reserved Bits—Bits 3–0

Bits 3–0 of SPSR0 and SPSR1 are reserved and are read as zero during read operations. These bits should be written with zero to ensure future compatibility.

7.2.3 SPI Data Registers (SPDR0 and SPDR1)

The SPI data registers (SPDR0 and SPDR1) are 16-bit read/write registers used when the SPI devices are transmitting or receiving data on the serial bus. Only a write to one of these registers initiates the transmission or reception of a byte, and this only occurs on the master device. At the completion of transferring a byte of data, the SPIF status bit (in the corresponding SPSR) is set in both the master and slave devices.

A read of an SPDR is actually a read of a buffer. To prevent an overrun and the loss of the byte that caused the overrun, the first SPIF must be cleared by the time a second transfer of data from the shift register to the read buffer is initiated. This register is double-buffered for input and single-buffered for output.

7.3 SPI Data and Control Pins

During an SPI transfer, data is simultaneously transmitted (shifted out serially) and received (shifted in serially). An SCK line synchronizes shifting and sampling of the information on the two serial data lines. An $\overline{SS}$ line allows individual selection of a slave SPI device. Slave devices that are not selected do not interfere with SPI bus activities. On a master SPI device, the select line can optionally be used to indicate a multiple-master bus contention.

Each SPI device has four dedicated I/O pins. The four pins on SPI0 are as follows:

- **MISO0/PC0 (SPI0 master in/slave out)**—The MISO0 pin carries one of two unidirectional serial data signals. It is an input to a master device and an output from a slave device. If a slave device is not selected, the MISO0 line of the slave device is placed in the high-impedance state. MISO0 can be programmed as a GPIO pin called PC0 when the SPI MISO0 function is not being used.

- **MOSI0/PC1 (SPI0 master out/slave in)**—The MOSI0 pin carries the second of the two unidirectional serial data signals. It is an output from a master device and an input to a slave device. The master device places data on the MOSI0 line a half-cycle before the clock edge that the slave device uses to latch the data. MOSI0 can be programmed as a general-purpose input/output (GPIO) pin called PC1 when the SPI MOSI0 function is not being used.
- **SCK0/PC2 (SPI0 serial clock)**—SCK0, an input to a slave device, is generated by the master device and synchronizes data movement in and out of the device through the MOSI0 and MISO0 lines. Master and slave devices are capable of exchanging a byte of information during a sequence of eight clock cycles. Slave devices ignore the SCK signal unless the slave select pin is active low.

  Four possible timing relationships can be chosen by using the CPL and CPH control bits in the SPCR. Both master and slave devices must operate with the same timing. The SPI clock rate select bits, SPR[2:0], in the SPCR of the master device, select the clock rate. In a slave device, the SPR[2:0] bits have no effect on the operation of the SPI. The SCK0 pin can be programmed as a GPIO pin (PC2) when the SPI SCK0 function is not being used.
- **$\overline{\text{SS0}}$/PC3 (SPI0 slave select)**—The slave select ($\overline{\text{SS}}$) input of a slave device must be externally asserted before a master device can exchange data with the slave device. $\overline{\text{SS}}$ must be low before data transactions and must stay low for the duration of the transaction. The $\overline{\text{SS}}$ line of the master must be held high. If it goes low, a mode fault error flag (the MDF bit) is set in the SPSR. To disable the mode fault circuit, program the $\overline{\text{SS}}$ pin as a GPIO pin in the PCC register. This inhibits the mode fault flag, while the other three lines are dedicated to the SPI peripheral.

  The state of the master and slave CPH bits (in the SPCR) affects the operation of $\overline{\text{SS}}$. The CPH bit settings should be the same for both master and slave. When the CPH bit is cleared, the shift clock is the OR of $\overline{\text{SS}}$ with SCK. In this clock phase mode, $\overline{\text{SS}}$ must go high between successive bytes in an SPI message. When the CPH bit is set, $\overline{\text{SS}}$ can be left low between successive SPI bytes. In cases where there is only one SPI slave MCU, its $\overline{\text{SS}}$ line can be tied to V_{SS} as long as only CPH = 1 clock mode is used. The $\overline{\text{SS0}}$ pin can be programmed as a GPIO pin (PC3) when the SPI $\overline{\text{SS0}}$ function is not being used.

The four pins on SPI1 are as follows:

- **MISO1/PC4 (SPI1 master in/slave out)**—MISO1 is one of two unidirectional serial data signals. It is an input to a master device and an output from a slave device. If a slave device is not selected, the MISO0 line of the slave device is placed in the high-impedance state. MISO1 can be programmed as a GPIO pin called PC4 when the SPI MISO1 function is not being used.
- **MOSI1/PC5 (SPI1 master out/slave in)**—The MOSI1 line is the second of the two unidirectional serial data signals. It is an output from a master device and an input to a slave device. The master device places data on the MOSI1 line a half-cycle before the clock edge that the slave device uses to latch the data. MOSI1 can be programmed as a GPIO pin called PC5 when the SPI MOSI1 function is not being used.
- **SCK1/PC6 (SPI1 serial clock)**—SCK1, an input to a slave device, is generated by the master device and synchronizes data movement in and out of the device through the MOSI1 and MISO1 lines. Master and slave devices are capable of exchanging a byte of information during a sequence of eight clock cycles. Slave devices ignore the SCK signal unless the $\overline{\text{SS}}$ pin is active low.

  Four possible timing relationships can be chosen by using the CPL and CPH control bits in the SPCR. Both master and slave devices must operate with the same timing. The SPI clock rate select bits (SPR[2:0]), in the SPCR of the master device, select the clock rate. In a slave device, the SPR[2:0] bits have no effect on the operation of the SPI. The SCK1 pin can be programmed as a GPIO pin (PC6) when the SPI SCK1 function is not being used.

- **$\overline{SS1}$/PC7 (SPI1 slave select)**—The $\overline{SS}$ input of a slave device must be externally asserted before a master device can exchange data with the slave device. $\overline{SS}$ must be low before data transactions and must stay low for the duration of the transaction. The $\overline{SS}$ line of the master must be held high. If it goes low, the mode fault flag (MDF) bit is set in the SPSR. To disable the mode fault circuit, program the $\overline{SS}$ pin as a GPIO pin in the PCC register. This inhibits the MDF bit, while the other three lines are dedicated to the SPI.

  The state of the master and slave CPH bits affects the operation of $\overline{SS}$. The CPH bit settings should be the same for both master and slave. When the CPH bit is cleared, the shift clock is the OR of $\overline{SS}$ with SCK. In this clock phase mode, $\overline{SS}$ must go high between successive bytes in an SPI message. When the CPH bit is set, $\overline{SS}$ can be left low between successive SPI bytes. In cases where there is only one SPI slave MCU, its $\overline{SS}$ line can be tied to V_{SS} as long as only CPH = 1 clock mode is used. $\overline{SS1}$ can be programmed as a GPIO (PC7) when the SPI $\overline{SS1}$ function is not being used.

SPI timing is described in the *DSP56824 Technical Data Sheet*. SPI output pins can be configured as open-drain drivers. The WOM bit in the SPCR is used to enable this option in each SPI. An external pull-up resistor is required on each Port C output pin while this option is selected. In multiple-master systems, this option provides extra protection against complementary metal-oxide semiconductor (CMOS) latchup, because even if more than one SPI device tries to simultaneously drive the same bus line, there can be no destructive contention.

In some multiple SPI systems where one device is always the master device, it may make sense to bypass the $\overline{SS}$ function and instead use this pin as a GPIO to drive the slave select of one of the slave SPI devices.

NOTE:

Bypassing $\overline{SS}$ disables the mode-fault detection capability.

7.4 SPI System Errors

Two system errors can be detected by the SPI system. The first type of error arises in a multiple-master system when more than one SPI device simultaneously tries to be a master. This error is called a mode-fault error. The second type of error, a write-collision error, indicates that an attempt was made to write data to an SPDR while a transfer was in progress.

7.4.1 SPI Mode-Fault Error

When the SPI system is configured as a master and the $\overline{SS}$ input line goes to active low, a mode-fault error occurs, usually because two devices attempted to act as master at the same time. Only an SPI master can experience a mode-fault error. In cases where more than one device is concurrently configured as a master, a chance of contention between two pin drivers exists. For push-pull CMOS drivers, this contention can cause permanent damage. The mode fault attempts to protect the device by disabling the drivers. When this error is detected, the following actions are taken immediately:

1. The SPI pins are reconfigured as all inputs.
2. The MST control bit is forced to zero to reconfigure the SPI as a slave.
3. The SPE control bit is forced to zero to disable the SPI system.
4. The MDF flag is set and an SPI interrupt is generated subject to masking by the SPIE bit, the appropriate bit in the IPR, and the I1 bit in the SR.

NOTE:

The SPI pins are reconfigured as inputs regardless of the values of the corresponding Port C data direction register (PCDDR) bits. This condition is removed once the SPE bit is set again, and the pins resume their normal SPI functionality.

After software has corrected the problems that led to the mode fault, the MDF bit is cleared and the system returns to normal operation. The MDF bit is automatically cleared by reading the SPSR while the MDF bit is set, followed by a write to the SPCR.

Other precautions may need to be taken to prevent driver damage. If two devices are made masters at the same time, mode fault does not help protect either one unless one of them selects the other as slave. The amount of damage possible depends on the length of time both devices attempt to act as master.

Because the $\overline{SS}$ pin is used to detect mode fault, it should not be configured as a GPIO pin on an SPI system where more than one SPI is capable of functioning as an SPI master. Maintaining the $\overline{SS}$ functionality can help prevent driver damage if mode fault occurs.

7.4.2 SPI Write-Collision Error

A write-collision error occurs if the SPDR is written while a transfer is in progress. Because the SPDR is not double-buffered in the transmit direction, writes to the SPDR cause data to be written directly into the SPI shift register. Because this write corrupts any transfer in progress, a write-collision error is generated. The transfer continues undisturbed, and the write data that caused the error is not written to the shifter. A write collision is normally a slave error because a slave has no control over when a master initiates a transfer. A master knows when a transfer is in progress, so there is no reason for a master to generate a write-collision error, although the SPI logic can detect write collisions in both master and slave devices.

The SPI configuration determines the characteristics of a transfer in progress. For a master, a transfer begins when data is written to the SPDR and ends when the SPIF bit is set. For a slave with the CPH bit cleared, a transfer starts when the $\overline{SS}$ pin goes low and ends when the $\overline{SS}$ pin returns high. In this case, the SPIF bit is set at the middle of the eighth SCK cycle (when data is transferred from the shifter to the parallel data register), but the transfer is still in progress until $\overline{SS}$ goes high.

For a slave with the CPH bit set, transfer begins when the SCK line goes to its active level, which is the edge at the beginning of the first SCK cycle. In this case, the transfer ends when the SPIF bit is set.

7.4.3 SPI Overrun

No mechanism is provided in the SPI for detecting overrun. Overrun is the condition where a second sample has been shifted in and transferred to the receive data buffer before a first sample has been read from the receive data buffer.

Overrun is avoided by reading the SPDR from the previous transfer before the start of the eighth cycle of the current transfer. Any data from a previous transfer becomes invalid at the start of the eighth cycle.

7.5 Configuring Port C for SPI Functionality

The PCC register is used to individually configure each pin as either an SPI pin or a GPIO pin. When the SPI is used in Slave mode, all four SPI pins must be programmed as SPI pins in the PCC register. When in Master mode, the SCK, MISO, and MOSI pins are configured as SPI pins, and the $\overline{SS}$ pin is optionally configured as an SPI pin if mode-fault detection is desired. Otherwise, $\overline{SS}$ can be configured as a GPIO pin in the PCC register. This is summarized in Table 7-4.

Table 7-4. PCC Register Programming for the $\overline{SS}$ Pin

Mode	CC Bit[1]	Functionality of the $\overline{SS}$ Pin
Slave	0	Not allowed
Slave	1	Used as $\overline{SS}$ pin
Master	0	Used as GPIO pin
Master	1	Used for mode-fault detection

1. For SPI0, use PCC[3]. For SPI1, use PCC[7].

When the PCC register bit is set for an SPI pin, it is not necessary to program the corresponding PCDDR bit. The SPI peripheral ensures the correct direction of this pin. Programming the PCDDR is necessary only when a pin is programmed as a GPIO pin.

In applications requiring minimum power consumption, the SPI module can be disabled by clearing the SPE bit in SPCR. This also shuts off the Phi clock signal as it enters the block.

Stop mode automatically disables the SPI peripheral and any external clocks for devices configured in Slave mode. The internal Phi clock is stopped in stop mode. If a device is in the middle of a transfer when it enters stop mode, all internal state machines are reset so that, upon exiting stop mode, the device waits for a new transfer to be initiated.

In wait mode, the SPI peripheral continues operation and can complete transfers in progress. If the SPI interrupt is enabled in the SPCR, IPR, and SR, the SPI peripheral can generate an interrupt request in wait mode that brings the DSP56824 out of wait mode.

7.6 Programming Examples

Example 7-1 through Example 7-3 on page 7-17 show how to configure one SPI port as a master, how to configure one as a slave, and how to send data from master to slave.

7.6.1 Configuring an SPI Port as Master

Example 7-1 shows how to configure an SPI port as a master.

Example 7-1. Configuring an SPI Port as Master

```
;****************************************
;* SPI master                           *
;* for serial peripheral interface (SPI)*
;* of DSP56824 chip                     *
;****************************************
;
START               EQU     $0040       ; Start of program
BCR                 EQU     $FFF9       ; Bus Control Register
IPR                 EQU     $FFFB       ; Interrupt Priority Register
PCC                 EQU     $FFED       ; Port C Control Register
PCDDR               EQU     $FFEE       ; Port C Data Direction Register
PCR0                EQU     $FFF2       ; PLL Control Register 0
PCR1                EQU     $FFF3       ; PLL Control Register 1
SPCR0               EQU     $FFE2       ; SPI0 Control Register
SPDR0               EQU     $FFE0       ; SPI0 Data Register
SPSR0               EQU     $FFE1       ; SPI0 Status Register
;**************
;* Vector setup*
;**************
;
      ORG           P:$0000             ; Cold Boot
      JMP           START               ; also Hardware RESET vector (Mode 0, 1, 3)
      ORG           P:$E000             ; Warm Boot
      JMP           START               ; Hardware RESET vector (Mode 2)
;     ORG           P:$0028             ;
;     JSR           [unused]            ; SPI1 Serial System vector
      ORG           P:START             ; Start of program
;*****************
;* General setup *
;*****************
;
      MOVEP          $0000,X:BCR        ; External Program memory has 0 wait states.
                                        ; External data memory has 0 wait states.
                                        ; Port A pins are tri-stated when no
                                        ; external access occurs.
```

Example 7-1. Configuring an SPI Port as Master (Continued)

```
;*********************************************************
;* Phi Clock included for serial bit clock of SPI Master *
;*********************************************************
;
       MOVEP        #$0180,X:PCR1       ;Configure:
                                        ;(PLLE) PLL disabled (bypassed)
                                        ; -- Oscillator supplies Phi clock.
                                        ;(PLLD) PLL Power Down disabled (PLL active).
                                        ; -- PLL block active for PLL to attain lock.
                                        ;(LPST) Low Power Stop disabled.
                                        ;(PS[2:0]) Prescaler Clock disabled.
                                        ;(CS[1:0]) Clockout pin sends Phi clock.
       MOVEP        #$0260,X:PCR0       ; Set Feedback Divider to 1/20
                                        ; ...
                                        ; insert delay here: wait for PLL lock
                                        ; as specified in data sheet
                                        ; ...
       BFSET        #$4000,X:PCR1       ; Enable PLL for Phi clock.
;********************
;* SPI Master setup *
;********************
;
       BFCLR        #$0040,X:SPCR0      ; (SPE) SPI disabled
       MOVEP        #$0116,X:SPCR0      ; Configure:
                                        ; (SPR[2:0]) SPI Clock Rate Select at /64.
                                        ; (SPIE) SPI Interrupt disabled.
                                        ; (WOM) Wired-OR mode disabled:
                                        ; - push-pull drivers.
                                        ; (MST) Master mode selected.
                                        ; (CPL) serial Clock Polarity:
                                        ; - SCK pin idles as logic low.
                                        ; (CPH) Clock Phase protocol:
                                        ; - ~SS line can be tied low if only 1 slave.
       BFSET        #$0007,X:PCC        ; Configure:
                                        ; MISO0, MOSI0, SCK0 for SPI master,
                                        ; ~SS0 as PC3 for GPIO.
                                        ; Other pins remain as previously set.
                                        ; (reset default is GPIO).
       BFSET        #$0008,X:PCDDR      ; Configure GPIO pin PC3 as output
       BFCLR        #$2000,X:IPR        ; Disable SPI0 interrupts
                                        ; (unnecessary but mentioned for
completeness).
       BFSET        #$0040,X:SPCR0      ; (SPE) SPI Enabled.
;****************
;* Main routine *
;****************
;
                                        ; ...
TEST                                    ; Test Loop
                                        ; ...
       BRA    TEST
```

7.6.2 Configuring an SPI Port as Slave

Example 7-2 shows how to configure an SPI port as a slave.

Example 7-2. Configuring an SPI Port as Slave

```
;****************************************
;* SPI slave                            *
;* for serial peripheral interface (SPI) *
;* of DSP56824 chip                     *
;****************************************
;
START               EQU     $0040       ; Start of program
BCR                 EQU     $FFF9       ; Bus Control Register
IPR                 EQU     $FFFB       ; Interrupt Priority Register
PCC                 EQU     $FFED       ; Port C Control Register
;PCDDR              EQU     $FFEE       ; Port C Data Direction Register [unused]
SPCR1               EQU     $FFE6       ; SPI1 Control Register
SPDR1               EQU     $FFE4       ; SPI1 Data Register
SPSR1               EQU     $FFE5       ; SPI1 Status Register
;****************
;* Vector setup *
;****************
;
      ORG           P:$0000             ; Cold Boot
      JMP           START               ; also Hardware RESET vector (Mode 0, 1, 3)
      ORG           P:$E000             ; Warm Boot
      JMP           START               ; Hardware RESET vector (Mode 2)
;     ORG           P:$002A             ;
;     JSR           [unused]            ; SPI0 Serial System vector
      ORG           P:START             ; Start of program
;*****************
;* General setup *
;*****************
;
      MOVEP         #$0000,X:BCR        ; External Program memory has 0 wait states.
                                        ; External data memory has 0 wait states.
                                        ; Port A pins are tri-stated when no
                                        ; external access occurs.
;******************
;* SPI Slave setup *
;******************
;
      BFCLR         #$0040,X:SPCR1      ; (SPE) SPI disabled.
      MOVEP         #$0004,X:SPCR1      ; Configure:
                                        ; (SPR) SPI Clock Rate Select at /1
                                        ; [irrelevant; mentioned for completeness]
                                        ; (SPIE) SPI Interrupt disabled.
                                        ; (WOM) Wired-OR Mode disabled:
                                        ; - push-pull drivers
                                        ; (MST) Master mode off (Slave mode selected)
                                        ; (CPL) serial Clock Polarity:
                                        ; - SCK pin idles as logic low
                                        ; (CPH) Clock Phase protocol:
                                        ; - ~SS line can be tied low if only 1 slave.
```

Example 7-2. Configuring an SPI Port as Slave (Continued)

```
       BFSET         #$00F0,X:PCC       ; Configure:
                                        ; MISO1, MOSI1, SCK1, ~SS1 for SPI slave
                                        ; (~SS0 required for slave mode)
                                        ; other pins remain as previously set
                                        ; (reset default is GPIO)
;      [PCDDR unused]            ; SPI ensures correct direction of used SPI pins
       BFCLR         #$1000,X:IPR       ; Disable SPI1 interrupts
                                        ; (unnecessary; mentioned for completeness)
       BFSET         #$0040,X:SPCR1     ; (SPE) SPI Enabled.
;****************
;* Main routine *
;****************
                                        ; ...
TEST                                    ; Test Loop
                                        ; ...
       BRA    TEST
```

7.6.3 Sending Data from Master to Slave

Example 7-3 shows how to send data from an SPI port configured as a master to an SPI port configured as a slave.

Example 7-3. Sending Data from Master to Slave

```
;****************************************
;* Transfer from SPI master to SPI slave *
;* for serial peripheral interface (SPI) *
;* of DSP56824 chip                      *
;****************************************
;* SPI0: Master                          *
;* SPI1: Slave                           *
;****************************************
START               EQU    $0040        ; Start of program
BCR                 EQU    $FFF9        ; Bus Control Register
IPR                 EQU    $FFFB        ; Interrupt Priority Register
PCC                 EQU    $FFED        ; Port C Control Register
PCDDR               EQU    $FFEE        ; Port C Data Direction Register
PCR0                EQU    $FFF2        ; PLL Control Register 0
PCR1                EQU    $FFF3        ; PLL Control Register 1
SPCR0               EQU    $FFE2        ; SPI0 Control Register
SPCR1               EQU    $FFE6        ; SPI1 Control Register
SPDR0               EQU    $FFE0        ; SPI0 Data Register
SPDR1               EQU    $FFE4        ; SPI1 Data Register
SPSR0               EQU    $FFE1        ; SPI0 Status Register
SPSR1               EQU    $FFE5        ; SPI1 Status Register
;****************
;* Vector setup *
;****************
       ORG          P:$0000            ; Cold Boot
       JMP          START              ; also Hardware RESET vector (Mode 0, 1, 3)
       ORG          P:$E000            ; Warm Boot
       JMP          START              ; Hardware RESET vector (Mode 2)
;      ORG          P:$0028            ;
;      JSR                             ; SPI1 Serial System vector [unused]
;      ORG          P:$002A            ;
;      JSR                             ; SPI0 Serial System vector [unused]
```

Example 7-3. Sending Data from Master to Slave (Continued)

```
        ORG          P:START             ; Start of program
;*****************
;* General setup *
;*****************
        MOVEP        #$0000,X:BCR        ; External Program memory has 0 wait states.
                                         ; External data memory has 0 wait states.
                                         ; Port A pins are tri-stated when no external
                                         ; access occurs.
;*********************************************************
;* Phi Clock included for serial bit clock of SPI Master *
;*********************************************************
        MOVEP         #$0180,X:PCR1      ;Configure:
                                         ;(PLLE) PLL disabled (bypassed)
                                         ; - Oscillator supplies Phi Clock.
                                         ;(PLLD) PLL Power Down disabled (PLL active)
                                         ; - PLL block active for PLL to attain lock.
                                         ;(LPST) Low Power Stop disabled.
                                         ;(PS[2:0]) Prescaler Clock disabled.
                                         ;(CS[1:0]) Clockout pin sends Phi Clock.
        MOVEP         #$0260,X:PCR0      ; Set Feedback Divider to 1/20.
                                         ; ...
                                         ; insert delay here: wait for PLL lock
                                         ; as specified in data sheet.
                                         ; ...
        BFSET          #$4000,X:PCR1     ; Enable PLL for Phi Clock.
;********************
;* SPI Master setup *
;********************
        BFCLR         #$0040,X:SPCR0     ; (SPE) SPI disabled.
        MOVEP         #$0116,X:SPCR0     ;Configure:
                                         ;(SPR[2:0]) SPI Clock Rate Select at /64.
                                         ;(SPIE) SPI Interrupt disabled.
                                         ;(WOM) Wired-OR Mode disabled:
                                         ; - push-pull drivers.
                                         ;(MST) Master mode selected.
                                         ;(CPL) serial Clock Polarity:
                                         ; - SCK pin idles as logic low.
                                         ;(CPH) Clock Phase protocol:
                                         ; - ~SS line is tied low if only one slave.
        BFSET         #$0007,X:PCC       ;Configure:
                                         ;MISO0, MOSI0, SCK0 for SPI master,
                                         ;~SS0 as PC3 for GPIO.
                                         ;Other pins remain as previously set
                                         ; (reset default is GPIO).
        BFSET         #$0008,X:PCDDR     ;Configure GPIO pin PC3 as output.
        BFCLR         #$2000,X:IPR       ;Disable SPI0 interrupts
                                         ; (unnecessary but here for completeness).
        BFSET         #$0040,X:SPCR0     ;(SPE) SPI Enabled.
```

Example 7-3. Sending Data from Master to Slave (Continued)

```
;*******************
;* SPI Slave setup *
;*******************
;
       BFCLR        #$0040,X:SPCR1      ;(SPE) SPI disabled.
       MOVEP        #$0004,X:SPCR1      ;Configure:
                                        ;(SPR) SPI Clock Rate Select at /1
                                        ; - [irrelevant but here for completeness]
                                        ;(SPIE) SPI Interrupt disabled.
                                        ;(WOM) Wired-OR Mode disabled:
                                        ; - push-pull drivers.
                                        ;(MST) Master mode off (slave mode selected).
                                        ;(CPL) serial Clock Polarity:
                                        ; - SCK pin idles as logic low
                                        ;(CPH) Clock Phase protocol:
                                        ; - ~SS line tied low if only one slave
       BFSET        #$00F0,X:PCC        ;Configure:
                                        ;MISO1, MOSI1, SCK1, ~SS1 for SPI slave
                                        ;(~SS0 required for slave mode).
                                        ;Other pins remain as previously set
                                        ; (reset default is GPIO).
;      [PCDDR unused]                   ;SPI sets correct direction of used SPI pins.
       BFCLR        #$1000,X:IPR        ;Disable SPI1 interrupts.
                                        ; (unnecessary but here for completeness).
       BFSET        #$0040,X:SPCR1      ;(SPE) SPI Enabled.
;****************
;* Main routine *
;****************
;
;+-------------------------------------------------------------+
;| This example serves to illustrate the mechanics of transfer.   |
;| Transfers can also be done with interrupts instead of polling. |
;| Data is transferred one byte at a time, lower byte first.  |
;| SPI data lines are connected via the Master Out/Slave In pins. |
;+-------------------------------------------------------------+
       MOVE         #$0000,X0           ;Clear X0 to initialize output pattern

XLOOP  ; Transfer Loop
       MOVE         X0,A1               ;Load output pattern into A1.
       DO           #2,XFER             ;Send two bytes (lower byte goes first).
       JSR          TXBYTE              ;Transmit byte out on MOSI0 line.
       JSR          RXBYTE              ;Receive byte in from MOSI1 line.
XFER
       CMP          X0,B0               ;Did the data come back all right?
       BNE          XLOOP               ;No, something is wrong with connection.
                                        ; -- Try again until successful.
       INC          X0                  ;Increment X0 for next output pattern.
       BRA          XLOOP
       TXBYTE                           ;Transmit Byte
       MOVEP        X:SPSR0,A0          ;Read SPSR0 to clear SPIF so SPI can write
                                        ; to SPDR0, otherwise write is inhibited.
       MOVEP        A1,X:SPDR0          ;Transmit lower byte of data from A1.
       REP          #8                  ;Shift upper byte of data for next transfer
       ASR          A                   ; (only needed during first pass of loop).
       RTSRXBYTE                              ;Receive Byte
       BFTSTH       $0080,X:SPSR1       ;Test for SPIF flag high in Slave
       BCC          RXBYTE              ;Wait until data ready to be received
       MOVEP        X:SPSR1,B1          ;Read SPSR1 to clear SPIF flag (for status)
       MOVEP        X:SPDR1,B1          ;Receive data into B1
       REP          #8                  ;
       ASR          B0                  ;Shift data into B0
       RTS
```

Freescale Semiconductor, Inc.
ARCHIVED BY FREESCALE SEMICONDUCTOR, INC. 2005

Chapter 8
Synchronous Serial Interface

This section presents the synchronous serial interface (SSI) and discusses the SSI's architecture, programming model, operating modes, and initialization.

The SSI is a full-duplex serial port that allows the DSP56824 to communicate with a variety of serial devices, including industry-standard codecs, other DSPs, microprocessors, and peripherals that implement the Motorola serial peripheral interface (SPI). It is typically used to transfer samples in a periodic manner. The SSI consists of independent transmitter and receiver sections with independent clock generation and frame synchronization.

The capabilities of the SSI include:

- Independent (asynchronous) or shared (synchronous) transmit and receive sections with separate or shared internal/external clocks and frame syncs
- Normal mode operation using frame sync
- Network mode operation allowing multiple devices to share the port with as many as 32 time slots
- Gated clock mode operation requiring no frame sync
- Programmable internal clock divider
- Programmable word length (8, 10, 12, or 16 bits)
- Program options for frame sync and clock generation
- SSI power-down feature
- Completely separate clock and frame sync selections for the receive and transmit sections

8.1 SSI Architecture

Figure 8-1 shows the synchronous serial interface (SSI) provided on Port C of the SSI.

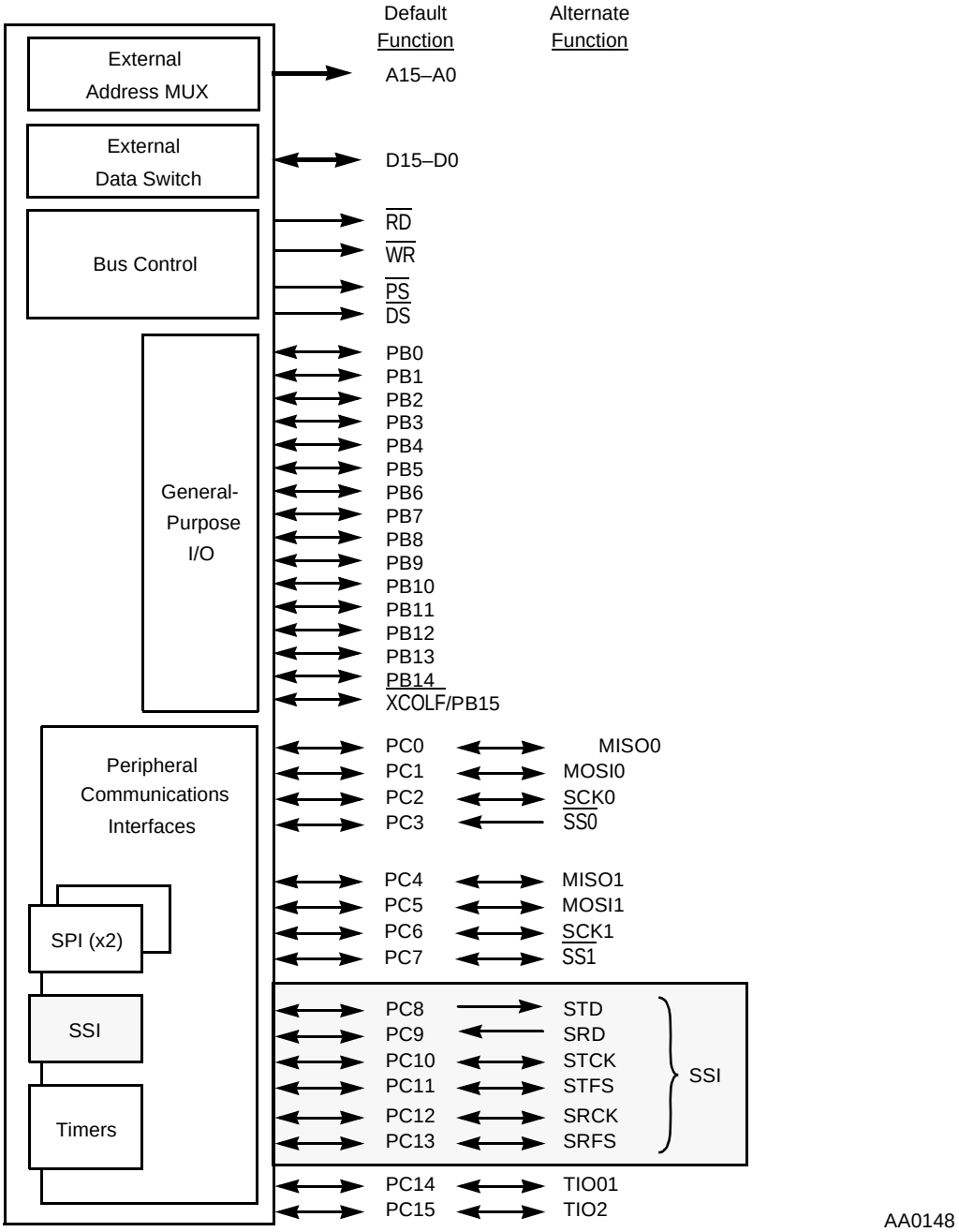


Figure 8-1. DSP56824 Input/Output Block Diagram

Figure 8-2 shows a block diagram of the SSI. It consists of three control registers to set up the port, one status register, separate transmit and receive circuits with buffer registers, and separate serial clock and frame sync generation for the transmit and receive sections.

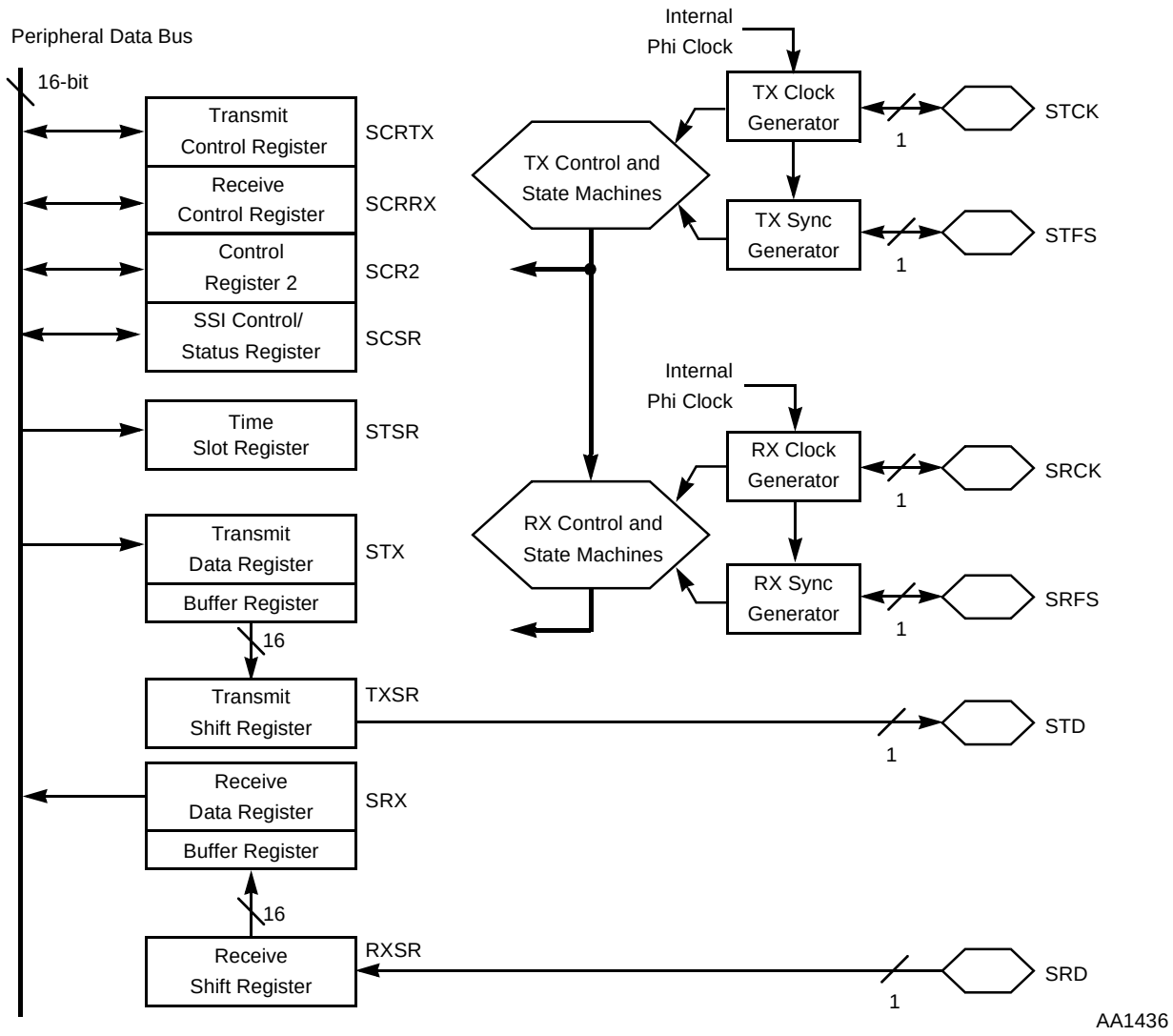


Figure 8-2. SSI Block Diagram

8.1.1 SSI Clocking

The SSI uses the following three clocks:

- Bit clock—used to serially clock the data bits in and out of the SSI port
- Word clock—used to count the number of data bits per word (8, 10, 12, or 16 bits)
- Frame clock—used to count the number of words in a frame

The bit clock, used to serially clock the data, is visible on the serial transmit clock (STCK) and serial receive clock (SRCK) pins. The word clock is an internal clock used to determine when transmission of an 8-, 10-, 12-, or 16-bit word has completed. The word clock in turn then clocks the frame clock, which counts the number of words in the frame. The frame clock can be viewed on the STFS and SRFS frame sync pins, because a frame sync is generated after the correct number of words in the frame have passed. The relationship between the clocks and the dividers is shown in Figure 8-3 on page 8-4. The bit clock can be received from an SSI clock pin or can be generated from the Phi clock through a divider, as shown in Figure 8-4 on page 8-5.

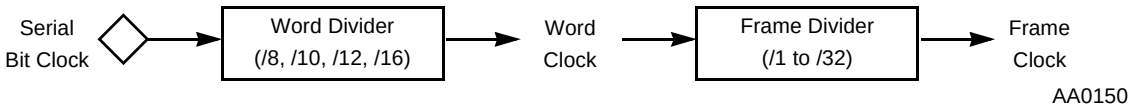


Figure 8-3. SSI Clocking

8.1.2 SSI Clock and Frame Sync Generation

Data clock and frame sync signals can be generated internally by the DSP56824 or can be obtained from external sources. If the signals are internally generated, the SSI clock generator is used to derive bit clock and frame sync signals from the Phi clock. The SSI clock generator consists of a selectable, fixed prescaler and a programmable prescaler for bit-rate clock generation. In gated clock mode, the data clock is valid only when data is being transmitted. Otherwise the clock pin is tri-stated. A programmable frame rate divider and a word length divider are used for frame rate sync signal generation.

Figure 8-4 shows a block diagram of the clock generator for the transmit section. The serial bit clock can be internal or external, depending on the transmit direction (TXD) bit in the SSI control register 2 (SCR2) control register. The receive section contains an equivalent clock generator circuit.

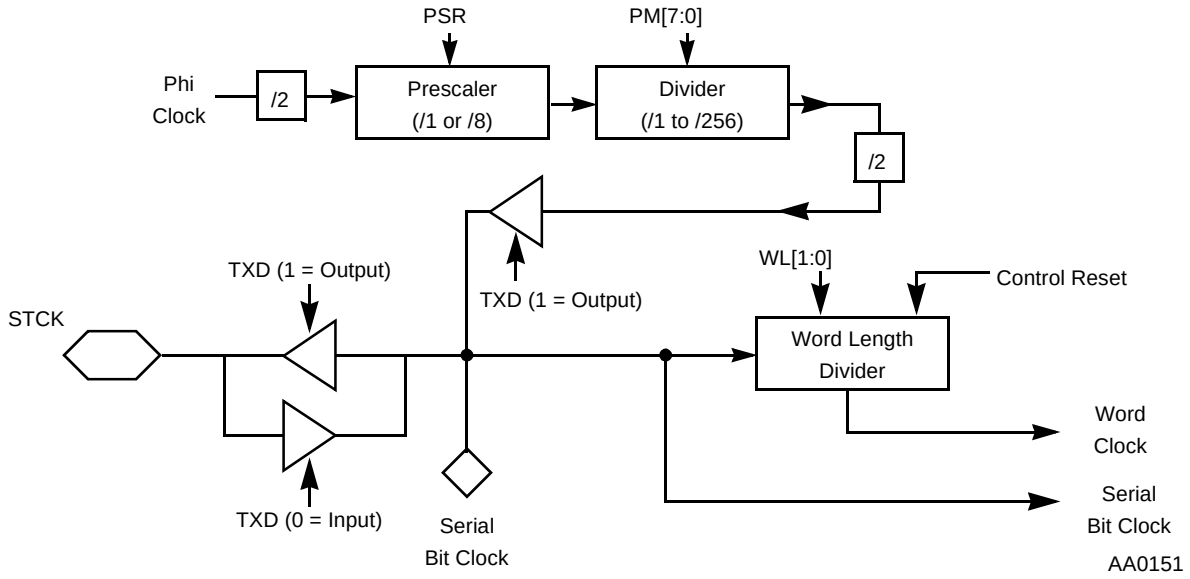


Figure 8-4. SSI Transmit Clock Generator Block Diagram

Figure 8-5 on page 8-5 shows the frame sync generator block for the transmit section. When internally generated, both receive and transmit frame sync are generated from the word clock and are defined by the frame rate divider (DC[4:0]) bits and the word length (WL[1:0]) bits of the SSI transmit control register (SCRTX). The receive section contains an equivalent circuit for the frame sync generator.

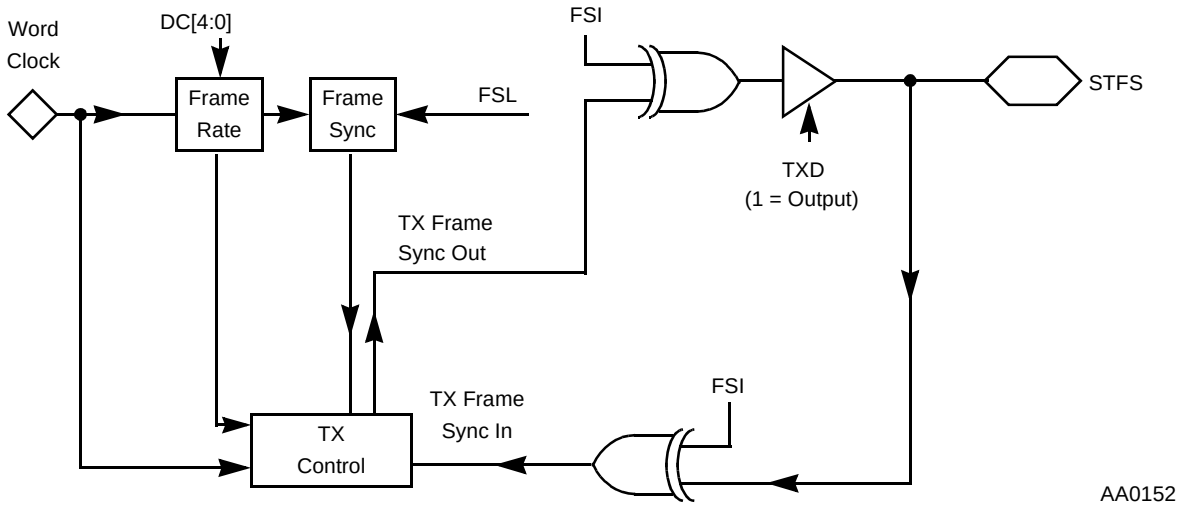


Figure 8-5. SSI Transmit Frame Sync Generator Block Diagram

8.2 SSI Programming Model

The registers associated with the SSI include the following:

- SSI transmit shift register (TXSR, not user accessible)
- SSI transmit data buffer register (not user accessible)
- SSI transmit data (STX, write-only) register
- SSI receive shift register (RXSR, not user accessible)
- SSI receive data buffer register (not user accessible)
- SSI receive data (SRX, read-only) register
- SSI transmit control register (SCRTX)
- SSI receive control register (SCRRX)
- SSI control register 2 (SCR2)
- SSI control/status register (SCSR, lower byte read-only)
- SSI time slot register (STSR, write-only)

The control registers associated with the SSI are shown in Figure 8-6 on page 8-7; Figure 8-7 on page 8-10 shows the programming information for SSI interrupts. Table 3-6, “Interrupt Priority Structure,” on page 3-16 lists the interrupt priority order for the DSP56824.

The TXSR, RXSR, receive data buffer register, and transmit data buffer register are not user accessible.

NOTE:

To use the SSI, the CC[13:8] bits in the Port C control (PCC) register must be correctly set. See Section 8.6, “Configuring Port C for SSI Functionality,” for more information.

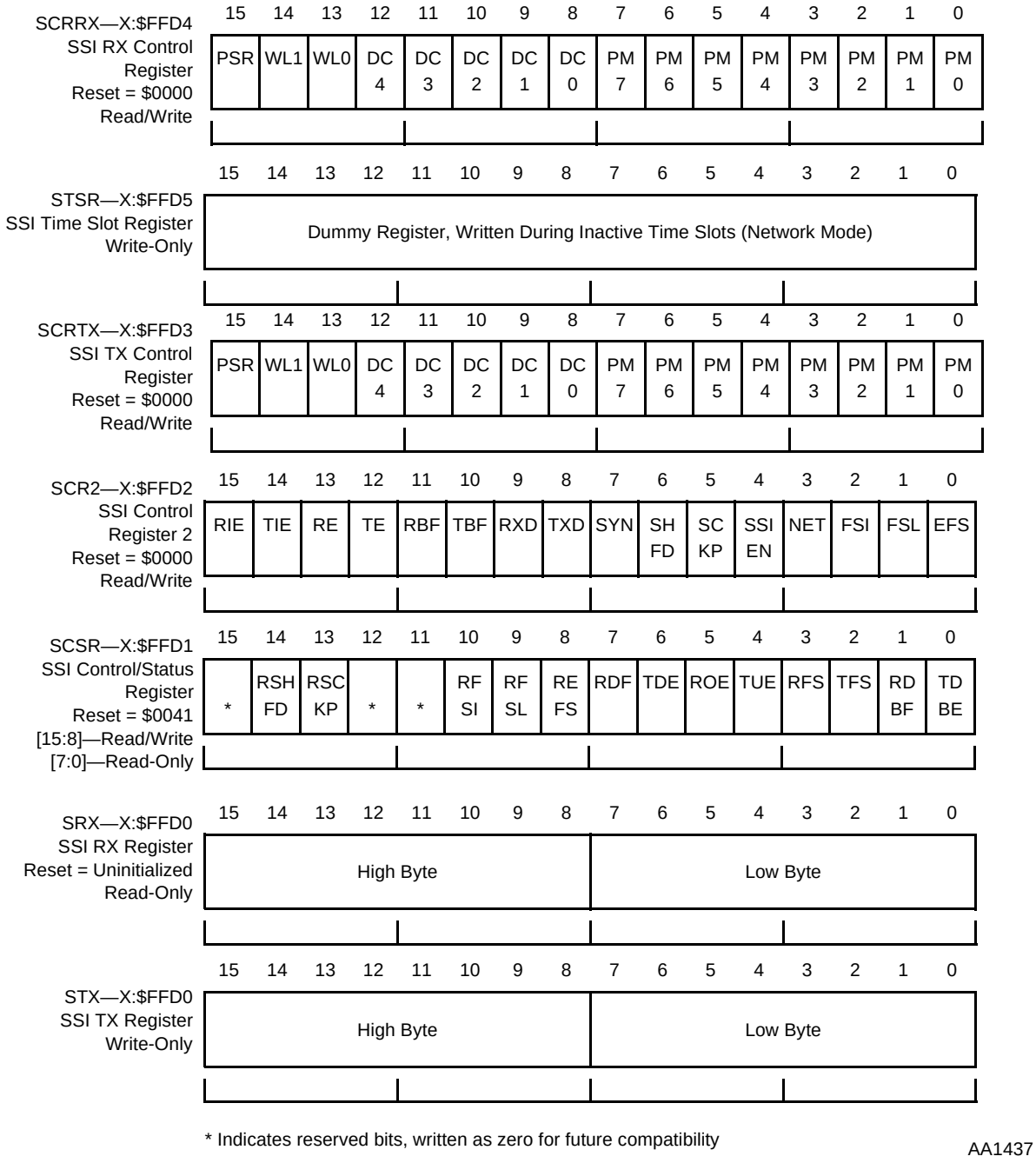


Figure 8-6. SSI Programming Model—Register Set

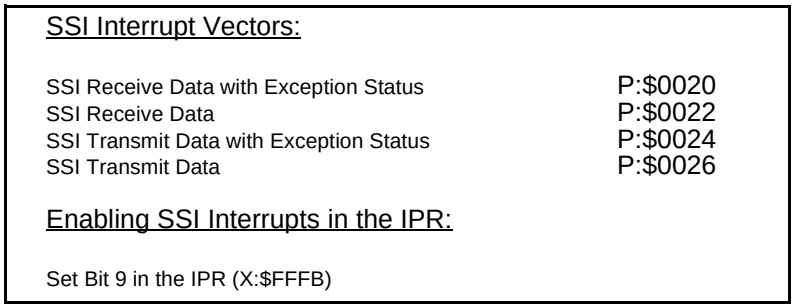


Figure 8-7. SSI Interrupt Vectors

8.2.1 SSI Transmit Shift Register (TXSR)

The SSI transmit shift register (TXSR) is a 16-bit shift register that contains the data being transmitted. When a continuous clock is used, data is shifted out to the serial transmit data (STD) pin by the selected (internal/external) bit clock when the associated (internal/external) frame sync is asserted. When a gated clock is used, data is shifted out to the STD pin by the selected (internal/external) gated clock. The word length control bits (WL[1:0]) in the SCRTX, described in Section 8.2.7, “SSI Transmit and Receive Control Registers,” determine the number of bits to be shifted out of the TXSR before it is considered empty and can be written to again. This word length can be 8, 10, 12, or 16 bits. The data to be transmitted occupies the most significant portion of the shift register. The unused portion of the register is ignored. Data is always shifted out of this register with the most significant bit (MSB) first when the SHFD bit of the SCR2 is cleared. If this bit is set, the least significant bit (LSB) is shifted out first.

8.2.2 SSI Transmit Data Buffer Register

The SSI transmit data buffer register is a 16-bit register used to buffer samples written to the STX register. It is written by the contents of the STX register whenever the transmit buffer feature is enabled. When it is enabled, the TXSR receives its values from this buffer register. If the transmit buffer feature is not enabled, this register is bypassed and the contents of the STX register is transferred into the TXSR.

When the transmit interrupt enable (TIE) bit in the SCR2 and TDE bit in the SCSR are set, the DSP56824 is interrupted whenever both the STX register and the SSI transmit data buffer register become empty.

8.2.3 SSI Transmit Data (STX) Register

The SSI transmit data (STX) register is a 16-bit, write-only register. Data to be transmitted is written into this register. If the transmit buffer is used, data is transferred from this register to the transmit data buffer register when it becomes empty. Otherwise, data written to this register is transferred to the TXSR when shifting of previous data is completed. The data written occupies the most significant portion of the STX register. The unused bits (least significant portion) of the STX register are ignored. The DSP56824 is interrupted whenever the STX register becomes empty (when both the STX register and SSI transmit data buffer register are empty, if buffering is enabled) if both the transmit data register empty (TDE) bit in the SCSR and the TIE bit in the SCR2 are set.

8.2.4 SSI Receive Shift Register (RXSR)

The SSI receive shift register (RXSR) is a 16-bit shift register that receives incoming data from the serial receive data (SRD) pin. When a continuous clock is used, data is shifted in by the selected (internal/external) bit clock when the associated (internal/external) frame sync is asserted. When a gated clock is used, data is shifted in by the selected (internal/external) gated clock. Data is assumed to be received MSB first if the SHFD bit of the SCR2 is cleared. If this bit is set, the data is received LSB first. Data is transferred to the SSI receive data (SRX) register or receive data buffer register (if the receive buffer is enabled) after 8, 10, 12, or 16 bits have been shifted in, depending on the WL[1:0] control bits.

8.2.5 SSI Receive Data Buffer Register

The SSI receive data buffer register is a 16-bit buffer register used to buffer samples received in the receive shift register. It is written by the receive shift register, whenever the receive buffer feature is enabled, by setting the receive buffer enable (RBF) bit in the SCR2. When enabled, the receive data register then receives its values from this buffer register. The DSP56824 is interrupted whenever both the SSI receive data register and SSI receive data buffer register become full, if the associated interrupt is enabled. If the receive buffer feature is not enabled, this register is bypassed and the receive shift register is automatically transferred into the SRX register.

8.2.6 SSI Receive Data (SRX) Register

The SSI receive data (SRX) register is a 16-bit read-only register. It accepts data contained in the receive data buffer register if receive buffering is enabled by setting the RBF bit in the SCR2. Otherwise, it accepts data from the receive shift register (RXSR) as it becomes full. The data read occupies the most significant portion of the SRX register. The unused bits (least significant portion) are read as zeros. The DSP56824 is interrupted whenever the SRX register becomes full (when both the SRX register and receive data buffer register are full if buffering is enabled), if the receive data full interrupt is enabled.

8.2.7 SSI Transmit and Receive Control Registers

The SSI transmit and receive control (SCRTX and SCRRX) registers are 16-bit read/write control registers used to direct the operation of the SSI. These registers control the SSI clock generator bit and frame sync rates, word length, and number of words per frame for the serial data. The SCRTX register is dedicated to the transmit section, and the SCRRX register is dedicated to the receive section—except in synchronous mode, in which the SCRTX register controls both the receive and transmit sections. DSP reset clears all SCRTX and SCRRX bits. SSI reset and STOP reset do not affect the SCRTX and SCRRX bits. The control bits are described in the following paragraphs.

Although the bit patterns of the SCRRX and SCTRX registers are the same, the contents of these two registers can be programmed differently. See Figure 8-6 on page 8-7 for the programming models of the SCRTX and SCRRX registers.

8.2.7.1 Prescaler Range (PSR)—Bit 15

The prescaler range (PSR) control bit controls a fixed divide-by-eight prescaler in series with the variable prescaler. It extends the range of the prescaler for those cases where a slower bit clock is desired. When the PSR bit is cleared, the fixed prescaler is bypassed. When the PSR bit is set, the fixed divide-by-eight prescaler is operational. This allows a 128 kHz master clock to be generated for Motorola MC1440x series codecs. The maximum internally generated bit clock frequency is $F_{osc}/4$, and the minimum internally generated bit clock frequency is $F_{osc}/(4 \times 8 \times 256)$.

8.2.7.2 Word Length Control (WL[1:0])—Bits 14–13

The word length control (WL[1:0]) bits are used to select the length of the data words being transferred by the SSI. Word lengths of 8, 10, 12, or 16 bits can be selected, as shown in Table 8-1.

Table 8-1. SSI Data Word Lengths

WL1	WL0	Number of Bits/Word
0	0	8
0	1	10
1	0	12
1	1	16

These bits control the word length divider shown in the SSI clock generator. The WL control bits also control the frame sync pulse length when the FSL bit is cleared.

8.2.7.3 Frame Rate Divider Control (DC[4:0])—Bits 12–8

The frame rate divider control (DC[4:0]) bits control the divide ratio for the programmable frame rate dividers. The divide ratio operates on the word clock. In normal mode, this ratio determines the word transfer rate. In network mode, this ratio sets the number of words per frame. The divide ratio ranges from 1 to 32 (DC[4:0] = 00000 to 11111) in normal mode and from 2 to 32 (DC[4:0] = 00001 to 11111) in network mode. A value of 00000 in DC[4:0] in network mode is illegal.

8.2.7.4 Prescale Modulus Select (PM[7:0])—Bits 7–0

The prescale modulus select (PM[7:0]) control bits specify the divide ratio of the prescale divider in the SSI clock generator. This prescaler is used only in internal clock mode to divide the internal clock of the core. A divide ratio from 1 to 256 (PM[7:0] = $00 to $FF) can be selected. The bit clock output is available at the clock SCK. The bit clock on the SSI can be calculated from the Phi clock value using the equation in Figure 8-8.

SCK = Phi Clock Frequency ÷ [4 × (7 × PSR + 1) × (PM + 1)]
where PM = PM[7:0]

SFS = (SCK) ÷ [(DC + 1) × WL]
where DC = DC[4:0] and WL = (8, 10, 12, or 16) AA1385

Figure 8-8. SSI Bit Clock Equation

For example, in 8-bit word normal mode with DC[4:0] set to 1 (00001), PM[7:0] set to 71 (0100 0111), the PSR bit cleared, and a 36.864 MHz Phi clock, a bit clock rate of 36.864 Mhz ÷ [1 × 4 × 72] = 128 kHz is generated. Since the 8-bit word rate is equal to two, the sampling rate (FS rate) would then be 128 kHz ÷ [2 × 8] = 16 kHz.

The bit clock output is also available internally for use as the bit clock to shift the transmit and receive shift registers. Careful choice of the crystal oscillator frequency and the prescaler modulus allows the telecommunication-industry-standard codec master clock frequencies of 2.048 MHz, 1.544 MHz, and 1.536 MHz to be generated. For example, a 24.576 MHz clock frequency can be used to generate the standard 2.048 MHz and 1.536 MHz rates, and a 24.704 MHz clock frequency can be used to generate the standard 1.544 MHz rate. Table 8-2 gives examples of PM[7:0] values that can be used to generate different bit clocks.

Table 8-2. SSI Bit Clock as a Function of Phi Clock and Prescale Modulus

Phi Clock (MHz)	Max Bit Clock (MHz)	PM[7:0] Values for Different SCK				
		2.048 MHz	1.544 MHz	1.536 MHz	128 kHz	64 kHz
16.384	4.096	1	—	—	31 ($1F)	63 ($3F)
18.432	4.608	—	—	2	35 ($23)	71 ($47)
20.480	5.12	—	—	—	39 ($27)	79 ($4F)
26.624	6.656	—	—	—	51 ($33)	103 ($67)
24.576	6.144	2	—	3	47 ($2F)	95 ($5F)
24.704	6.176	—	3	—	—	—
32.768	8.192	3	—	—	63 ($3F)	127 ($7F)
36.864	9.216	—	—	5	71 ($47)	143 ($8F)

8.2.8 SSI Control Register 2 (SCR2)

The SSI control register 2 (SCR2) is one of three 16-bit read/write control registers used to direct the operation of the SSI. The SSI reset is controlled by a bit in SCR2. SCR2 controls the direction of the bit clock and frame sync pins, STCK, SRCK, STFS, and SRFS. Interrupt enable bits for the receive and transmit sections are provided in this control register. SSI operating modes are also selected in this register. The DSP reset clears all SCR2 bits. However, SSI reset and STOP reset do not affect the SCR2 bits. The

SCR2 bits are described in Section 8.2.8.1, “Receive Interrupt Enable (RIE)—Bit 15,” through Section 8.2.8.16, “Early Frame Sync (EFS)—Bit 0.” See Figure 8-6 on page 8-7 for the programming model of the SCR2.

As with all on-chip peripheral interrupts for the DSP56824, the status register (SR—bits I[1:0 = 01) must first be set to enable maskable interrupts (interrupts of level IPL 0). Next, the CH6 bit (Bit 9) in the interrupt priority register (IPR) must be set to enable the interrupt. Finally, the interrupt can be enabled from within the SSI.

8.2.8.1 Receive Interrupt Enable (RIE)—Bit 15

The SSI receive interrupt enable (RIE) control bit allows interrupting the program controller. When the RIE bit is set, the program controller is interrupted when the SSI receive data register full (RDF) bit in the SCSR is set.

When the receive buffer is enabled, two values are available to be read. If it is not enabled, then one value can be read from the SRX register. If the RIE bit is cleared, this interrupt is disabled. However, the RDF bit still indicates the receive data register full condition. Reading the SRX register clears the RDF bit, thus clearing the pending interrupt.

Two receive data interrupts with separate interrupt vectors are available: receive data with exception status, and receive data without exception. Table 8-3 shows these vectors and the conditions under which these interrupts are generated.

Table 8-3. SSI Receive Data Interrupts

Interrupt	Vector	RIE	ROE	RDF
Receive data with exception status	$0020	1	1	1
Receive data (without exception)	$0022	1	0	1

8.2.8.2 Transmit Interrupt Enable (TIE)—Bit 14

The SSI transmit interrupt enable (TIE) control bit allows interrupting the program controller. When the TIE bit is set, the program controller is interrupted when the SSI transmit data register empty (TDE) flag in the SCSR is set.

When the transmit buffer is enabled, two values can be written to the SSI. If it is not enabled, then one value can be written to the STX register. When the TIE bit is cleared, this interrupt is disabled. However, the TDE bit always indicates the STX register empty condition, even when the transmitter is disabled by the transmit enable (TE) bit (in the SCR2). Writing data to the STX or STSR clears the TDE bit, thus clearing the interrupt.

Two transmit data interrupts with separate interrupt vectors are available: transmit data with exception status and transmit data without exceptions. Table 8-4 shows the conditions under which these interrupts are generated and lists the interrupt vectors.

Table 8-4. SSI Transmit Data Interrupts

Interrupt	Vector	TIE	TUE	TDE
Transmit data with exception status	$0024	1	1	1
Transmit data (without exception)	$0026	1	0	1

8.2.8.3 Receive Enable (RE)—Bit 13

The SSI receive enable (RE) control bit enables the receive portion of the SSI. When the RE bit is set, the receive portion of the SSI is enabled. When the RE bit is cleared, the receiver is disabled by inhibiting data transfer into the receive (RX) buffer. If data is being received when this bit is cleared, the rest of the word is not shifted in, nor is it transferred to the SRX register. If the RE bit is reenabled during a time slot before the second-to-last bit, then the word will be received.

8.2.8.4 Transmit Enable (TE)—Bit 12

The SSI transmit enable (TE) control bit enables the transfer of the contents of the STX register to the transmit shift register and also enables the internal gated clock. When the TE bit is set and a word boundary is detected, the transmit portion of the SSI is enabled. When the TE bit is cleared, the transmitter continues to send the data currently in the SSI transmit shift register and then disables the transmitter. The serial output is tri-stated and any data present in the STX register is not transmitted. In other words, data can be written to the STX register with the TE bit cleared, and the TDE bit is cleared but data is not transferred to the transmit shift register. If the TE bit is cleared and then set again during the same transmitted word, the data continues to be transmitted. If the TE bit is set again during a different time slot, data is not transmitted until the next word boundary.

The normal transmit-enable sequence is to write data to the STX register or to the STSR before setting the TE bit. The normal transmit-disable sequence is to clear the TE bit and the TIE bit after the TDE bit is set. When an internal gated clock is being used, the gated clock runs during valid time slots if the TE bit is set. If the TE bit is cleared, the transmitter continues to send the data currently in the SSI transmit shift register until it is empty. Then the clock stops. When the TE bit is set again, the gated clock starts immediately and runs during any valid time slots.

8.2.8.5 Receive Buffer Enable (RBF)—Bit 11

The receive buffer enable (RBF) control bit enables the buffer register for the receive section. When the RBF bit is set, it allows for two samples to be received by the SSI (a third sample can be shifting in) before the RDF bit is set and for an interrupt request to be generated when enabled by the RIE bit. When the RBF bit is cleared, the buffer register is not used, and an interrupt request is generated when a single sample is received by the SSI. (Interrupts need to be enabled.)

8.2.8.6 Transmit Buffer Enable (TBF)—Bit 10

The transmit buffer enable (TBF) control bit enables the buffer register for the transmit section. When the TBF bit is set, two samples can be written to the SSI (a third sample can be shifting out) before the TDE bit is set, and an interrupt request can be generated when enabled by the TIE bit. When the TBF bit is cleared, the buffer register is not used, and an interrupt request is generated when a single sample needs to be written to the SSI.

8.2.8.7 Receive Direction (RXD)—Bit 9

The receive direction (RXD) control bit selects the direction and source of the clock and frame sync signals used to clock the RXSR. When the RXD bit is set, the frame sync and clock are generated internally and are output to the SRFS and SRCK pins, respectively, if not configured as general-purpose input/output (GPIO). When the RXD bit is cleared, the clock source is external, the internal clock generator is disconnected from the SRCK pin, and an external clock source can drive this pin to clock the RXSR. The SRFS pin is an input, meaning that the receive frame sync is supplied from an external source. Table 8-5 shows the frame sync and clock pin configuration.

Table 8-5. Frame Sync and Clock Pin Configuration

SYN	RXD	TXD	SRFS	STFS	SRCK	STCK
0	0	0	RFS in	TFS in	RCK in	TCK in
0	0	1	RFS in	TFS out	RCK in	TCK out
0	1	0	RFS out	TFS in	RCK out	TCK in
0	1	1	RFS out	TFS out	RCK out	TCK out
1	0	0	GPIO	FS in	GPIO	CK in
1	0	1	GPIO	FS out	GPIO	CK out
1	1	0	GPIO	GPIO	GPIO	Gated in
1	1	1	GPIO	GPIO	GPIO	Gated out

8.2.8.8 Transmit Direction (TXD)—Bit 8

The transmit direction (TXD) control bit selects the direction and source of the clock and frame sync signals used to clock the TXSR. When the TXD bit is set, the frame sync and clock are generated internally and are output to the STFS and STCK pins, respectively, if not configured as GPIO. When the TXD bit is cleared, the clock source is external, the internal clock generator is disconnected from the STCK pin, and an external clock source can drive this pin to clock the TXSR. The STFS pin is an input, meaning that the transmit frame sync is supplied from an external source.

8.2.8.9 Synchronous Mode (SYN)—Bit 7

The synchronous mode (SYN) control bit enables the synchronous mode of operation. In this mode, the transmit and receive sections share a common clock pin (STCK) and frame sync pin (STFS).

8.2.8.10 Transmit Shift Direction (TSHFD)—Bit 6

The transmit shift direction (TSHFD) control bit controls whether the MSB or LSB is transmitted first for the transmit section. If the TSHFD bit is set, the LSB is transmitted first. If the TSHFD bit is cleared, the data is transmitted MSB first.

NOTE:

The codec device labels the MSB as bit 0, whereas the DSP56824 labels the LSB as bit 0. Therefore, when using a standard codec, the DSP56824 MSB (or codec bit 0) is shifted out first, and the TSHFD bit should be cleared.

8.2.8.11 Transmit Clock Polarity (TSCKP)—Bit 5

The transmit clock polarity (TSCKP) control bit controls which bit clock edge is used to clock out data and latch in data for the transmit section. If the TSCKP bit is set, the falling edge of the bit clock is used to clock the data out. If the TSCKP bit is cleared, the data is clocked out on the rising edge of the bit clock.

8.2.8.12 SSI Enable (SSIEN)—Bit 4

The SSI enable (SSIEN) control bit enables and disables the SSI. If the SSIEN bit is set, the SSI is enabled, which causes an output frame sync to be generated when set up for internal frame sync, or causes the SSI to wait for the input frame sync when set up for external frame sync. If the SSIEN bit is cleared, the SSI is disabled. When disabled, the STCK, STFS, and STFS pins are tri-stated, the status register bits are preset to the same state produced by the DSP reset, and the control register bits are unaffected.

8.2.8.13 Network Mode (NET)—Bit 3

The network mode (NET) control bit selects the operational mode of the SSI. When the NET bit is cleared, normal mode is selected. When the NET bit is set, network mode is selected.

8.2.8.14 Frame Sync Invert (FSI)—Bit 2

The frame sync invert (FSI) control bit selects the logic of frame sync I/O. If the FSI bit is set, the frame sync is active low. If the FSI bit is cleared, the frame sync is active high.

8.2.8.15 Frame Sync Length (FSL)—Bit 1

The frame sync length (FSL) control bit selects the length of the frame sync signal to be generated or recognized. If the FSL bit is set, a one-clock-bit-long frame sync is selected. If the FSL bit is cleared, a one-word-long frame sync is selected. The length of this word-long frame sync is the same as the length of the data word selected by WL[1:0].

8.2.8.16 Early Frame Sync (EFS)—Bit 0

The early frame sync (EFS) control bit controls when the frame sync is initiated for the transmit and receive sections. When the EFS bit is cleared, the frame sync is initiated as the first bit of data is transmitted or received. When the EFS bit is set, the frame sync is initiated 1 bit before the data is transmitted or received. The frame sync is disabled after one bit-for-bit-length frame sync and after one word-for-word-length frame sync.

8.2.9 SSI Control/Status Register (SCSR)

The SSI control/status register (SCSR) is a 16-bit register used to set up and monitor the SSI. The top half of the register (bits 15–8) is the read/write portion and is used for SSI setup. The bottom half of the register (bits 7–0) is read-only and is used by the DSP56824 to interrogate the status and serial input flags of the SSI. The control and status bits are described in Section 8.2.9.1, “Reserved Bit—Bit 15,” through Section 8.2.9.15, “Transmit Data Buffer Empty (TDBE)—Bit 0.” See Figure 8-6 on page 8-7 for the programming models of the SCSR.

NOTE:

All the flags in the status portion of the SCSR are updated after the first bit of the next SSI word has completed transmission or reception. Some status bits (ROE and TUE) are cleared by reading the SCSR followed by a read or write to either the SRX or STX register. Because of this, the control bits in the top half of the SCSR should only be read when the SSI is disabled (SSIEN = 0).

8.2.9.1 Reserved Bit—Bit 15

Bit 15 is reserved and is read as zero during read operations. This bit should be written with zero to ensure future compatibility.

8.2.9.2 Receive Shift Direction (RSHFD)—Bit 14

The receive shift direction (RSHFD) control bit controls whether the MSB or LSB is received first for the receive section. If the RSHFD bit is cleared, data is received MSB first. If the RSHFD bit is set, the LSB is received first.

NOTE:

The codec device labels the MSB as bit 0, whereas the DSP56824 labels the LSB as bit 0. Therefore, when using a standard codec, the DSP56824 MSB (or codec bit 0) is shifted out first, and the RSHFD bit should be cleared.

8.2.9.3 Receive Clock Polarity (RSCKP)—Bit 13

The receive clock polarity (RSCKP) control bit controls which bit clock edge is used to latch in data for the receive section. If the RSCKP bit is cleared, the data is latched in on the falling edge of the clock. If the RSCKP bit is set, the rising edge of the clock is used to latch the data in.

8.2.9.4 Reserved Bits—Bits 12–11

Bits 12 and 11 are reserved and are read as zero during read operations. These bits should be written with zero to ensure future compatibility.

8.2.9.5 Receive Frame Sync Invert (RFSI)—Bit 10

The receive frame sync invert (RFSI) control bit selects the logic of frame sync I/O for the receive section. If the RFSI bit is set, the frame sync is active low. If the RFSI bit is cleared, the frame sync is active high.

8.2.9.6 Receive Frame Sync Length (RFSL)—Bit 9

The receive frame sync length (RFSL) control bit selects the length of the frame sync signal to be generated or recognized for the receive section. If the RFSL bit is set, then a one-clock-bit-long frame sync is selected. If the RFSL bit is cleared, a one-word-long frame sync is selected. The length of this word-long frame sync is the same as the length of the data word selected by WL[1:0].

8.2.9.7 Receive Early Frame Sync (REFS)—Bit 8

The receive early frame sync (REFS) control bit controls when the frame sync is initiated for the receive section. When the REFS bit is cleared, the frame sync is initiated as the first bit of data is received. When the REFS bit is set, the frame sync is initiated 1 bit before the data is received. The frame sync is disabled after one bit-for-bit-length frame sync and after one word-for-word-length frame sync.

8.2.9.8 Receive Data Register Full (RDF)—Bit 7

The SSI receive data register full (RDF) flag bit is set when the SRX register is loaded with a new value. If the receive buffer is enabled, the SRX register is loaded from the receive data buffer register. Otherwise, the SRX register is loaded from the RXSR. RDF is cleared when the DSP56824 reads the SRX register. If the RIE bit is set, a DSP receive data interrupt request is issued when the RDF bit is set. The interrupt request vector depends on the state of the receiver overrun (ROE) bit (in the SCSR). The RDF bit is cleared by DSP, SSI, and STOP reset.

8.2.9.9 Transmit Data Register Empty (TDE)—Bit 6

The SSI transmit data register empty (TDE) flag bit is set when there is no data waiting to be transferred to the STX register. If the transmit buffer is enabled, this occurs when the data in the STX register has been transferred first into the transmit data buffer register and then into the TXSR before a new value has been written to the STX register. If the transmit buffer is not enabled, this occurs when the contents of the STX register is transferred into the TXSR. When set, the TDE bit indicates that data should be written to the STX register or to the STSR before the transmit shift register becomes empty, which would cause an underrun error.

The TDE bit is cleared when the DSP56824 writes to the STX register or to the STSR to disable transmission of the next time slot. If the TIE bit is set, an SSI transmit data interrupt request is issued when the TDE bit is set. The vector of the interrupt depends on the state of the TUE bit (in the SCSR). The TDE bit is set by DSP, SSI, and STOP reset.

8.2.9.10 Receiver Overrun Error (ROE)—Bit 5

The receiver overrun error (ROE) flag bit is set when the RXSR is filled and ready to transfer to the SRX register or the receive data buffer register (when enabled), and these registers are already full, as indicated by the RDF bit being set. The RXSR is not transferred in this case. A receiver overrun error does not cause any interrupts. However, when the ROE bit is set, it causes a change in the interrupt vector used, allowing the use of a different interrupt handler for a receiver overrun condition. If a receive interrupt occurs with the ROE bit set, the receive data with exception status interrupt (vector $0020) is generated. If a receive interrupt occurs with the ROE bit cleared, the receive data interrupt (vector $0022) is generated.

The ROE bit is cleared by DSP, SSI, or STOP reset and is cleared by reading the SCSR with the ROE bit set, followed by reading the SRX register. Clearing the RE bit does not affect the ROE bit.

8.2.9.11 Transmitter Underrun Error (TUE)—Bit 4

The transmitter underrun error (TUE) flag bit is set when the TXSR is empty (no data to be transmitted), as indicated by the TDE bit being set, and a transmit time slot occurs. When a transmitter underrun error occurs, the previously sent data is retransmitted.

A transmit time slot in the normal mode occurs when the frame sync is asserted. In network mode, each time slot requires data transmission and is, therefore, a transmit time slot (TE = 1).

The TUE bit does not cause any interrupts. However, the TUE bit does cause a change in the interrupt vector used for transmit interrupts so that a different interrupt handler can be used for a transmit underrun condition. If a transmit interrupt occurs with the TUE bit set, the transmit data with exception status interrupt (vector $0024) is generated. If a transmit interrupt occurs with the TUE bit cleared, the transmit data interrupt (vector $0026) is generated.

The TUE bit is cleared by DSP, SSI, or STOP reset. The TUE bit is also cleared by reading the SCSR with the TUE bit set, followed by writing to the STX register or to the STSR.

8.2.9.12 Transmit Frame Sync (TFS)—Bit 3

When set, the transmit frame sync (TFS) flag bit indicates that a frame sync occurred during transmission of the last word written to the STX register. Data written to the STX register during the time slot when the TFS bit is set is sent during the second time slot (in network mode) or in the next first time slot (in normal mode). In network mode, the TFS bit is set during transmission of the first slot of the frame. It is then cleared when starting transmission of the next slot. The TFS bit is cleared by DSP, SSI, or STOP reset.

8.2.9.13 Receive Frame Sync (RFS)—Bit 2

When set, the receive frame sync (RFS) flag bit indicates that a frame sync occurred during reception of the next word into the SRX register. In network mode, the RFS bit is set while the first slot of the frame is being received. It is cleared when the next slot of the frame begins to be received. The RFS bit is cleared by DSP, SSI, or STOP reset.

NOTE:

In synchronous mode (the SCR2's SYN bit is set to one), the RFS bit is not valid. In asynchronous mode (the SYN bit is cleared), the RFS bit indicates the state of the SRFS pin. (See section Section 8.3, "SSI Data and Control Pins," for details on the SRFS pin.) When the SSI is being used in synchronous mode, the TFS bit should be used to check the state of the STFS pin, since that pin is used for both transmit and receive frame sync in synchronous mode.

8.2.9.14 Receive Data Buffer Full (RDBF)—Bit 1

The receive data buffer full (RDBF) flag bit is set when the receive section is programmed with the receive buffer enabled, and the contents of the RXSR are transferred to the receive data buffer register. When set, RDBF indicates that data can be read from the SRX register. Note that an interrupt is only generated if both the RDF and RIE bits are set. The RDBF bit is cleared by DSP, SSI, or STOP reset.

8.2.9.15 Transmit Data Buffer Empty (TDBE)—Bit 0

The transmit data buffer empty (TDBE) flag bit is set when the transmit section is programmed with the transmit buffer enabled and the contents of the transmit data buffer register are transferred to the TXSR. When set, the TDBE bit indicates that data can be written to the STX register. Note that an interrupt is generated only if both the TDE and the TIE bits are set. The TDBE bit is set by DSP, SSI, or STOP reset.

8.2.10 SSI Time Slot Register (STSR)

The SSI time slot register (STSR) is used when data is not to be transmitted in an available transmit time slot. For the purposes of timing, the time slot register is a write-only register that behaves like an alternate transmit data register, except that instead of transmitting data, the STD pin is tri-stated. Using this register is important for avoiding overflow or underflow during inactive time slots.

8.3 SSI Data and Control Pins

The SSI has the following six dedicated I/O pins:

- Serial transmit data (STD/PC8)
- Serial receive data (SRD/PC9)
- Serial transmit clock (STCK/PC10)
- Serial transmit frame sync (STFS/PC11)
- Serial receive clock (SRCK/PC12)
- Serial receive frame sync (SRFS/PC13)

Figure 8-9 on page 8-20 and Figure 8-10 on page 8-21 show the main SSI configurations. These pins support all transmit and receive functions with a continuous or gated clock, as shown. Note that gated clock implementations do not require the use of the frame sync pins (STFS and SRFS). In this case, these pins can be used as GPIO pins, if desired.

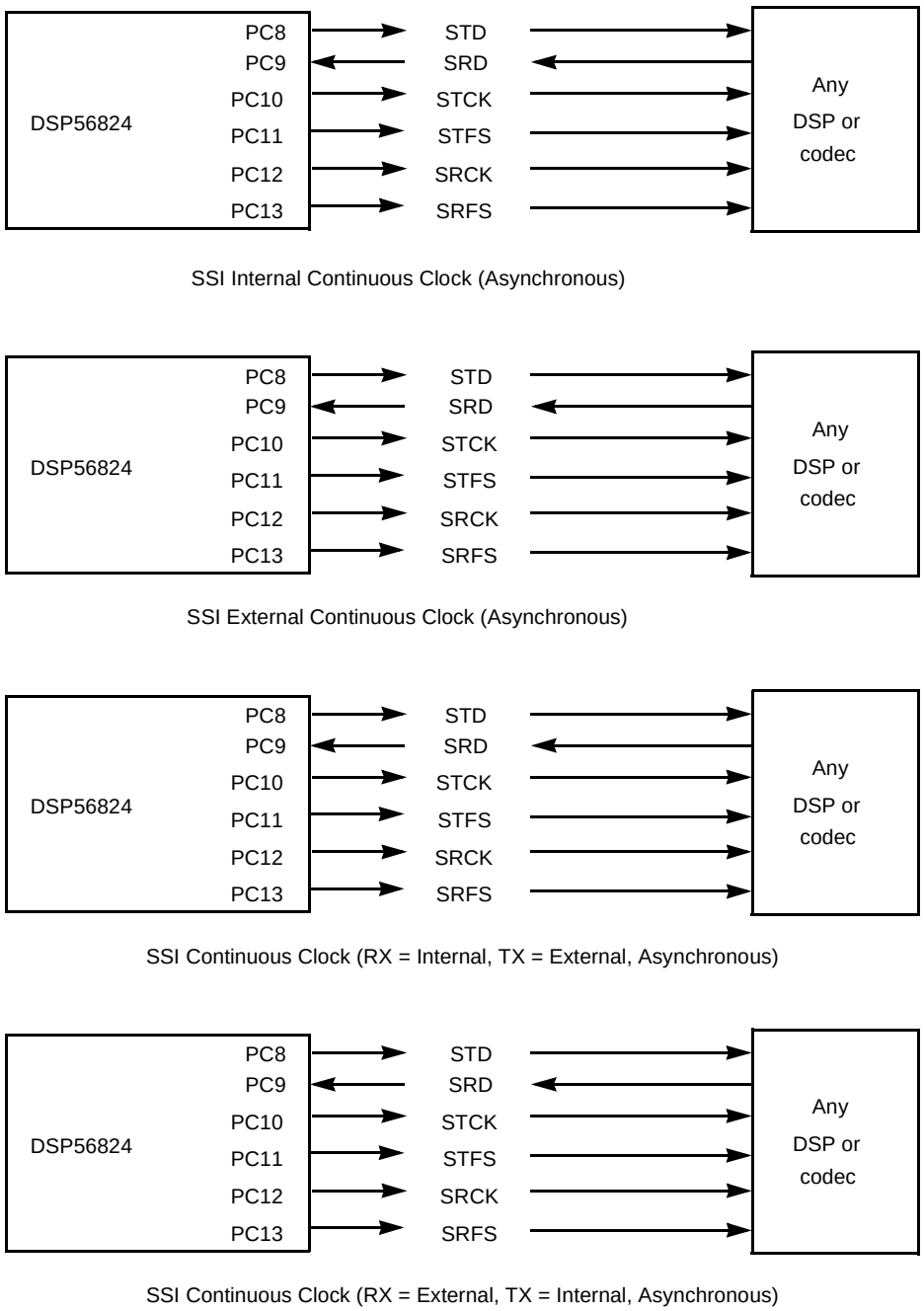


Figure 8-9. Asynchronous SSI Configurations—Continuous Clock

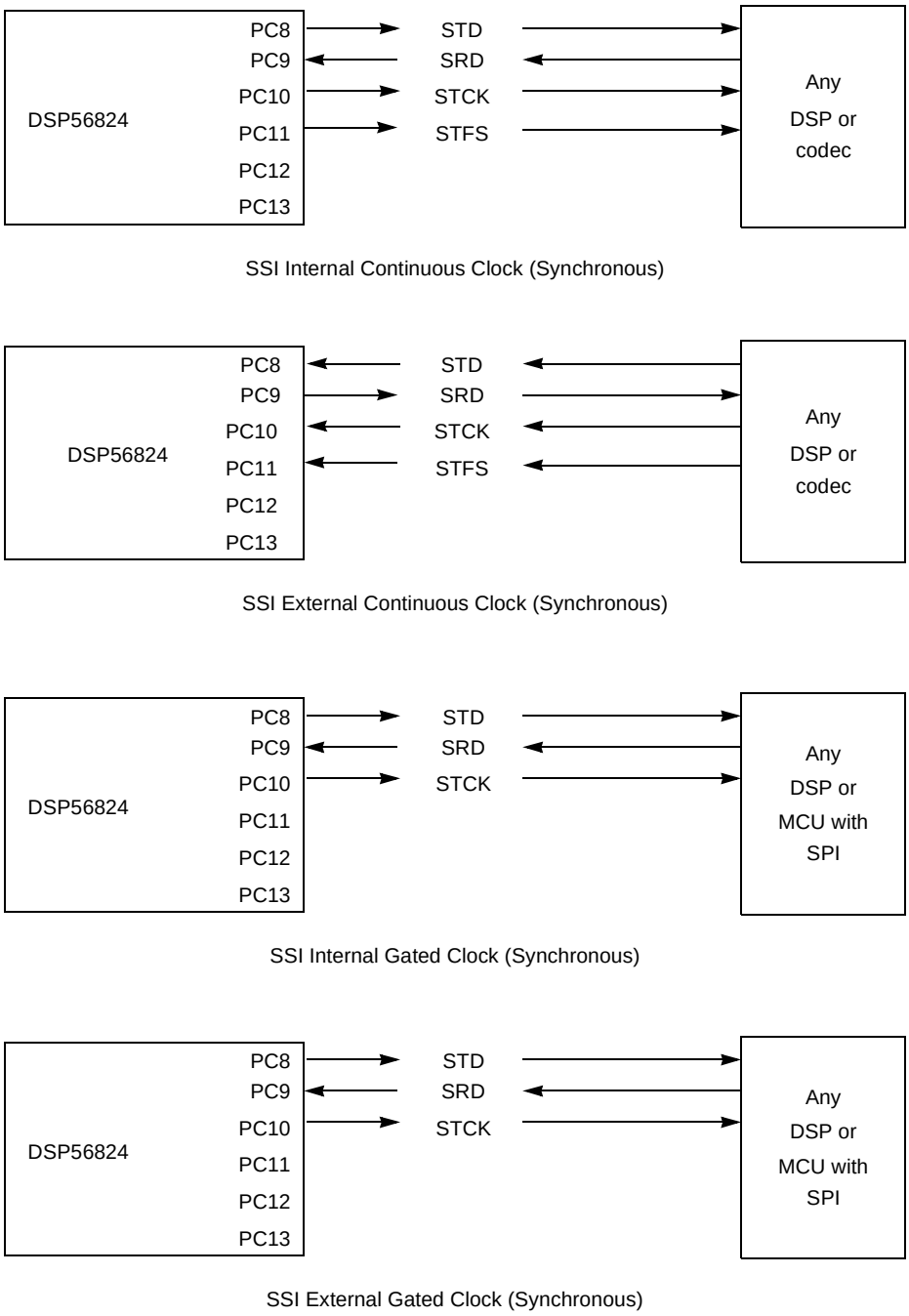


Figure 8-10. Synchronous SSI Configurations—Continuous and Gated Clock

The following paragraphs describe the configuration of the SSI pins.

- **STD/PC8 (serial transmit data)—**The STD pin transmits data from the serial transmit shift register. The STD pin is an output pin when data is being transmitted and is tri-stated between data word transmissions and on the trailing edge of the bit clock after the last bit of a word is transmitted. Connect an external resistor to this pin to prevent the signal from floating when not being driven. (A floating pin may provide spurious edge transitions or may go into oscillation.) Since this pin is tri-stated, the external resistor can be either high or low, depending on the circuit designer's choice.
- **SRD/PC9 (serial receive data)—**The SRD pin is used to bring serial data into the receive data shift register.

- **STCK/PC10 (serial transmit clock)—**The STCK pin can be used as either an input or an output. This clock signal is used by the transmitter and can be either continuous or gated. During gated clock mode, data on the STCK pin is valid only during the transmission of data; otherwise it is tri-stated. In synchronous mode, this pin is used by both the transmit and receive sections. When gated clock mode is being used, an external resistor should be connected to this pin to prevent the signal from floating when not being driven.
- **STFS/PC11 (serial transmit frame sync)—**The STFS pin can be used as either an input or an output. The frame sync is used by the transmitter to synchronize the transfer of data. The frame sync signal can be 1 bit or 1 word in length and can occur 1 bit before the transfer of data or right at the transfer of data. In synchronous mode, this pin is used by both the transmit and receive sections. In gated clock mode, frame sync signals are not used.
- **SRCK/PC12 (serial receive clock)—**The SRCK pin can be used as either an input or an output. This clock signal is used by the receiver and is always continuous. During gated clock mode, the STCK pin is used instead for clocking in data. This pin is not used in synchronous mode.
- **SRFS/PC13 (serial receive frame sync)—**The SRFS pin can be used as either an input or an output. The frame sync is used by the receiver to synchronize the transfer of data. The frame sync signal can be 1 bit or 1 word in length and can occur 1 bit before the transfer of data or right at the transfer of data.

An example of the pin signals for an 8-bit data transfer is shown in Figure 8-11. Continuous and gated clock signals are shown, as well as the bit-length frame sync signal and the word-length frame sync signal. Note that the shift direction can be defined as MSB first or LSB first, and that there are other options on the clock and frame sync.

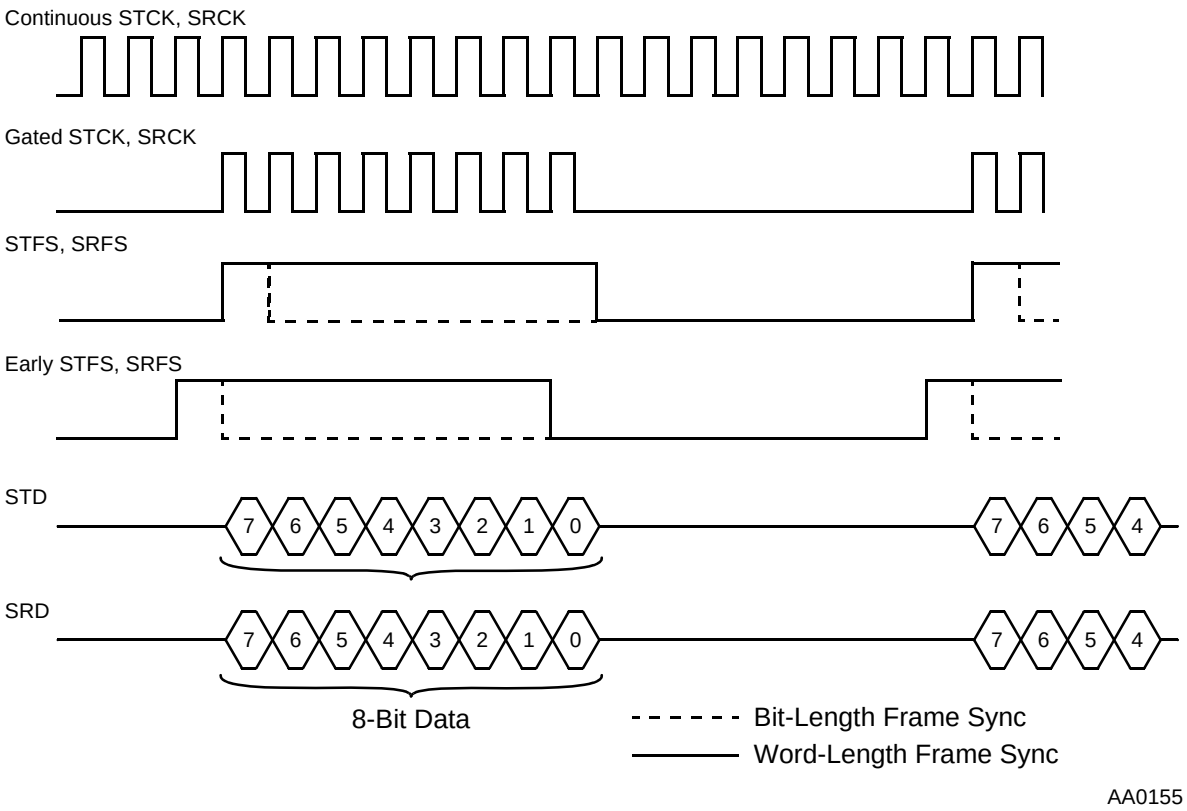


Figure 8-11. Serial Clock and Frame Sync Timing

8.4 SSI Operating Modes

The SSI has three basic operating modes, with the option of asynchronous or synchronous protocol, as follows:

- Normal mode
  - Asynchronous protocol
  - Synchronous protocol
- Network mode
  - Asynchronous protocol
  - Synchronous protocol
- Gated clock mode
  - Synchronous protocol only

These modes can be programmed by several bits in the SSI control registers. Table 8-6 lists these operating modes and some of the typical applications in which they can be used:

Table 8-6. SSI Operating Modes

TX, RX Sections	Serial Clock	Mode	Typical Applications
Asynchronous	Continuous	Normal	Multiple synchronous codecs
Asynchronous	Continuous	Network	TDM codec or DSP networks
Synchronous	Continuous	Normal	Multiple synchronous codecs
Synchronous	Continuous	Network	TDM codec or DSP networks
Synchronous	Gated	Normal	SPI-type devices; DSP to MCU

The transmit and receive sections of the SSI can be synchronous or asynchronous. In synchronous mode, the transmitter and the receiver use a common clock and frame synchronization signal. In asynchronous mode, the transmitter and receiver each has its own clock and frame synchronization signals. Continuous or gated clock mode can be selected. In continuous mode, the clock runs continuously. In gated clock mode, the clock is only functioning during transmission.

Normal or network mode can also be selected. In normal mode, the SSI functions with one data word of I/O per frame. In network mode, any number from 2 to 32 data words of I/O per frame can be used. Network mode is typically used in star or ring time division multiplex networks with other processors or codecs, allowing an interface to time division multiplexed networks without additional logic. Use of the gated clock is not allowed in network mode. These distinctions result in the basic operating modes that allow the SSI to communicate with a wide variety of devices.

The SSI supports both normal and network modes, and these can be selected independently of whether the transmitter and receiver are synchronous or asynchronous. Typically these protocols are used in a periodic manner, where data is transferred at regular intervals, such as at the sampling rate of an external codec.

Both modes use the concept of a frame. The beginning of the frame is marked with a frame sync when programmed with a continuous clock. The frame sync occurs at a periodic interval. The length of the frame is determined by the DC[4:0] bits in either the SCRRX or SCRTX register, depending on whether data is being transferred or received. The number of words transferred per frame depends on the mode of the SSI.

In normal mode, one data word is transferred per frame. In network mode, the frame is divided into anywhere between 2 and 32 time slots, where in each time slot 1 data word can optionally be transferred.

8.4.1 Normal Mode

Normal mode is the simplest mode of the SSI. It is used to transfer one word per frame. In continuous clock mode, a frame sync occurs at the beginning of each frame.

The length of the frame is determined by the following factors:

- The period of the serial bit clock (PSR, PM[7:0] bits for internal clock or the frequency of the external clock on the STCK pin)
- The number of bits per sample (WL[1:0] bits)
- The number of time slots per frame (DC[4:0] bits)

If normal mode is configured to provide more than one time slot per frame, data is transmitted only in the first time slot. No data is transmitted in subsequent time slots.

8.4.1.1 Normal Mode Transmit

The conditions for data transmission from the SSI in normal mode are:

1. SSI enabled (SSIEN = 1)
2. Transmitter enabled (TE = 1)
3. Frame sync active (for continuous clock case)
4. Bit clock begins (for gated clock case)

When the preceding conditions occur in normal mode, the next data word is transferred into the TXSR from the STX register, or from the transmit data buffer register, if transmit buffering is enabled. The new data word is transmitted immediately. If buffering is not enabled, the TDE bit is set (transmitter empty), and the transmit interrupt occurs if the TIE bit is set (transmit interrupt is enabled). If buffering is enabled, the TDE bit is set (transmitter empty) and the transmit interrupt occurs, if the TIE bit is set (transmit interrupt is enabled), when both values have been transferred to the TXSR. If buffering is enabled, a second data word can be transferred and shifted before the DSP56824 must write new data to the STX register.

The STD pin is tri-stated except during the data transmission period. For a continuous clock, the optional frame sync output and clock outputs are not tri-stated, even if both receiver and transmitter are disabled.

8.4.1.2 Normal Mode Receive

The conditions for data reception from the SSI are:

1. SSI enabled (SSIEN = 1)
2. Receiver enabled (RE = 1)
3. Frame sync active (for continuous clock case)
4. Bit clock begins (for gated clock case)

With the preceding conditions in normal mode with a continuous clock, each time the frame sync signal is generated (or detected), a data word is clocked in. With the preceding conditions and a gated clock, each time the clock begins, a data word is clocked in. If buffering is not enabled, the data word, after being received, is transferred from the RXSR to the SRX register, the RDF flag is set (receiver full), and the receive interrupt occurs if it is enabled (the RIE bit is set). If buffering is enabled, the data word, after being received, is transferred to the receive data buffer register. The RDF flag is set if both the SRX register and receive data buffer register are full, and the receive interrupt occurs if it is enabled (the RIE bit is set).

The DSP56824 program has to read the data from the SRX register before a new data word is transferred from the RXSR; otherwise the ROE bit is set. If buffering is enabled, the ROE bit is set when both the SRX register and the receive data buffer register contain data and a new data word is ready to be transferred to the receive data buffer register.

Figure 8-12 shows transmitter and receiver timing for an 8-bit word with two words per time slot in normal mode, continuous clock, with a late word length frame sync.

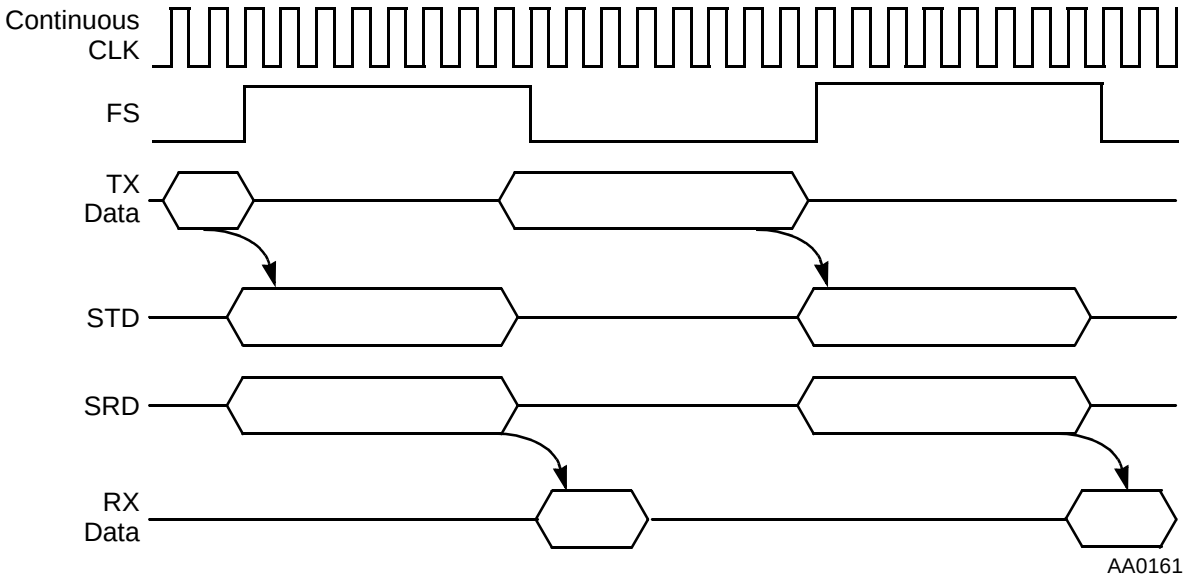


Figure 8-12. Normal Mode Timing—Continuous Clock

Figure 8-13 shows a similar case for gated clock. Note that a pull-down resistor is required in the gated clock case because the clock pin is tri-stated between transmissions.

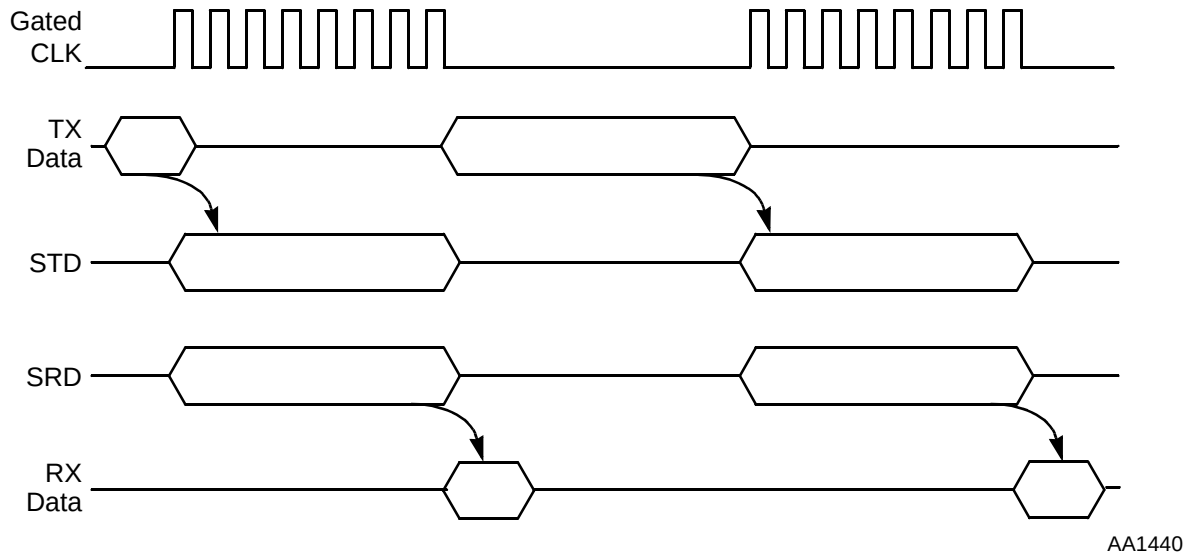


Figure 8-13. Normal Mode Timing—Gated Clock

8.4.2 Network Mode

Network mode is used for creating a time division multiplexed (TDM) network, such as a TDM codec network or a network of DSPs. In continuous clock mode, a frame sync occurs at the beginning of each frame. In this mode, the frame is divided into more than one time slot. During each time slot, one data word can be transferred. Each time slot is then assigned to an appropriate codec or DSP on the network. The DSP can be a master device that controls its own private network, or a slave device that is connected to an existing TDM network and occupies a few time slots.

The frame sync signal indicates the beginning of a new data frame. Each data frame is divided into time slots, and transmission or reception (or both) of one data word can occur in each time slot (rather than in just the frame sync time slot as in normal mode). The frame rate dividers, controlled by the DC[4:0] bits, select 2 to 32 time slots per frame. The length of the frame is determined by the following factors:

- The period of the serial bit clock (PSR, PM[7:0] bits for internal clock, or the frequency of the external clock on the STCK pin)
- The number of bits per sample (WL[1:0] bits)
- The number of time slots per frame (DC[4:0] bits)

In network mode, data can be transmitted in any time slot. The distinction of the network mode is that each time slot is identified with respect to the frame sync (data word time). This time slot identification allows the option of transmitting data during the time slot by writing to the STX register or ignoring the time slot by writing to STSR. The receiver is treated in the same manner, except that data is always being shifted into the RXSR and transferred to the SRX register. The DSP56824 reads the SRX register and either uses it or discards it.

8.4.2.1 Network Mode Transmit

The transmit portion of SSI is enabled when the SSIEN and the TE bits in the SCR2 are both set. However, for continuous clock, when the TE bit is set, the transmitter is enabled only after detection of a new time slot (if the TE bit is set during a slot other than the first). *Software has to find the start of the next frame.*

The normal startup sequence for transmission is to do the following:

1. Write the data to be transmitted to the STX register. This clears the TDE flag.
2. Set the TE bit to enable the transmitter on the next word boundary (for a continuous clock case).
3. Enable transmit interrupts.

Alternatively, the programmer may decide *not* to transmit in a time slot by writing to the STSR. This clears the TDE flag just as if data were going to be transmitted, but the STD pin remains tri-stated during the time slot.

When the frame sync is detected or generated (continuous clock), the first enabled data word is transferred from the STX register to the TXSR and is shifted out (transmitted). When the STX register is empty, the TDE bit is set, which causes a transmitter interrupt to be sent if the TIE bit is set. Software can poll the TDE bit or use interrupts to reload the STX register with new data for the next time slot or write to the STSR to prevent transmitting in the next time slot. Failing to reload the STX register (or writing to the STSR) before the TXSR is finished shifting (empty) causes a transmitter underrun and the TUE error bit to be set, and the STD pin is tri-stated for the next time slot.

The operation of clearing the TE bit disables the transmitter after the completion of the transmission of the current data word. Setting the TE bit enables the transmission of the next word. During that time the STD pin is tri-stated. The TE bit should be cleared after the TDE bit is set to ensure that all pending data is transmitted.

To summarize, the network mode transmitter generates interrupts every enabled time slot and requires the DSP program to respond to each enabled time slot. These responses may be one of the following:

- Write the data register with data to enable transmission in the next time slot.
- Write the time slot register to disable transmission in the next time slot.
- Do nothing—transmitter underrun occurs at the beginning of the next time slot and the previous data is retransmitted.

8.4.2.2 Network Mode Receive

The receiver portion of the SSI is enabled when both the SSIEN and the RE bits in the SCR2 are set. However, the receive enable only takes place during that time slot if RE is enabled before the second-to-last bit of the word. If the RE bit is cleared, the receiver is disabled immediately. *Software has to find the start of the next frame*.

When the word is completely received, it is transferred to the SRX register, which sets the RDF bit (receive data register full). Setting the RDF bit causes a receive interrupt to occur if the receiver interrupt is enabled (the RIE bit is set).

The second data word (second time slot in the frame) begins shifting in immediately after the transfer of the first data word to the SRX register. The DSP program has to read the data from the receive data register (which clears RDF) before the second data word is completely received (ready to transfer to RX data register) or a receiver overrun error occurs (the ROE bit is set).

An interrupt can occur after the reception of each enabled data word, or the programmer can poll the RDF flag. The DSP56824 program response can be one of the following:

- Read RX and use the data.
- Read RX and ignore the data.
- Do nothing—the receiver overrun exception occurs at the end of the current time slot.

NOTE:

For a continuous clock, the optional frame sync output and clock output signals are not affected, even if the transmitter or receiver is disabled. TE and RE do not disable the bit clock or the frame sync generation. The only way to disable the bit clock and the frame sync generation is to disable the SSIEN bit in the SCR2.

The transmitter and receiver timing for an 8-bit word with a continuous clock, buffering disabled, and three words per frame sync in network mode is shown in Figure 8-14 on page 8-28.

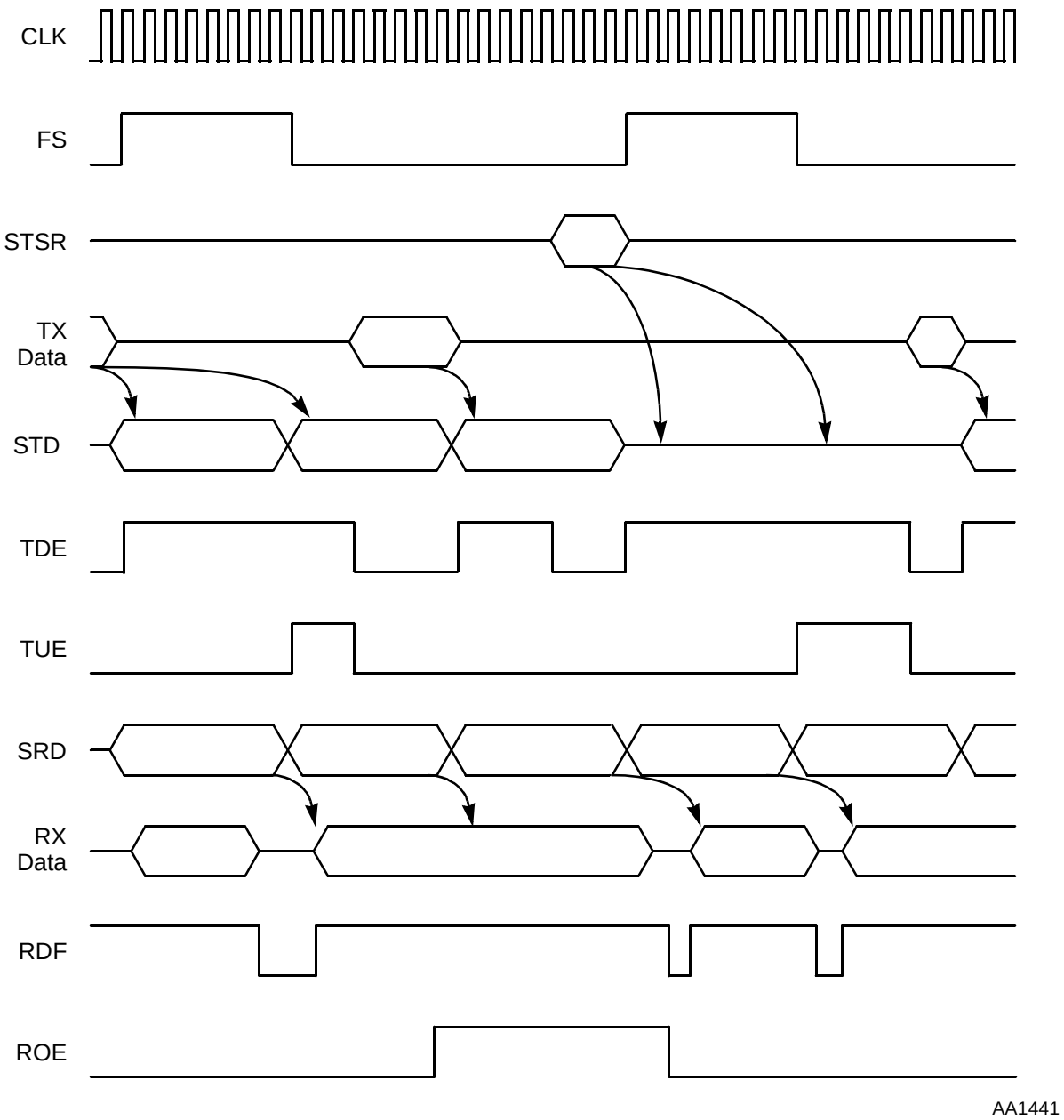


Figure 8-14. Network Mode Timing—Continuous Clock

8.4.3 Gated Clock Operation

Gated clock mode is often used to hook up to SPI-type interfaces on microcontroller units (MCUs) or external peripheral chips. In gated clock mode, the presence of the clock indicates that valid data is on the STD or SRD pins. For this reason, no frame sync is needed in this mode. Once the transmission of data has completed, the clock pin is tri-stated. Gated clocks are allowed for both the transmit and receive sections with either an internal or external clock and in normal mode. Gated clocks are not allowed in network mode.

The clock runs when the TE bit, the RE bit, or both are appropriately enabled. For the case of an internally generated clock, all internal bit clocks, word clocks, and frame clocks continue to operate. When a valid time slot occurs (such as the first time slot in normal mode), the internal bit clock is enabled onto the

appropriate clock pin. This allows data to be transferred out in periodic intervals in gated clock mode. With an external clock, the SSI waits for a clock signal to be received. Once the clock begins, valid data is shifted in.

NOTE:

In gated clock mode with the clock generated externally, the receive interrupt does not occur until the first serial clock of the following word. This can appear in an application as a one-word shift in a series of received data words.

In general, the bit clock pins must be kept free of timing glitches. If a single glitch occurs, all ensuing transfers will be out of synchronization.

8.5 SSI Reset and Initialization Procedure

The SSI is affected by three types of reset:

- **DSP reset**—The DSP reset is generated by asserting either the $\overline{\text{RESET}}$ pin or the Computer Operating Properly (COP) timer reset. The DSP reset clears the SSIEN bit in SCR2, which disables the SSI. All other status and control bits in the SSI are affected as described in Section 8.2, “SSI Programming Model.”
- **SSI reset**—The SSI reset is generated when the SSIEN bit in the SCR2 is cleared. The SSI status bits are preset to the same state produced by the DSP reset. The SSI control bits are unaffected. The control bits in the top half of the SCSR are also unaffected. The SSI reset is useful for selective resetting of the SSI without changing the present SSI control bits and without affecting the other peripherals.
- **STOP reset**—The STOP reset is caused by executing the STOP instruction. While in stop mode, no clock is active in the SSI, which is always powered down in stop mode. The SSI status bits are preset to the same state produced by the DSP reset. The SSI control bits are unaffected. The control bits in the top half of the SCSR are also unaffected.

The correct sequence to initialize the SSI is as follows:

1. Issue a DSP or SSI reset.
2. Program SSI control registers.
3. Set the SSIEN bit in SCR2.

To ensure proper operation of the SSI, the DSP programmer should use the DSP or SSI reset before changing any of the control bits listed in Table 8-7 on page 8-30. These control bits should not be changed during SSI operation.

Table 8-7. SSI Control Bits Requiring Reset Before Change

Control Register	Bit
SCRRX SCRTX	WL0 WL1
SCR2	TEFS TFSI TFSL NET RBF RXD TSCKP TSHFD SYN TBF TXN
SCSR	REFS RFSI RFSL RSCKP RSHFD

NOTE:

The SSI bit clock must go low for at least one complete period to ensure proper SSI reset.

8.6 Configuring Port C for SSI Functionality

The Port C control (PCC) register is used to individually configure each pin as either an SSI pin or a GPIO pin. Setting the corresponding CC bit in the PCC register configures the pin as an SSI pin. When the PCC register bit is set, it is not necessary to program the corresponding PCDDR bit. The SSI peripheral ensures the correct direction of this pin. Programming the Port C data direction register (PCDDR) is necessary only when a pin is programmed as a GPIO pin.

Chapter 9
Timers

This section describes the timer module provided on the DSP56824 as a part of Port C. The timer module provides three independently programmable 16-bit timer/event counters, which are referred to as Timer 0, Timer 1, and Timer 2. All three timer/event counters can be clocked with signals coming from one of two internal sources. Timer 1 and Timer 2 can also be clocked by the overflow events of Timer 0 and Timer 1, respectively. In addition, when configured as inputs, the counters can be clocked with external signals from the timer I/O pins (TIO01 or TIO2) on Port C to count external events. The same pins, when configured as outputs, can be used to provide a timer pulse or for timer clock generation. The timer/event counters can be used either to interrupt the DSP56824 or to signal an external device at periodic intervals.

The capabilities of the timer module include the ability to do the following:

- Decrement a timer to zero and interrupt
- Decrement a timer to zero and apply a pulse to a TIO pin
- Decrement a timer to zero and toggle a TIO pin (50 percent duty cycle)
- Generate a wave form on a TIO pin with a duty cycle other than 50 percent using two timers
- Count events on an external TIO pin when selected as the input clock
- Operate timers independently or cascade timers together

Each timer can be clocked from the following:

- A TIO pin
- A clock running at half the instruction rate of the chip
- A slow clock generated from the prescaler divider

In addition, Timer 1 can be clocked by the overflow events of Timer 0, and Timer 2 can be clocked by the overflow events of Timer 1.

When a timer is clocked using the slower clock from the prescaler divider, it can continue counting even when the DSP56800 core is in stop mode. All three timers are capable of operating in stop mode. However, only Timer 2 is capable of bringing the DSP56800 core out of stop mode when it times out. Figure 9-1 on page 9-2 shows the timers provided on Port C, and Figure 9-2 on page 9-3 shows a block diagram of the timer module.

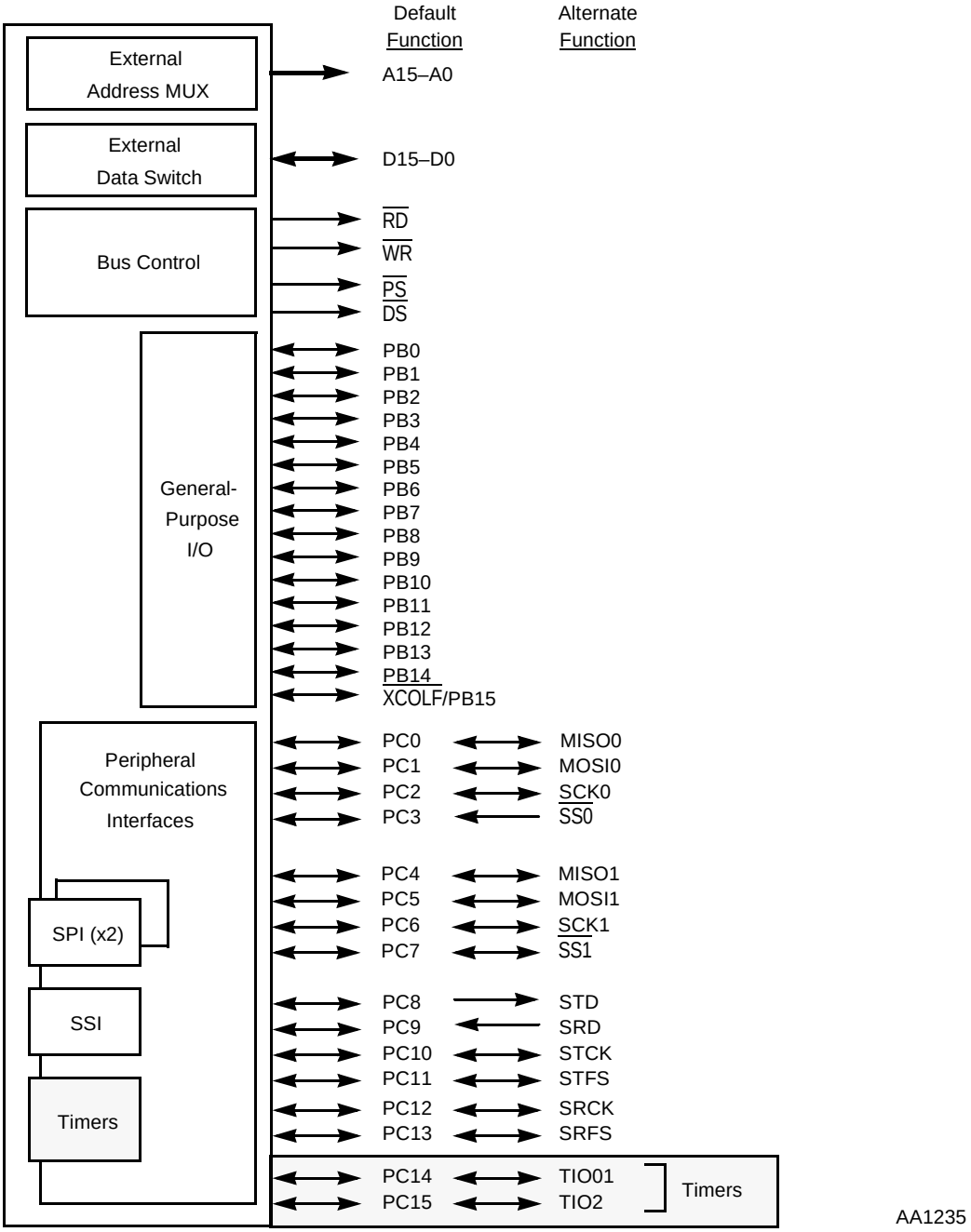


Figure 9-1. DSP56824 Input/Output Block Diagram

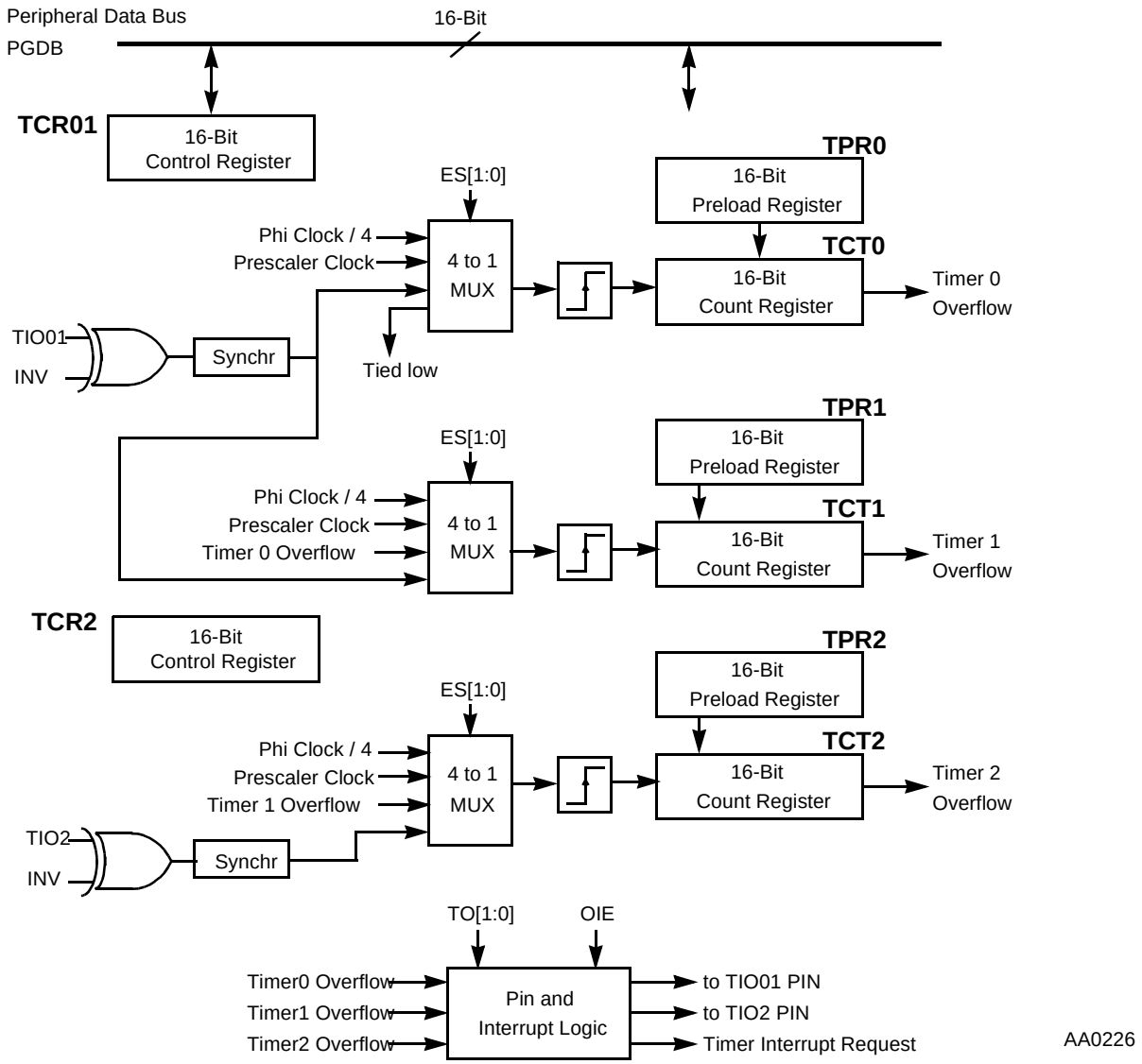


Figure 9-2. Timer Module

9.1 Timer Programming Model

The timer module contains eight read/write registers, all of which are memory-mapped in the X memory space. These eight registers include three sets of two 16-bit registers, one set for each timer: the timer count (TCT) register and the timer preload register (TPR). In addition, two timer control registers (TCR01 and TCR2) control the operations of the timers.The TCR01 provides control for Timers 0 and 1. The TCR2 provides control for Timer 2 (the top 8 bits of the TCR2 are not used). The timer module's programming model is shown in Figure 9-3 on page 9-4.

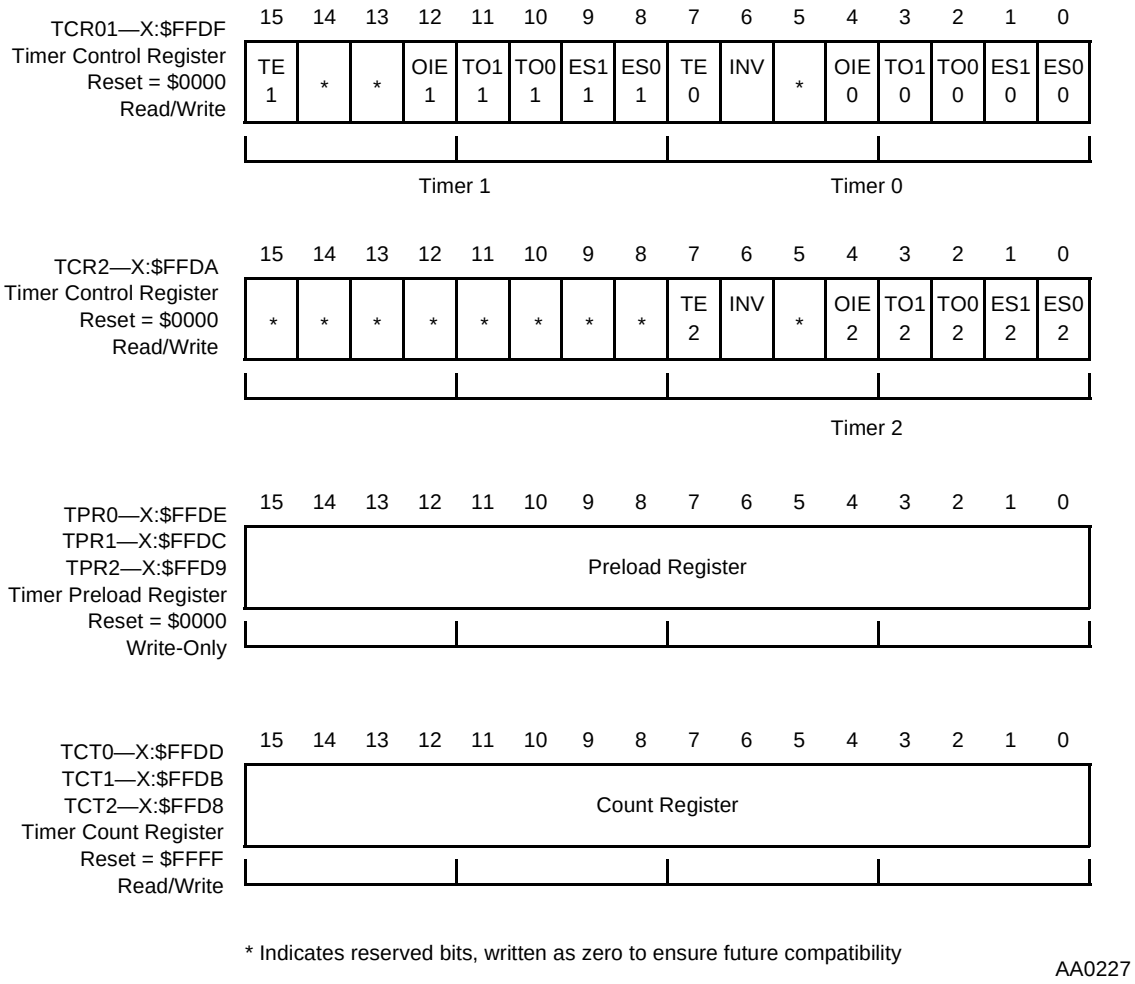


Figure 9-3. Timer Module Programming Model

NOTE:

To use the timer module, the CC[15:14] bits in the Port C control (PCC) register must be correctly set.

9.1.1 Timer Control Registers (TCR01 and TCR2)

Two 16-bit, read/write timer control registers (TCRs) contain the control bits for three 16-bit timers. The upper byte controls one timer and the lower byte controls another timer, as shown in Table 9-1.

Table 9-1. Timer Control Registers (TCR01 and TCR02)

Timer	Control Register Used
0	TCR01[7:0]
1	TCR01[15:8]
2	TCR2[7:0]
(Unused)	TCR2[15:8]

The timer control bits are defined in Section 9.1.1.1, “Timer Enable (TE)—Bit 15, Bit 7,” through Section 9.1.1.6, “Reserved TCR Bits.”

9.1.1.1 Timer Enable (TE)—Bit 15, Bit 7

The timer enable (TE) control bit (bit 7 in TCR01 for Timer 0, bit 15 in TCR01 for Timer 1, or bit 7 in TCR2 for Timer 2) is used to enable or disable the timer. Setting the TE bit to one enables the timer. The counter starts decrementing from its preset value each time an event comes in. Clearing the TE bit disables the timer. The count register is not affected by this operation. However, if a direct write to the count register occurs after the last count register reload, the value written is loaded into the count register instead of the preload value. The TE bit is cleared by reset.

9.1.1.2 Invert (INV)—Bit 6

When the invert (INV) control bit (bit 6 in TCR01 for the TIO01 pin, or bit 6 in TCR2 for the TIO2 pin) is set, the external signal coming in the TIO pin (TIO01 or TIO2, depending on which register has its INV bit set) is inverted before being synchronized and entering the 16-bit counter. All one-to-zero transitions of the TIO pin then decrement the 16-bit counter. When the INV bit is cleared, the external signal on TIO is not inverted and the 16-bit counter is decremented on all zero-to-one transitions. The INV bit is cleared on reset. See Table 9-2.

Table 9-2. INV Bit Definition

INV	External Event on TIO Pin
0	Detects rising edges
1	Detects falling edges

9.1.1.3 Overflow Interrupt Enable (OIE)—Bit 12, Bit 4

When the overflow interrupt enable (OIE) control bit (bit 4 in TCR01 for Timer 0, bit 12 in TCR01 for Timer 1, or bit 4 in TCR2 for Timer 2) is set, an interrupt is requested to the DSP56824 at the next event after the count register reaches zero. When the OIE bit is cleared, the interrupt is disabled and any pending interrupts are cleared. The OIE bit can be individually set for each timer, selectively enabling the interrupt capability independently for each timer. The OIE bit is cleared on reset.

As with all on-chip peripheral interrupts for the DSP56824, the status register (SR) must first be set to enable maskable interrupts (interrupts of level IPL 0). Next, the CH4 bit (bit 11) in the IPR (see Section 3.3.1, “DSP56824 Interrupt Priority Register (IPR),” on page 3-14) must also be set to enable this interrupt. Table 9-3 lists the interrupt vectors for the three timers.

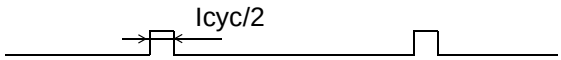
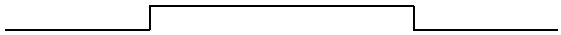
Table 9-3. Timer Interrupt Vectors

Timer	Interrupt Vector	Interrupt Priority
0	$0018	0
1	$001A	0
2	$001C	0

9.1.1.4 Timer Output Enable (TO[1:0])—Bits 11–10, Bits 3–2

The timer output enable (TO[1:0]) control bits (bits 3–2 in TCR01 for Timer 0, bits 11–10 in TCR01 for Timer 1, or bits 3–2 in TCR2 for Timer 2) are used to program the function of the timer output (TIO) pin. Table 9-4 shows the relationship between the value of the TO[1:0] bits and the function of the TIO pin. The TO bits are cleared on reset.

Table 9-4. TIO Pin Function

TO1	TO0	Function of TIO	Signal on TIO
0	0	TIO configured as input	
0	1	(Reserved)	
1	0	Overflow pulse mode, TIO configured as output	
1	1	Overflow toggle mode, TIO configured as output	

When the TO[1:0] bits are programmed for overflow pulse mode, the width of the pulse is the period of the internal Phi clock.

NOTE:

For the overflow pulse and overflow toggle modes, it is possible to have more than one timer enabled to drive the TIO pin. In this case, the TIO pin is pulsed or toggled when either counter reaches zero. In this way, it is possible with two timers to generate waveforms on the TIO pin with duty cycles other than 50 percent. Both timers are programmed with the same preload value, but their initial starting value differs. The amount by which their starting values differs determines the duty cycle of the resulting waveform.

It is also acceptable for all timers to be programmed such that no timer drives the TIO pin (TO[1:0] = 00 for the timers). The TIO pin is programmed in this manner when used as input.

NOTE:

If the overflow toggle mode is selected (TO[1:0] = 11) and the TE bit is written as zero while the TIO pin is either high or low, the TIO pin remains in the same state. If the TO1 bit or TO0 bit is written as zero with the TE bit equal to zero, the pin remains high and is driven low when the timer is reenabled.

9.1.1.5 Event Select (ES[1:0])—Bits 9–8, Bits 1–0

The event select (ES[1:0]) control bits (bits 1–0 in TCR01 for Timer 0, bits 9–8 in TCR01 for Timer 1, or bits 1–0 in TCR2 for Timer 2) select the source of the timer clock. If ES[1:0] is 11, an external signal coming from the TIO pin is used as input to the decrement register. The external signal is synchronized to the internal clock as described in Section 9.4, “Event Counting with the Timer Module.” The ES bits are cleared by reset. See Table 9-5 on page 9-7.

Table 9-5. ES[1:0] Bit Definition

ES[1:0]	Clock Selected
00	Internal Phi clock/4
01	Internal prescaler clock
10	Previous timer overflow
11	External event from TIO pin

NOTE:

For Timer 0, setting the ES[1:0] bits to 10 causes the clock source to be pulled low, and no signal is applied to the timer. This is because no previous timer is available.

When a timer is clocked using the slower clock from the prescaler divider, it can continue counting even when the DSP56800 core is in stop mode.

9.1.1.6 Reserved TCR Bits

Bits 14, 13, and 5 of TCR01 are reserved and are read as zero during read operations. These bits should be written with zero for future compatibility. Bits 15–8 and 5 of TCR2 are reserved and are read as zero during read operations. These bits should be written with zero for future compatibility.

9.1.2 Timer Preload Register (TPR)

The timer preload register (TPR) is a 16-bit write-only register that contains the value to be reloaded into the count register when a timer is enabled and when the (TCT) register has decremented to zero. Three preload registers are provided, one for each timer.

The timer must be disabled (its TE bit in TCR01 or TRC2 is cleared) when the user program writes a new value to its TPR. This new value transfers immediately into the TCT register (described in the following section) unless a direct write to the TCT register has already been performed.

NOTE:

The TPR can be written only when the corresponding timer is disabled (its TE bit cleared) in the TCR.

Because the TPR is write-only and cannot be read, reading its value can be accomplished by writing the TPR with the TE bit cleared and then reading the corresponding count register (with the TE bit cleared). The TPRs are initialized to zero on reset.

9.1.3 Timer Count Register (TCT)

The timer count (TCT) register is a 16-bit read/write register that contains the count for a timer. Three count registers are provided, one for each timer.

When a timer is enabled (its TE bit in the TCR01 or TRC2 is set), its TCT register is decremented by one with each clock. On the next event after the count register reaches zero, an overflow interrupt is generated if the OIE bit is set in the timer control register. Also, the state of the TIO pin can then be affected according to the mode selected by the TO[1:0] bits of the timer control register. If *n* is the value stored in

the count register when the timer is enabled, the overflow interrupt occurs after *n* + 1 input events. After reaching zero, the count register is reloaded with the contents of the preload register. The count register is loaded with a direct value when a direct write to the count register is executed.

The TCT register may also be read or written by a user program. When writing to the TCT register with the corresponding timer disabled, the value is immediately written to the count register and is not overwritten by the value stored in the preload register when the timer is enabled (its TE bit is set). The value stored or written in the preload register is loaded into the count register on the next event after it reaches zero, unless another write to the count register is performed in the meantime. Refer to Section 9.6, "Timer Module Timing Diagrams," for more details.

NOTE:

The count register can only be written when the corresponding timer is disabled (the TE bit is cleared) in the timer's control register.

The count register may be *read* only if one of the following conditions is true:

- The phase lock loop (PLL) is enabled (the PLL enable [PLLE] bit in the PLL Control Register 1 [PCR1] is set) and the prescaler clock is no more than half of the frequency of the Phi clock (that is, the frequency of the prescaler clock is not the same as the frequency of the Phi clock).
- The PLL is bypassed (the PLLE bit in the PCR1 is cleared) and the prescaler within the PLL is set to divide by 1 (the PS[2:0] bits in the PCR1 equal 000).
- The timer with the desired count register is disabled (the TE bit in the appropriate timer's control register is cleared).

See Section 10.2.1, "PLL Control Register 1 (PCR1)," on page 10-5 for information on the PCR1.

9.2 Timer Resolution

Table 9-6 shows the range of timer interrupt rates (overflow interrupt using the Phi clock) that are provided by the timer count register (TCR2–TCT0) and the timer preload register (TPR2–TCT0).

Table 9-6. Timer Range and Resolution

Input Clock Period	Timer Resolution (Preload = 0)	Timer Range (Preload = 2^{16} – 1)	Two Cascaded Timer Range
Phi clock = 14.29 ns (70 MHz)	57.14 ns	3.74 ms	245 s (approx. 4 minutes)
Phi clock = 25 ns (40 MHz)	100 ns	6.55 ms	429 s (approx. 7 minutes)
Phi clock = 50 ns (20 MHz)	200 ns	13.1 ms	859 s (approx. 14 minutes)
Phi clock = 100 ns (10 MHz)	400 ns	26.2 ms	1718 s (approx. 28 minutes)
Prescaler clock = 31 µs (32.00 kHz)	31.25 µs	2.0 s	134,218 s (approx. 37 hours)

The value stored in the TPR is the preload count. The overflow interrupt occurs every time the input clock provides a quantity of cycles equal to the preload count + 1.

9.3 Timer Interrupt Priorities

The timer module has a simple interrupt-priority scheme among its different timers, as shown in Table 9-7. It is important to service timer interrupts with higher priorities in a timely fashion to determine if any timer interrupts with a lower priority are also present.

Table 9-7. Timer Interrupt Priorities

Timer	Priority
0	Highest
1	Middle
2	Lowest

9.4 Event Counting with the Timer Module

This section contains examples of how to program some of the timing and counting capabilities of the general-purpose timers. This set of examples is by no means a complete list of either the capabilities or the programming techniques that can be used. Instead, it offers a representative range of what can be accomplished. The timer module can be used as an event counter. Setting the ES bits to 11 configures the timer to count the number of edges (external events) detected on the specified TIO pin. An external event is defined as a rising edge when the INV bit is cleared or as a falling edge when the INV bit is set. After the pin is conditionally inverted, it is synchronized to the Phi clock. Events on the TIO pin can be counted in wait mode, but cannot be counted in stop mode.

NOTE:

The maximum allowed frequency on the TIO pin is the frequency of the Phi clock divided by four.

Example 9-1 shows how to count events on a pin.

Example 9-1. Counting Events on a Pin

```
;*******************************************************
;* Example of Counting Events on a pin (with interrupt) *
;* for Timer Module                                     *
;* of DSP56824 chip                                     *
;*******************************************************
;
START                EQU     $0040      ; Start of program
BCR                  EQU     $FFF9      ; Bus Control Register
IPR                  EQU     $FFFB      ; Interrupt Priority Register
TCR01                EQU     $FFDF      ; Timer 0 & 1 Control Register
TCR2                 EQU     $FFDA      ; Timer 2 Control Register
TCT0                 EQU     $FFDD      ; Timer 0 Count Register
TPR0                 EQU     $FFDE      ; Timer 0 Preload Register
;**************
;* Vector setup*
;**************
;
       ORG           P:$0000            ;Cold Boot
       JMP           START              ;also Hardware RESET vector (Mode 0, 1, 3)
       ORG           P:$E000            ;Warm Boot
       JMP           START              ;Hardware RESET vector (Mode 2)
       ORG           P:$0018            ;
       JSR           TISR               ;Timer 0 Overflow vector
       ORG           P:START            ;Start of program
;**************
;* General setup*
;**************
;
       MOVEP         #$0000,X:BCR       ;External Program memory has 0 wait states.
                                        ;External data memory has 0 wait states.
                                        ;Port A pins tri-stated when no external
                                        ; access.
       BFCLR         #$0200,SR          ;Allow IPL (Interrupt Priority Level) 0
                                        ; -- Enable maskable interrupts.
                                        ; -- (peripherals, and so on)
;**********************
;* Timer Module setup*
;**********************
;
       MOVEP         #$0013,X:TCR01     ;Configure:
                                        ;Timer 0 & 1 disabled.
                                        ;Timer 0
                                        ; - don't Invert input: detect rising edges
                                        ; - Overflow Interrupt enabled
                                        ; - TIO pin configured as input
                                        ; - timer clock Event source is TIO pin
       MOVEP         #$0000,X:TCR2      ;Timer 2 disabled
       MOVEP         #234,X:TCT0        ;Set Timer 0 Count Register to 234--235 events
       MOVEP         #234,X:TPR0        ;Set Timer 0 Preload Register to 234.
       BFSET         #$0800,X:IPR       ;Enable timer module interrupts.
       ; ...
       BFSET         #$0080,X:TCR01     ;Enable Timer 0
;**************
;* Main routine*
;**************
;
TEST                                    ;Test Loop
                                        ; ...
       BRA    TEST
TISR                                    ;Timer Interrupt Service Routine
                                        ; interrupt code
        RTI
```

A timer can also be used to decrement to zero and generate an interrupt, as shown in Example 9-2.

Example 9-2. Decrementing to Zero and Generating an Interrupt

```
;*********************
;* INTERRUPT example *
;* for Timer Module  *
;* of DSP56824 chip  *
;*********************
START        EQU     $0040       ; Start of program
BCR          EQU     $FFF9       ; Bus Control Register
IPR          EQU     $FFFB       ; Interrupt Priority Register
PCR0         EQU     $FFF2       ; PLL Control Register 0
PCR1         EQU     $FFF3       ; PLL Control Register 1
TCR01        EQU     $FFDF       ; Timer 0 & 1 Control Register
TCR2         EQU     $FFDA       ; Timer 2 Control Register
TCT0         EQU     $FFDD       ; Timer 0 Count Register
TPR0         EQU     $FFDE       ; Timer 0 Preload Register
;*****************
;* Vector setup *
;*****************
       ORG    P:$0000             ; Cold Boot
       JMP    START               ; also Hardware RESET vector (Mode 0, 1, 3)
       ORG    P:$E000             ; Warm Boot
       JMP    START               ; Hardware RESET vector (Mode 2)
       ORG    P:$0018             ;
       JSR    TISR                ; Timer 0 Overflow vector
       ORG    P:START             ; Start of program
;******************
;* General setup *
;******************
       MOVEP  #$0000,X:BCR        ; External Program memory has 0 wait states.
                                  ; External data memory has 0 wait states.
                                  ; Port A pins tri-stated when no external access
       BFCLR  #$0200,SR           ; Allow IPL (Interrupt Priority Level) 0
                                  ; -- Enable maskable interrupts.
                                  ; -- (peripherals, and so on)
;***************************
;* PLL setup               *
;* (to increase Phi Clock) *
;***************************
       MOVEP  #$0180,X:PCR1       ; Configure:
                                  ; (PLLE) PLL disabled (bypassed)
                                  ; -- Oscillator supplies Phi Clock
                                  ; (PLLD) PLL Power Down disabled (PLL active)
                                  ; -- PLL block active for PLL to attain lock
                                  ; (LPST) Low Power Stop disabled
                                  ; (PS[2:0]) Prescaler Clock disabled
                                  ; Select Phi Clock for Clockout pin (CLKO)
       MOVEP  #$0260,X:PCR0       ; Set Feedback Divider to 1/20
                                  ; ...
                                  ; insert delay here: wait for PLL lock
                                  ; as specified in data sheet
                                  ; ...
       BFSET  #$4000,X:PCR1       ; Enable PLL for Phi Clock.
```

Example 9-2. Decrementing to Zero and Generating an Interrupt (Continued)

```
;**********************
;* Timer Module setup *
;**********************
;
       MOVEP  #$001C,X:TCR01      ; Configure:
                                  ; Timer 0 & 1 disabled
                                  ; Timer 0
                                  ; -- don't Invert TIO input: detect rising edges
                                  ;   (irrelevant but mentioned for completeness)
                                  ; -- Overflow Interrupt enabled
                                  ; -- Timer Output toggles TIO pin on overflow
                                  ; -- timer clock Event source is Phi Clock /4
       MOVEP  #$0000,X:TCR2       ; Timer 2 disabled
       MOVEP  #99,X:TCT0          ; Set Timer 0 Count Register to 99 (100 events).
       MOVEP  #99,X:TPR0          ; Set Timer 0 Preload Register to 99.
       BFSET  #$0800,X:IPR        ; Enable Timer Module interrupts.
                                  ; ...
       BFSET  #$0080,X:TCR01      ; Enable Timer 0
;****************
;* Main routine *
;****************
;
                                  ; ...
TEST                              ; Test Loop
       BRA    TEST
TISR                              ; Timer Interrupt Service Routine
                                  ; interrupt code
       RTI
```

Example 9-3 shows how to program Timer 0 and Timer 1 to generate a timing signal with a 25 percent duty cycle.

Example 9-3. Timer Using 25% Duty Cycle

```
;*************************
;* 25% duty cycle example *
;* for Timer Module*
;* of DSP56824 chip     *
;*************************
START        EQU     $0040       ; Start of program
BCR          EQU     $FFF9       ; Bus Control Register
IPR          EQU     $FFFB       ; Interrupt Priority Register
PCR0         EQU     $FFF2       ; PLL Control Register 0
PCR1         EQU     $FFF3       ; PLL Control Register 1
TCR01        EQU     $FFDF       ; Timer 0 & 1 Control Register
TCR2         EQU     $FFDA       ; Timer 2 Control Register
TCT0         EQU     $FFDD       ; Timer 0 Count Register
TCT1         EQU     $FFDB       ; Timer 1 Count Register
TPR0         EQU     $FFDE       ; Timer 0 Preload Register
TPR1         EQU     $FFDC       ; Timer 1 Preload Register
;*****************
;* Vector setup *
;*****************
;+-----------------------------------------------------------------------+
;| Note: Bootstrap ROM configures OMR (Operating Mode Register) to set   |
;|     chip operating mode for Mode 2 (Normal Expanded Mode), then       |
;|     jumps to first location of internal program RAM (P:$0000).        |
;+-----------------------------------------------------------------------+
       ORG    P:$0000             ; Cold Boot
       JMP    START               ; also Hardware RESET vector (Mode 0, 1, 3)
       ORG    P:$E000             ; Warm Boot
       JMP    START               ; Hardware RESET vector (Mode 2)
       ORG    P:START             ; Start of program
;*****************
;* General setup *
;*****************
       MOVEP   #$0000,X:BCR       ; External Program memory has 0 wait states.
                                  ; External data memory has 0 wait states.
                                  ; Port A pins tri-stated when no external access.
;***************************
;* PLL setup               *
;* (to increase Phi Clock) *
;***************************
       MOVEP  #$0180,X:PCR1       ; Configure:
                                  ; (PLLE) PLL disabled (bypassed)
                                  ; -- Oscillator supplies Phi Clock
                                  ; (PLLD) PLL Power Down disabled (PLL active)
                                  ; -- PLL block active for PLL to attain lock
                                  ; (LPST) Low Power Stop disabled
                                  ; (PS[2:0]) Prescaler Clock disabled
                                  ; Select Phi Clock for Clockout pin (CLKO).
       MOVEP  #$0260,X:PCR0       ; Set Feedback Divider to 1/20.
                                  ; ...
                                  ; insert delay here: wait for PLL lock
                                  ; as specified in data sheet
                                  ; ...
       BFSET  #$4000,X:PCR1       ; Enable PLL for Phi Clock.
```

Example 9-3. Timer Using 25% Duty Cycle (Continued)

```
;**********************
;* Timer Module setup *
;**********************
;
       MOVEP  #$0C0C,X:TCR01      ; Configure:
                                  ; Timer 0 & 1 disabled
                                  ; Timer 0 & 1
                                  ; -- don't Invert TIO input: detect rising edges
                                  ;   (irrelevant but mentioned for completeness)
                                  ; -- Overflow Interrupt disabled
                                  ; -- Timer Output toggles TIO pin on overflow
                                  ; -- timer clock Event source is Phi Clock /4
       MOVEP  #$0000,X:TCR2       ; Timer 2 disabled
                                  ; Setup Timer 0 & 1 for 25% duty cycle:
                                  ; high for first 25 of 100 events
       MOVEP  #00,X:TCT0          ; Set Timer 0 Count Register to 00.
       MOVEP  #99,X:TPR0          ; Set Timer 0 Preload Register to 99.
       MOVEP  #25,X:TCT1          ; Set Timer 1 Count Register to 25.
       MOVEP  #99,X:TPR1          ; Set Timer 1 Preload Register to 99.
       BFCLR  #$0800,X:IPR        ; Disable Timer Module interrupts.
                                  ; (superfluous but done for thoroughness)
                                  ; ...
       BFSET  #$8080,X:TCR01      ; Enable Timer 0 & 1
;****************
;* Main routine *
;****************
;
                                  ; ...
TEST                              ; Test Loop
       BRA    TEST
```

9.5 Timer Module Low-Power Operation

In applications requiring minimum power consumption, there are several options for lowering the power consumption of the chip using the timer module:

- Turn off the entire timer module.
- Turn off timers not in use.
- Lower the timer frequency.
- Run a timer in wait mode.
- Run a timer in stop mode.

The following sections discuss these options individually.

9.5.1 Turning Off the Entire Timer Module

If the timer module is not required by an application, it is possible to shut off the entire module for the lowest power consumption by clearing all the TE bits in TCR01 and TCR2 and setting all the ES[1:0] bits to 01 in these same registers. This provides the prescaler clock to the timers, which is the lowest power setting possible when using the ES bits.

If no other module on the DSP56824 is using the prescaler clock, it is possible to further reduce the overall power consumption by shutting down the prescaler divider that generates this clock. See Section 10.2.1.6, "Prescaler Divider (PS[2:0])—Bits 10–8," on page 10-6 for more information.

9.5.2 Turning Off Any Timer Not in Use

For applications not requiring all of the timers, it is possible to turn off any timer not in use. Individual timers are shut off by resetting the TE bit to zero for that individual timer and setting the ES[1:0] bits to 01 for that individual timer. These bits are located in register TCR01 or TCR2, depending on which timer is to be turned off.

9.5.3 Lowering the Timer Frequency

For applications requiring a timer to execute at a low frequency, less power is consumed if the timer is clocked using the prescaler clock instead of the Phi clock.

9.5.4 Running the Timer in Wait Mode

Overall power consumption on the DSP56824 can be reduced by using the wait mode of the DSP56800 core. It is possible to place the chip in wait mode for a predetermined amount of time. A timer is enabled and begins counting immediately before executing a WAIT instruction. When the timer reaches zero, a signal is generated that interrupts the DSP56800 core and brings it out of wait mode. All modes of operation in the timer module are available in wait mode.

9.5.5 Running the Timer in Stop Mode

Overall power consumption on the DSP56824 can be greatly reduced using the stop mode of the DSP56800 core. It is possible to place the chip in stop mode for a predetermined amount of time by setting up Timer 2 using the prescaler clock as input. This timer is enabled and begins counting. Then the DSP56800 core executes a STOP instruction. When Timer 2 reaches zero, a signal is generated that wakes up the DSP core and brings it out of stop mode.

NOTE:

The prescaler clock is the only clock available to the timer module that is active in stop mode. The TIO pin cannot be used as an event counter in stop mode. Also, only Timer 2 is capable of bringing the DSP56824 out of stop mode, and it performs this function independently of the bits in the IPR—see Section 3.3.1, “DSP56824 Interrupt Priority Register (IPR),” on page 3-14—or the value of the OIE control bit—see Section 9.1.1.3, “Overflow Interrupt Enable (OIE)—Bit 12, Bit 4.”

9.6 Timer Module Timing Diagrams

The figures in this section illustrate configurations in which the timer can be enabled, disabled, and used. Figure 9-4 on page 9-16 shows the standard timer operation, and Figure 9-5 on page 9-16 shows a write to the count register after writing the preload register when the timer is disabled.

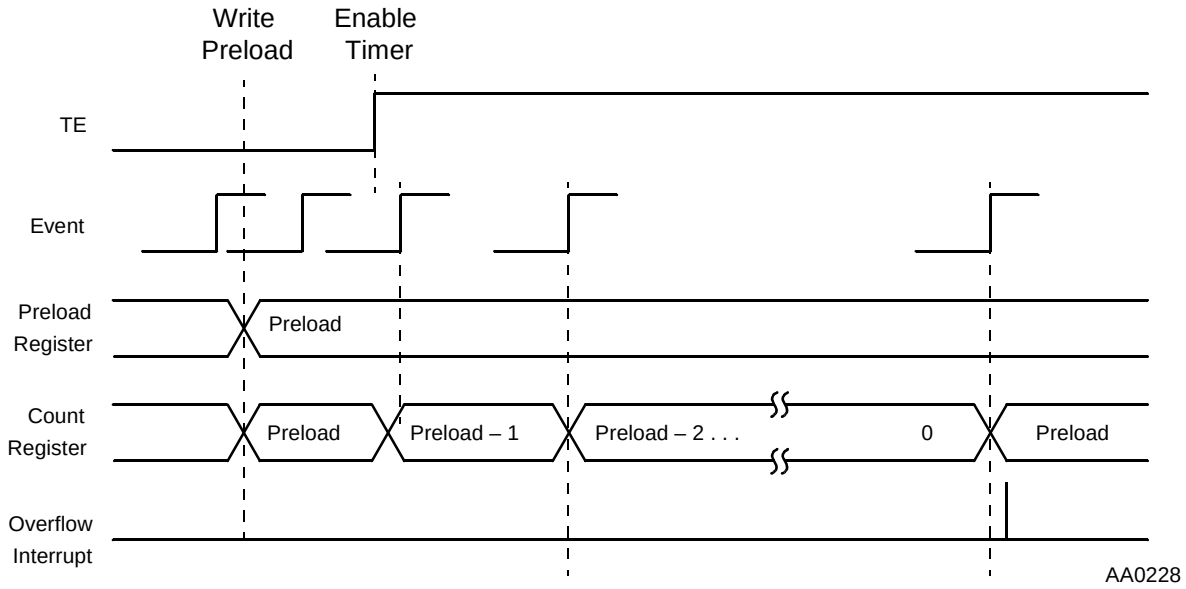


Figure 9-4. Standard Timer Operation with Overflow Interrupt

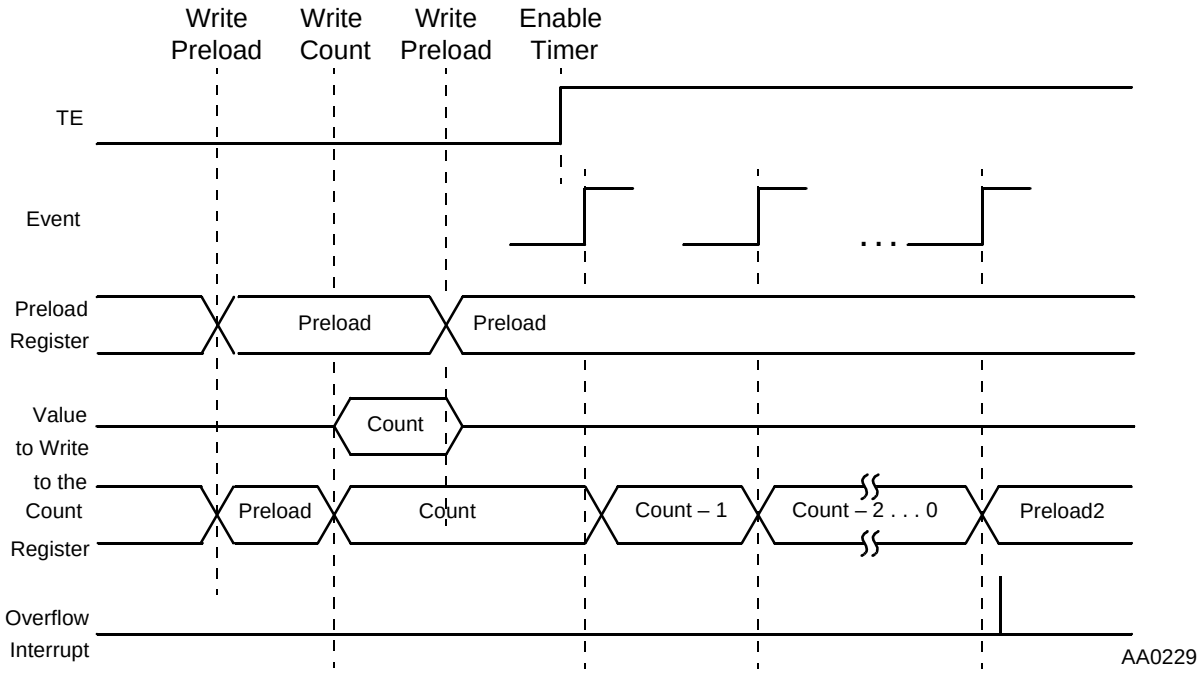


Figure 9-5. Write to the Count Register with Timer Disabled

9.7 Configuring Port C for Timer Functionality

The Port C control (PCC) register is used to individually configure each pin as either a timer pin or a general-purpose input/output (GPIO) pin. Setting the corresponding CC bit in the PCC register configures the pin as a timer pin. When the PCC register bit is set, it is not necessary to program the corresponding Port C data direction register (PCDDR) bit. The triple timer peripheral ensures the correct direction of this pin. Programming the PCDDR is necessary only when a pin is programmed as a GPIO pin.

Freescale Semiconductor, Inc.
ARCHIVED BY FREESCALE SEMICONDUCTOR, INC. 2005

Chapter 10
On-Chip Clock Synthesis

The clock synthesis module generates the clocking for the DSP56824. This section describes the module's architecture, programming model, and different low-power modes of operation. The module generates three clock signals for use by the DSP56800 core and DSP56824 peripherals. It also contains a phase lock loop (PLL) that can multiply the frequency, as well as a prescaler divider used to distribute lower-frequency clocks to peripherals, leading to lower power consumption on the chip. It also selects which clock, if any, is routed to the clock output (CLKO) pin of the DSP56824.

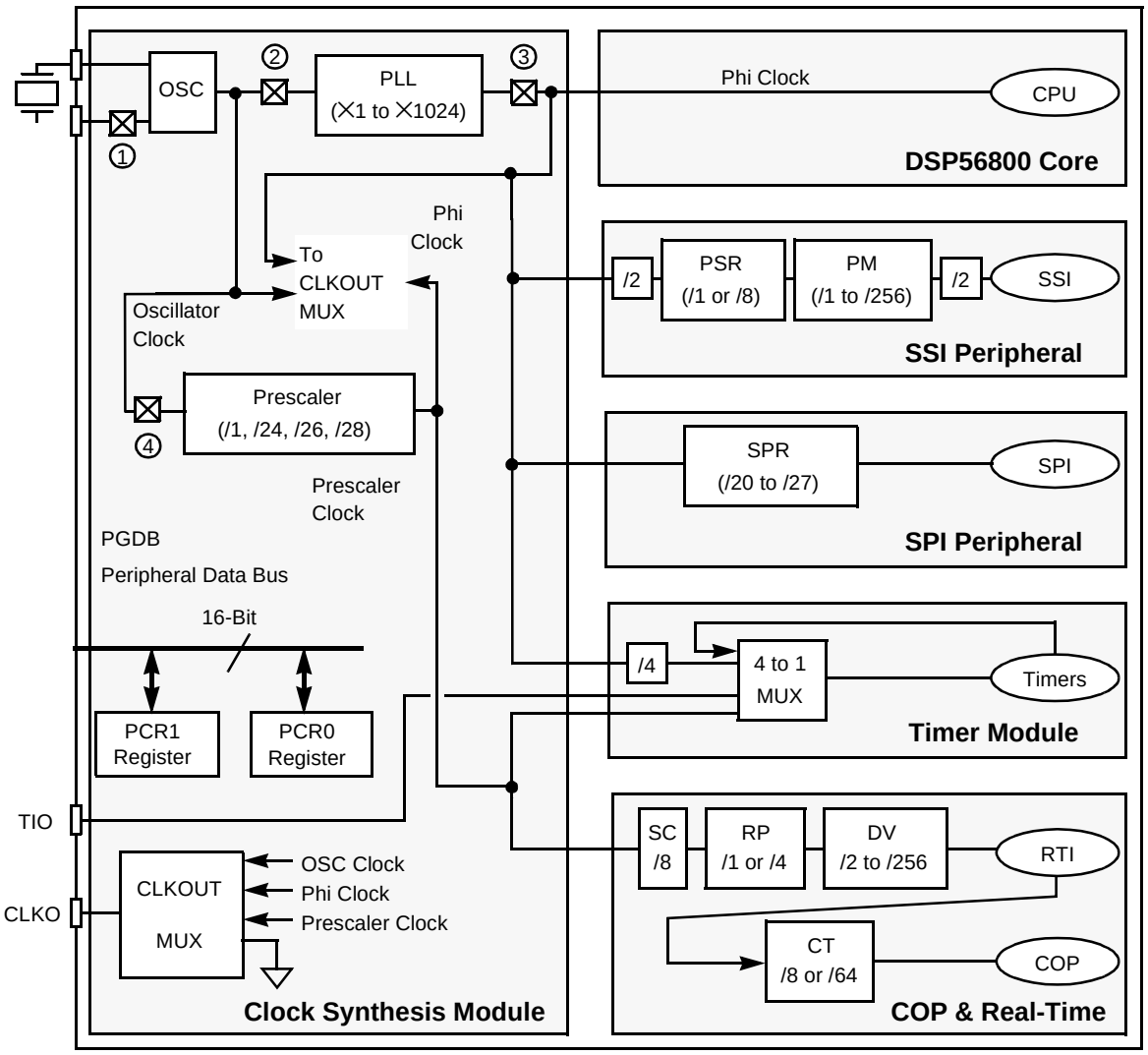
10.1 Timing System Architecture

The DSP56824 timing system, shown in Figure 10-1 on page 10-2, has the clock synthesis module at its core. This module is composed of the following five blocks:

- Oscillator
- Phase lock loop (PLL)
- Prescaler
- Clockout multiplexer (MUX)
- Control registers

Together, these five blocks generate the following three clock signals used for core and peripheral operation:

- Oscillator clock
- Phi clock
- Prescaler clock



① All clocks can be disabled at this point in stop mode by the LPST control bit.

② Clocks are disabled beyond this point in stop mode if the PLL is powered down.

③ Clocks are disabled beyond this point in stop mode.

④ Clocks beyond this point can be powered down by the PS[2:0] control bits.

AA0170

Figure 10-1. DSP56824 Timing System

Typically, the oscillator is attached to an external crystal. It can also be driven by an external oscillator. The output of the oscillator, called the oscillator clock signal, is provided to the prescaler, the PLL blocks, and the CLKOUT MUX.

The prescaler divides the oscillator clock signal and provides it to the Computer Operating Properly and real-time interrupt (COP/RTI) module, discussed in Chapter 11, “COP and RTI Module,” to the general-purpose timer module, and to the clockout MUX. This signal is called the prescaler clock.

The PLL multiplies up the oscillator clock signal and provides it to the DSP56800 core, to the SSI, to the SPI modules, to the general-purpose timer module, and to the clockout MUX. This multiplied signal is called the Phi clock.

The clockout MUX delivers one of these three signals (or none) to the CLKO pin.

The two control registers PCR0 and PCR1 control PLL multiplication, powering down, and MUX selects as well as other PLL-related options.

10.1.1 Oscillator

The DSP56824 supports frequencies from 32 kHz to the maximum specified frequency of the chip. The oscillator derives a clock signal from an external crystal. It is also possible to input an external clock directly to the EXTAL pin. In this case, no crystal is used and the XTAL pin can be left floating. The output of the oscillator drives the prescaler and the PLL inputs. This output also can provide an input for the clockout MUX. Detailed information and design guidelines are furnished in the *DSP56824 Technical Data Sheet.*

10.1.2 Phase Lock Loop (PLL)

The PLL is used to multiply up the oscillator clock frequency to the frequency needed by the core and peripherals for operation. For basic operation, the PCR1 is configured with the PLL power down (PLLD) bit cleared and the PLLE bit set to 1. When the PLLD bit is cleared, the PLL loop is powered on, and the oscillator clock is multiplied by the value of the bits in YD[9:0] + 1 at the voltage controlled oscillator (VCO). (The YD bits are contained in the PCR0.) When the PLL enable (PLLE) bit is set to one, the output of the VCO drives the Phi clock.

By setting the PLLE bit to zero, the PLL is bypassed and the Phi clock is driven directly by the oscillator clock. The bits in PCR0 and PCR1 are described in Section 10.2, “Clock Synthesis Programming Model.” Figure 10-2 shows a block diagram of the PLL.

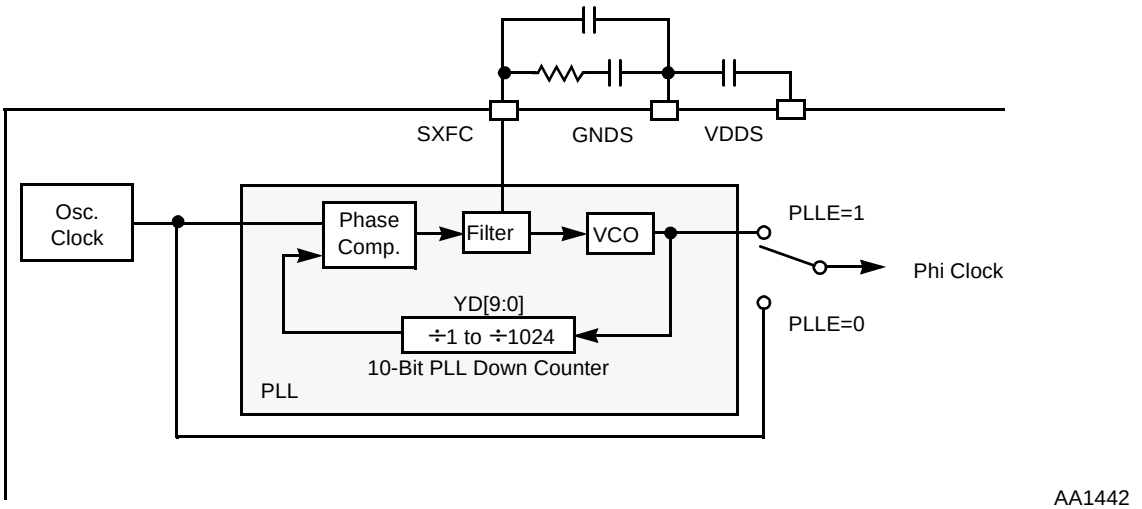


Figure 10-2. PLL Block Diagram

10.1.3 Prescaler

The prescaler is used when a higher frequency input clock or crystal (1 MHz or more) is required in an application. The prescaler provides a slower clock signal as input to the RTI and COP timer. In addition, this low-frequency clock signal can be used as input to the timers in the timer module. When a 32 kHz or 38.4 kHz crystal is used with the DSP56824, it is appropriate to set the prescaler to divide by one (effectively bypassing the prescaler).

The prescaler clock is generated by a divider clocked on the oscillator output. The following divide ratios are supported: /1, /16, /64, and /256. The prescaler can also be disabled. The divide ratio is determined by the value of the PS[2:0] bits in the PCR1. See Section 10.2.1.6, “Prescaler Divider (PS[2:0])—Bits 10–8,” for detailed information on PS bit values and corresponding divide rates.

NOTE:

The maximum frequency of the prescaler clock is limited to one-sixteenth of the maximum clocking frequency of the part. This means that for a 70 MHz DSP56824, the clocking frequency of the prescaler clock must be less than or equal to 4.375 MHz.

10.1.4 Clockout Multiplexer (MUX)

The clockout multiplexer (MUX) selects which clock is provided to the CLKO pin. Disable this pin for lowest power operation using the control bits in PCR1. For testing and some user applications, it is desirable to provide the Phi clock on this pin. In some cases, it may be desirable to provide the oscillator clock on the CLKO pin.

10.1.5 Control Registers

The PCR0 and PCR1 control registers provide the majority of the user control over the clock synthesis module. Individual bits and their functions are described in Section 10.2, “Clock Synthesis Programming Model.”

10.2 Clock Synthesis Programming Model

The clock synthesis module provides two control registers to manage the frequency, signal paths, and outputs of the DSP56824 clocks:

- PCR1—PLL control register 1
- PCR0—PLL control register 0

Clock signal control is also provided by additional registers within the following peripherals:

- Synchronous serial interface (SSI)
- Serial peripheral interface (SPI)
- COP/RTI

NOTE:

Clock signals used within the peripherals can be provided as clock outputs. Users should be reminded that changing a clock may change the behavior of another peripheral.

The clock synthesis programming model is shown in Figure 10-3 on page 10-5.

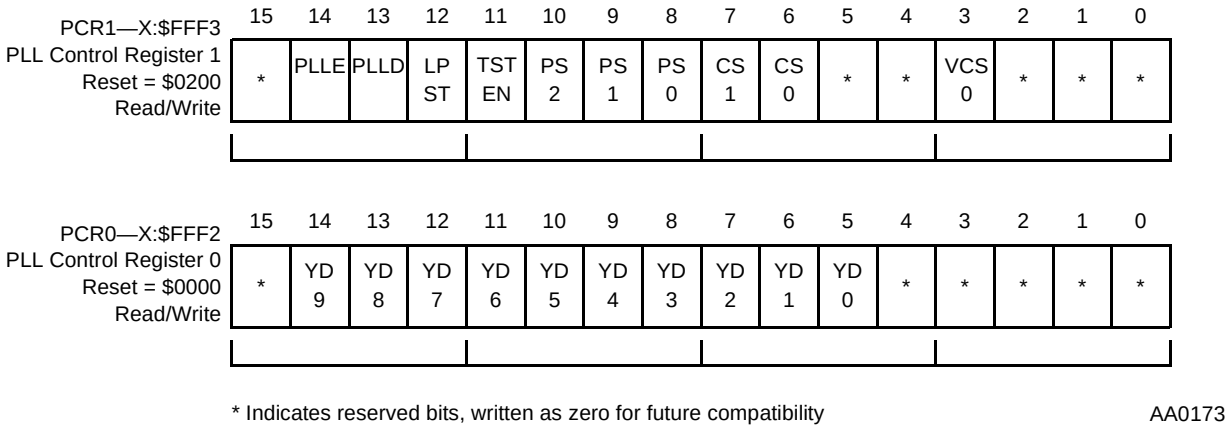


Figure 10-3. Clock Synthesis Programming Model

10.2.1 PLL Control Register 1 (PCR1)

The PLL control register 1 (PCR1) is a 16-bit read/write register used to direct the operation of the on-chip clock synthesizer. The PCR1 control bits are defined in Section 10.2.1.1, “Reserved Bit—Bit 15,” through Section 10.2.1.10, “Reserved Bits—Bits 2–0.”

10.2.1.1 Reserved Bit—Bit 15

Bit 15 is reserved and is read as zero during read operations. This bit should be written with zero to ensure future compatibility.

10.2.1.2 PLL Enable (PLLE)—Bit 14

The PLL enable (PLLE) control bit and the PLLD control bit (bit 13) interact to control PLL operation. When PLLE is set, the DSP56824 system clock (Phi clock) is generated by the on-chip PLL, using the YD bits to select the PLL multiplication factor (MF).

The state of the PLL is defined by the PLLD control bit (bit 13). The interaction of PLLE and PLLD is shown in Table 10-1 on page 10-6.

10.2.1.3 PLL Power Down (PLLD)—Bit 13

The state of the PLL is defined by the PLL power down (PLLD) control bit (bit 13). The PLLE and PLLD bits work together to control PLL operation, as shown in Table 10-1 on page 10-6. When the PLLE bit is set, the DSP56824 system clock (Phi clock) is generated by the on-chip PLL.

When the PLLD bit is set, the PLL is in the power down mode, a low-current mode in which the VCO is inactive. When the PLLD bit is cleared, the PLL is in the active mode. Before turning the PLL off, clear the PLLE bit to bypass the PLL. Then put the PLL in power down mode by setting the PLLD bit. Setting the PLLD bit powers down the complete PLL block, including the PS and YD registers, described in Section 10.2.1.6, “Prescaler Divider (PS[2:0])—Bits 10–8,” and Section 10.2.2.2, “Feedback Divider (YD[9:0])—Bits 14–5,” respectively.

The PLLD bit should not be set when the PLLE bit is set. Both the PLLD bit and the PLLE bit are cleared when the chip is reset.

NOTE:

The STOP instruction does not power down the PLL if the PLL is not powered down (PLLD = 0) when entering stop mode.

Table 10-1. PLL Operations

PLLE	PLLD	Phi Clock	PLL Mode
0	0	Oscillator clock	Active
0	1	Oscillator clock	Power down
1	0	Oscillator clock × [YD + 1]	Active
1	1	(Reserved)	(Reserved)

10.2.1.4 Low Power Stop (LPST)—Bit 12

The low power stop (LPST) control bit is used to place the chip in the lowest power configuration when entering stop mode. If the LPST bit is set when entering stop mode, the clock is disabled at the crystal oscillator. If the LPST bit is cleared when entering stop mode, the oscillator continues running in stop mode. The LPST bit is cleared on DSP reset.

10.2.1.5 Test Enable (TSTEN)—Bit 11

The test enable (TSTEN) control bit allows the prescaler output to drive the CLKO pin when set in conjunction with the CS[1:0] bits, as shown in Table 10-3 on page 10-7. The TSTEN bit is cleared on DSP reset.

10.2.1.6 Prescaler Divider (PS[2:0])—Bits 10–8

The prescaler divider (PS[2:0]) control bits are used to pass, disable, or divide the oscillator clock by several different divide ratios: 16, 64, and 256. The output of the divider can be used as the operating clock for the timer module or for the COP and real-time timers. On reset, the PS[2:0] bits are set to 010, providing a divide-by-16 prescaler rate. This ensures that implementations using a high-speed clock will function properly, because the general-purpose and COP timers are not designed to work at frequencies higher than one-sixteenth of the maximum frequency of the DSP56824.

The prescaler divider is used for systems with higher frequency crystals to provide a slower clocking frequency near 32 kHz for the RTI and COP timers discussed in detail in **Chapter 11, “COP and RTI Module.”** Typically a user would set the divider to divide by one for systems with a 32.0 kHz or 38.4 kHz crystal, but would use the divider when a higher frequency crystal is used. Likewise, it is possible to disable this divider and its output clock for low-power applications that do not require a real-time or COP timer and do not require a low-frequency clock for the timer module.

The prescaler should always be set up with the correct division ratio before any peripheral using the prescaler clock is enabled. Table 10-2 on page 10-7 shows how to program the PS[2:0] bits.

Table 10-2. PS Divider Programming

PS[2:0]	Function	Comments
000	Divide by 1	Used for 32.0 kHz and 38.4 kHz crystals
001	Disabled	For low-power applications not requiring real-time or COP timers
010	Divide by 16	Reset value; also, typically used for higher frequency crystals
011	(Reserved)	(Reserved)
100	Divide by 64	Typically for higher frequency crystals
101	(Reserved)	(Reserved)
110	Divide by 256	Typically for higher frequency crystals
111	(Reserved)	(Reserved)

NOTE:

The maximum frequency of the prescaler clock is limited to one-sixteenth of the maximum clocking frequency of the part. This means that for a 70 MHz DSP56824, the clocking frequency of the prescaler clock must be less than or equal to 4.375 MHz.

Changing the prescaler control bits when the COP timer is enabled via the CPE bit (in the COPCTL register) does *not* result in a change of the prescaler divider. This prevents an application from accidentally disabling the COP timer by disabling the prescaler clock. If the COP enable (CPE) bit is set, then the PS bits can still be written, but the prescaler division ratio is not changed. See Section 11.2.1, “COP and RTI Control Register (COPCTL),” on page 11-4 for a description of the COPCTL register.

NOTE:

There is a restriction when setting the prescaler’s division ratio. The case where the prescaler is set to divide by one *and* the PLL is set for an MF of one when the PLL provides the Phi clock (PLLE = 1) is *not allowed*. Violating this restriction results in faulty prescaler clock synchronization in the peripherals.

10.2.1.7 CLKO Select (CS[1:0])—Bits 7–6

When programmed in conjunction with the TSTEN bit, the CLKO select (CS[1:0]) control bits are used to enable one of three different clocks to the CLKO pin or to disable all clocks to this pin. After DSP reset, the Phi clock output is provided on the CLKO pin. The other options are presented in Table 10-3. The CS[1:0] bits are cleared on DSP reset.

Table 10-3. CLKOUT Pin Control

TSTEN	CS[1:0]	CS0	CLKO
0	0	0	Phi clock
0	0	1	(Reserved)
0	1	0	Oscillator clock

Table 10-3. CLKOUT Pin Control (Continued)

TSTEN	CS[1:0]	CS0	CLKO
0	1	1	Disabled
1	0	0	(Reserved)
1	0	1	(Reserved)
1	1	0	(Reserved)
1	1	1	Prescaler output

10.2.1.8 Reserved Bits—Bits 5–4

Bits 5–4 are reserved and are read as zero during read operations. These bits should be written with zero to ensure future compatibility.

10.2.1.9 VCO Curve Select (VCS0)—Bit 3

The VCO curve select 0 (VCS0) control bit optimizes the VCO for the desired frequency range to provide better lock time and stability, as shown in Table 10-4. The VCS0 bit is cleared on DSP reset.

Table 10-4. VCS0 Programming

VCS0	PLL Output Frequency
0	40 MHz to 70 MHz
1	10 MHz to 40 MHz

10.2.1.10 Reserved Bits—Bits 2–0

Bits 2–0 are reserved and are read as zero during read operations. These bits should be written with zero to ensure future compatibility.

10.2.2 PLL Control Register 0 (PCR0)

The PLL control register 0 (PCR0) is a 16-bit read/write register used to direct the operation of the on-chip clock synthesis. The PCR0 controls the frequency programming of the PLL. The PCR0 control bits are defined in Section 10.2.2.1, “Reserved Bit—Bit 15,” through Section 10.2.2.3, “Reserved Bits—Bits 4–0.” All bits of PCR0 are cleared by DSP hardware reset.

10.2.2.1 Reserved Bit—Bit 15

Bit 15 is reserved and is read as zero during read operations. This bit should be written with zero to ensure future compatibility.

10.2.2.2 Feedback Divider (YD[9:0])—Bits 14–5

The feedback divider (YD[9:0]) bits control the down counter in the feedback loop, causing it to divide by the value YD + 1, where YD is the value contained in the YD[9:0] bits. The YD[9:0] bits are cleared on DSP reset.

The resulting Phi clock signal must be within the limits specified in the “Specifications” section of the *DSP56824 Technical Data Sheet*. The frequency of the VCO should also remain higher than the specified minimum value. Recommended PLL MFs for the DSP56824 are also provided in the *DSP56824 Technical Data Sheet*.

10.2.2.3 Reserved Bits—Bits 4–0

Bits 4–0 are reserved and are read as zero during read operations. These bits should be written with zero to ensure future compatibility.

10.3 Low-Power Wait and Stop Modes

The DSP56824 can be configured for very low-power consumption in wait mode or minimal power consumption in stop mode, based on application needs. In wait mode, all internal core clocks are gated off, but the Phi clock continues to operate so that peripherals continue to function and can bring the chip out of wait mode by using a peripheral interrupt. In stop mode, the Phi clock and all internal core clocks are gated off. Stop mode uses less power than wait mode. Before entering either stop or wait mode, it is possible to enable or disable the following blocks individually:

- CLKO pin
- PLL module
- COP/RTI module
- Timer module
- SPI module
- SSI module

In addition, it is also possible to disable the prescaler block, but this in turn disables the COP/RTI and the timer module blocks. All blocks left enabled continue to run in stop mode, except the SPI and SSI. The timer module and COP/RTI can bring the chip out of stop mode. The SPI and SSI are always powered down in stop mode.

Several options for the clock synthesis block are available to the user in stop mode.

- Leave PLL enabled—low power, short wake-up time since PLL already stabilized
- Disable PLL—lower power, long wake-up time for PLL to stabilize
- Leave prescaler enabled—allows COP and real-time timers to continue operating
- Disable prescaler—lower power, disables clock to COP and real-time timers
- Disable oscillator—lowest power, all internal clocks disabled, longest wake-up time

Leaving the PLL enabled in stop or wait mode avoids the need to wait for the PLL to relock upon exiting. Examples of low-power configurations in stop mode follow in Section 10.3.1, “PLL, COP, Real-Time Clock, Timers, and CLKO Enabled,” through Section 10.3.5, “Everything Disabled.”

10.3.1 PLL, COP, Real-Time Clock, Timers, and CLKO Enabled

In this stop mode, the DSP56800 core and the SPI and SSI peripherals are placed in low-power stop mode, in which all clocks are gated off. The PLL remains running and locked, the COP and real-time clock timers can still continue functioning, and a clock waveform can be provided on the CLKO pin, if desired. In this mode the oscillator is still running.

NOTE:

It is also possible to leave the timer module running if clocked with the prescaler clock. This stop mode consumes the most power.

This mode is entered by executing a STOP instruction with the PLL, COP/RTI, and timer module peripherals still enabled. The timer module must be clocked with the prescaler clock.

NOTE:

The CLKO pin can be disabled independently using the CS[1:0] control bits in the PCR1, described in Section 10.2.1.7, “CLKO Select (CS[1:0])—Bits 7–6.”

10.3.2 COP, Realtime Clock and CLKO Pin Enabled

In this stop mode, the DSP56800 core and the PLL, SPI, SSI, and timer-module peripherals are placed in low-power stop mode where all clocks are gated off. The PLL loses lock in this condition. The COP and real-time clock timers can still continue functioning, and a clock waveform can be provided on the CLKO pin, if desired. In this mode the oscillator is still running.

This mode is entered by executing a STOP instruction with the PLL disabled and the COP/RTI peripheral still enabled. The CLKO pin can also be disabled independently using the CS[1:0] control bits in the PCR1.

10.3.3 PLL and CLKO Pin Enabled

In this stop mode, the DSP56800 core and the COP/RTI, SPI, SSI, and timer-module peripherals are placed in low-power stop mode where all clocks are gated off. The PLL remains locked and running in this condition. A clock waveform can be provided on the CLKO pin, if desired. In this mode the oscillator is still running.

This mode is entered by executing a STOP instruction with the PLL enabled and the COP/RTI peripheral disabled. The CLKO pin can also be disabled independently using the CS[1:0] control bits in the PCR1.

10.3.4 CLKO Pin Enabled

In this stop mode, the DSP56800 core and the COP/RTI, PLL, SPI, SSI, and timer-module peripherals are placed in low-power stop mode where all clocks are gated off. The PLL loses lock in this condition. A clock waveform is provided on the CLKO pin. In this mode the oscillator is still running. It may be necessary to wait for the PLL to relock after exiting stop mode.

This mode is entered by executing a STOP instruction when the PLL and COP/RTI peripherals are both disabled and the CLKO pin is enabled.

10.3.5 Everything Disabled

In this stop mode, the DSP56800 core and the COP/RTI, PLL, SPI, SSI, and timer-module peripherals are placed in low-power stop mode where all clocks are gated off. The PLL loses lock in this condition. The CLKO pin is disabled and the oscillator is disabled as well. If an external crystal is used, the processor must wait for the crystal to stabilize before exiting stop mode. It may also be necessary to wait for the PLL to relock.

This mode is entered by executing a STOP instruction when the LPST bit in the PCR1 is set. This is the lowest power stop mode available.

10.4 PLL Lock

There are several conditions when it is necessary to wait for the PLL to lock:

- When the chip first powers up
- When the PLL is taken out of its power-down state (clearing the PLLD bit when it had previously been set)
- When the frequency of the PLL is changed by modifying the YD bits in the PCR0

NOTE:

Changing the PLL frequency may require changing external filter components and is not recommended. See the *DSP56824 Technical Data Sheet* for more information.

In each of these cases, it is necessary to wait until the PLL locks on its final frequency before the PLL clock is sent to the DSP56800 core and the peripherals (PLLE = 1). PLL lock time is provided in the *DSP56824 Technical Data Sheet*. Failure to wait until the PLL locks can result in improper processing states, software errors, and other problems.

10.4.1 PLL Programming Example

Example 10-1 on page 10-12 provides an example of how to program the PLL to multiply by a factor of 20 and wait for the PLL to stabilize.

Example 10-1. Programming the PLL

```
;********************
;* PLL Setup example*
;* of DSP56824 chip *
;********************
;
PCR0          EQU      $FFF2      ; PLL Control Register 0
PCR1          EQU      $FFF3      ; PLL Control Register 1
;********************
;
;* PLL setup        *
;* (to increase Phi Clock) *
;********************
;
      MOVEP   #$0180,X:PCR1       ; Configure:
                                  ; (PLLE) PLL disabled (bypassed)
                                  ; -- Oscillator supplies Phi Clock
                                  ; (PLLD) PLL Power Down disabled (PLL active)
                                  ; -- PLL block active for PLL to attain lock
                                  ; (LPST) Low Power Stop disabled
                                  ; (PS[2:0]) prescaler clock disabled
                                  ; (CS[1:0]) Clockout pin sends Oscillator Clock
      MOVEP  #$0260,X:PCR0        ; Set Feedback Divider to 1/20
                                  ; ...
                                  ; insert delay here: wait for PLL lock
                                  ; as specified in data sheet
                                  ; ...
      BFSET  #$4000,X:PCR1        ; Enable PLL for Phi Clock
```

10.4.2 Changing the PLL Frequency

To change the output frequency of the PLL (by reprogramming the YD bits) while the PLL output is used by the DSP56800 core (PLLE = 1; PLLD = 0), perform the following sequence of operations:

1. Clear the PLLE bit to switch back to the oscillator clock.
2. Program the YD bits (only after clearing PLLE).
3. Wait until the PLL has locked. See the *DSP56824 Technical Data Sheet* for settling time specifications.
4. Set the PLLE bit.

NOTE:

Changing the PLL frequency may require changing external filter components and is not recommended. See the *DSP56824 Technical Data Sheet* for more information on supported PLL frequencies and recommended external filter component values.

10.4.3 Turning Off the PLL Before Entering Stop Mode

To turn off the PLL before entering stop mode, execute the following sequence before issuing the STOP instruction:

1. Clear the PLLE bit to switch back to the oscillator clock.
2. Set the PLLD bit to one to power down the PLL.
3. Execute the STOP instruction.

NOTE:

The PLL can be left running when entering stop mode. This allows a faster exit to normal mode. There is no wait for PLL lock because the PLL continues to run in stop mode.

10.5 PLL Module Low-Power Operation

In applications requiring minimum power consumption, there are several options for lowering the power consumption of the chip within the clock synthesis module in stop or wait mode. These are discussed individually below.

10.5.1 Turning Off the Entire Clock Synthesis Module

If no internal clocks are required by an application in stop mode, shut off the entire module for lowest power consumption. This is done by resetting the PLLD bit to 1 (PLLE must already be cleared), setting the PS[2:0] bits (in the PCR1) to 001, setting the CS[1:0] bits (in the PCR1) to 11, and setting the LPST bit (in the PCR1) to 1. See Section 10.2.1, “PLL Control Register 1 (PCR1),” for details on the PCR1.

When the LPST bit is set to 1, an additional period of time is required for crystal oscillator stabilization. Upon exiting stop mode, it is necessary to wait for the PLL to stabilize again (relock). Both these times are specified in the *DSP56824 Technical Data Sheet*.

10.5.2 Turning Off the Prescaler Divider When Not in Use

For applications not requiring a prescaler clock to any peripherals, turn off the prescaler divider by setting the PS[2:0] bits in PCR1 to 001.

NOTE:

The prescaler divider can be turned off independently from the PLL or CLKO pin.

10.5.3 Turning Off the PLL When Not in Use

For applications in which the time required for the PLL to relock when exiting stop mode is not an issue, the PLL can be turned off by setting the PLLD bit in the PCR1 to one. (See Section 10.2.1, “PLL Control Register 1 (PCR1),” on page 10-5.) This can be done only after the PLLE bit in PCR1 has been cleared.

NOTE:

The PLL can be turned off independently from the prescaler divider or CLKO pin. Upon exiting stop mode, the PLL must be reenabled and it is necessary to wait for the PLL to stabilize again (relock).

10.5.4 Turning Off the CLKO Pin When Not in Use

For applications where no external clockout pin is required, it is recommended to turn off the CLKO pin to lower power and reduce switching on the pins. This can be accomplished by setting the CS[1:0] bits to 11. See Section 10.2.1, “PLL Control Register 1 (PCR1).”

NOTE:

The CLKO pin can be turned off independently from the PLL or prescaler divider.

Chapter 11
COP and RTI Module

This section describes the Computer Operating Properly (COP) and real-time interrupt (RTI) module (COP/RTI) provided on the DSP56824.

The COP/RTI module provides two separate functions: a watchdog-like timer and a periodic interrupt generator. The COP timer guards processor activity and provides an automatic reset signal if a failure occurs. Both functions are contained in the same block because the input clock for both comes from a common clock divider.

11.1 COP and Real-Time Timer Architecture

The COP timer protects against system failures by providing a means to escape from unexpected input conditions, external events, or programming errors. Once started, the COP timer must be reset by software on a regular basis so that it never reaches its time-out value. When the COP timer reaches its time-out value, an internal reset is generated within the chip and the COP reset vector is fetched. It is assumed that if the COP reset is not received from the program, a system failure has occurred.

The COP functionality is typically used by software to ensure that the chip is operating properly. Software must periodically service the COP timer by correctly writing to the COP reset (COPRST) register before the COP timer times out. The COP timer has its own reset vector. This allows the reset recovery from a COP time-out to differ from the reset procedure done after a hardware reset.

A COP reset is very similar to a hardware reset through the $\overline{\text{RESET}}$ pin. The minor differences are the address from which the reset vector is fetched and the length of the period during which reset is asserted.

The RTI capability provides a periodic interrupt in an application. It consists of a long decrementing counter chain that runs continuously when enabled. When it reaches zero, a status flag is set and an interrupt is generated (optionally, when enabled by the user). The RTI has its own interrupt vector location to reduce overhead in interrupt servicing.

Figure 11-1 on page 11-2 shows a block diagram of the COP/RTI timer module. This module contains a programmable divider chain for dividing the prescaler clock to a periodic rate that can be used as an RTI. This real-time clock is further divided down to get a COP timer reset. The COP and RTI control (COPCTL) register is used to set up the peripheral and program the divide ratios.

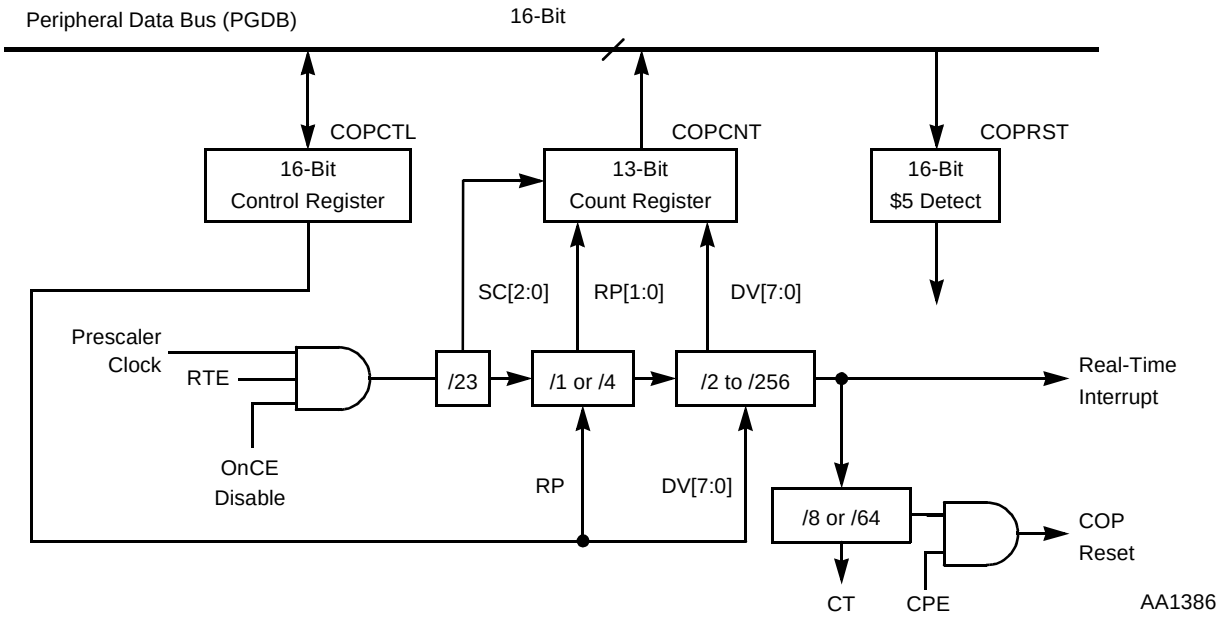


Figure 11-1. RTI and COP Timer Block Diagram

NOTE:

The maximum frequency of the prescaler clock is limited to one-sixteenth of the maximum clocking frequency of the part. This means that for a 70 MHz DSP56824, the clocking frequency of the prescaler clock must be less than or equal to 4.375 MHz.

The RTI timer and the COP timer share the scaler (SC), real-time prescaler (RP), and COP/RTI divider (DV) dividers in the clock divider chain. The current value of the RTI timer can be determined at any time by reading the COP/RTI count (COPCNT) register, the bits of which reflect the shared components of the COP/RTI divider chain. These shared components comprise the RTI timer.

The COP timer is a timer whose clock source is cascaded from the RTI timer. It consists of the COP timer (CT) divider. The COP timer counts from either 7 or 63 down to 0, and then provides the COP reset signal, which resets the DSP chip. The COP timer cannot be read. Its only function is to count down to 0 and send a COP reset signal, unless the COP timer itself is reset. The COP reset sequence, provided in Example 11-1 on page 11-9, must be programmed to run periodically. Sending this reset sequence is analogous to renewing a library book before it is due.

The COPCTL register reflects the control status of both the RTI timer and the COP timer. The COP/RTI peripheral is enabled by the RTI enable (RTE) bit in the COPCTL register. In addition, the On-Chip Emulation (OnCE™) module within the DSP56800 core can disable the counting in the RTI and COP timers to prevent counting when the chip is no longer executing instructions, but instead is in debug mode. This is useful for debugging real-time systems.

When the COP timer expires and a COP reset occurs, the following sequence takes place:

1. DSP reset occurs.
2. The original MODA and MODB values that were captured on $\overline{\text{RESET}}$ deassertion are reloaded into the MA and MB bits of the OMR.
3. Program control is transferred to the appropriate COP reset vector determined by the values in MA and MB.

MODA, MODB, and $\overline{\text{XCOLF}}$ are not resampled on COP reset.

For example, if Mode 2 was entered on $\overline{\text{RESET}}$ deassertion and the OMR was later altered to select Mode 1, an ensuing COP reset would change program flow to reflect the change to Mode 1. In this case, the reset vector at P:$E002 would be used.

11.2 COP and RTI Timer Programming Model

The COP/RTI block contains the following registers:

- COP/RTI control (COPCTL) register
- COP/RTI count (COPCNT) register
- COP reset (COPRST) register

These three registers are shown in Figure 11-2 and explained in Section 11.2.1, “COP and RTI Control Register (COPCTL),” through Section 11.2.3, “COP Reset (COPRST) Register.”

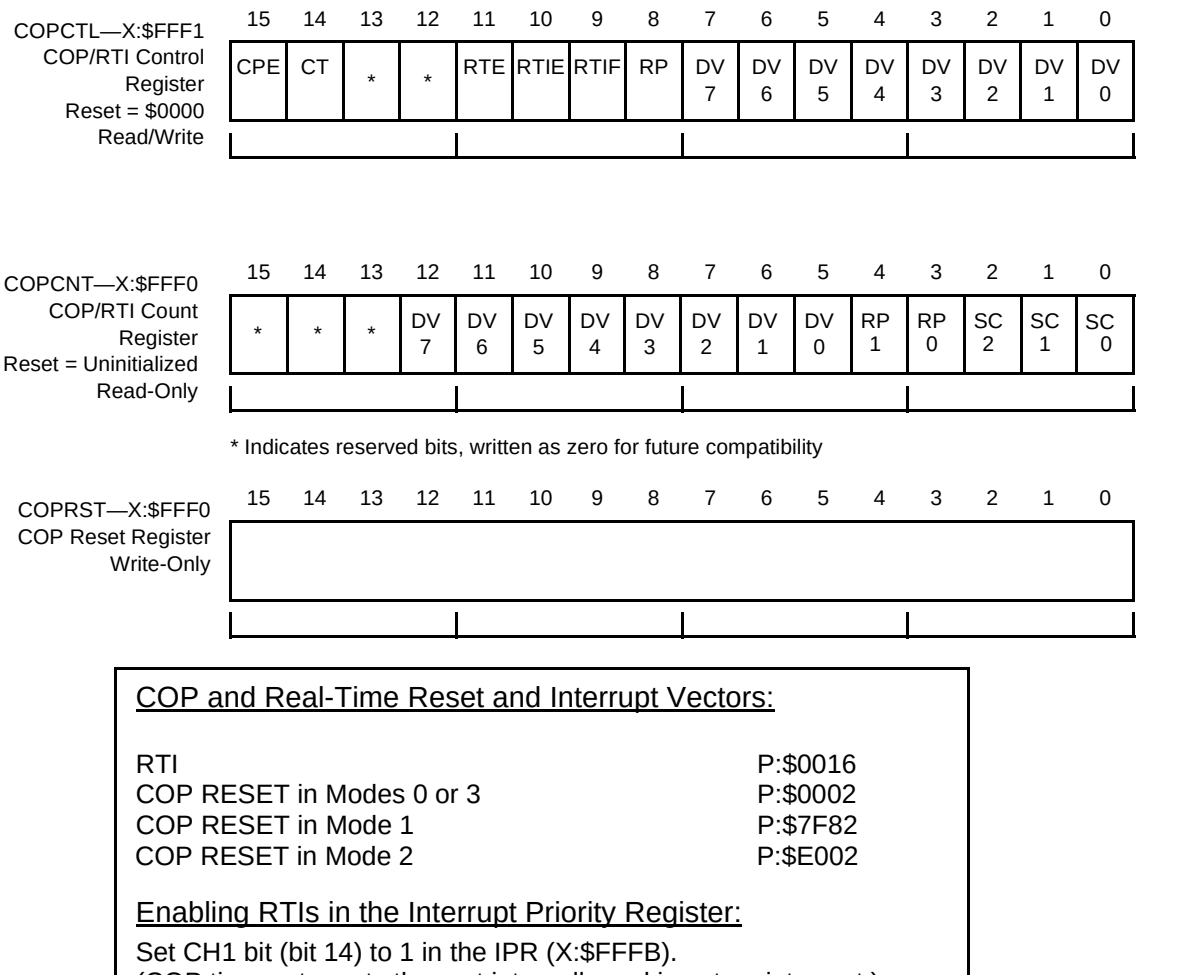


Figure 11-2. RTI and COP Timer Programming Model

11.2.1 COP and RTI Control Register (COPCTL)

The COP and RTI Control (COPCTL) register is a 16-bit read/write register used to program both the COP timer and the RTI timer. The COPCTL register is reset to $0000 on hardware reset. When changing the bits in this register, it is important to follow the guidelines in Section 11.3, “Programming the COP and RTI Timers.” The bits of this register are defined in Section 11.2.1.1, “COP Enable (CPE)—Bit 15,” through Section 11.2.1.8, “RTI/COP Divider (DV[7:0])—Bits 7–0.”

NOTE:

The COPCTL register cannot be written unless preceded by the sequence documented in Section 11.3, “Programming the COP and RTI Timers.” This ensures that the COPCTL register cannot be modified if the computer is not operating properly. The COPCTL register can be read at any time.

11.2.1.1 COP Enable (CPE)—Bit 15

The COP enable (CPE) control bit enables the COP timer functionality. Both the RTE bit and the CPE bit must be set for the COP timer to function. The CPE bit is cleared on hardware reset.

NOTE:

When set, the COP timer disable (COPDIS) bit in the OnCE control register (OCR) overrides the CPE bit. See Section 12.4.4.1, “COP Timer Disable (COPDIS)—Bit 15,” on page 12-14 for more information.

11.2.1.2 COP Timer Divider (CT)—Bit 14

The COP timer divider (CT) control bit is used to program the division ratio for the COP timer. The CT bit is cleared on hardware reset. Table 11-1 shows how this bit is set.

Table 11-1. COP Timer Divider Definition

CT	Division
0	/8
1	/64

11.2.1.3 Reserved Bits—Bits 13–12

Bits 13–12 are reserved and are read as zero during read operations. These bits should be written with zero for future compatibility.

11.2.1.4 RTI Timer Enable (RTE)—Bit 11

The RTI timer enable (RTE) control bit is used to enable the RTI and COP timer functionality. The RTI timer can operate without the CPE bit being set, but both the RTE bit and the CPE bit must be set for the COP timer to operate. The RTE bit is cleared on hardware reset.

NOTE:

Clearing the RTE bit disables the input clock to both the RTI timer and COP timer, and can be used to reduce power consumption in systems that do not require these functions.

11.2.1.5 RTI Enable (RTIE)—Bit 10

The RTI enable (RTIE) control bit is used to enable interrupts from the real-time timer. When the RTIE bit is cleared, the interrupt is disabled and any pending RTI is cleared. The RTIE bit is cleared on hardware reset.

The interrupt vector for the RTI is $0016. As with all on-chip peripheral interrupts for the DSP56824, the status register (SR) must first be set to enable maskable interrupts (interrupts of level IPL 0). Next, the CH1 bit (bit 14) in the IPR must also be set to enable this interrupt. (See Section 3.3.1, “DSP56824 Interrupt Priority Register (IPR),” on page 3-14 for more information.) Finally, set the RTIE bit to enable the RTI.

11.2.1.6 RTI Flag (RTIF)—Bit 9

The RTI flag (RTIF), bit 9, is automatically set to one at the end of every RTI period—that is, when the RTI timer reaches zero. This read-only bit is cleared by writing a one to bit 9, RTIF, in the COPCTL register. The RTIF bit is cleared on hardware reset.

11.2.1.7 RTI Prescaler (RP)—Bit 8

The RTI prescaler (RP) control bit is used to program the prescaler for the RTI timer. Table 11-2 shows the different available selections. The RP bit is cleared on hardware reset.

Table 11-2. Real-Time Prescaler Definition

RP	Division
0	/1
1	/4

11.2.1.8 RTI/COP Divider (DV[7:0])—Bits 7–0

The RTI/COP divider (DV[7:0]) control bits are used to program the last divider in the RTI clock chain. When the COPCNT register decrements to zero, the 8 bits of this register are reloaded with the value in the DV[7:0] bits. To set the COPCNT register for division by *n* counts, load the DV bits with the value (*n* – 1). The DV bits are cleared on hardware reset.

NOTE:

Dividing by 1 (DV[7:0] = $0) is not allowed for this divider. All other divider values from 2 to 256 are allowed. Since the value of the DV[7:0] bits is $0 upon reset (an illegal value), this register must be written to before the real-time or COP timer is first used.

11.2.2 COP and RTI Count (COPCNT) Register

The COP and RTI Count (COPCNT) register is a 16-bit read-only register reflecting the value of the COP/RTI divider chain. The COPCNT register is written by the divider chain on the edge of the prescaler clock opposite that used to clock the divider chain. The COPCNT register can be read at any time.

11.2.2.1 Reserved Bits—Bits 15–13

Bits 15–13 are reserved and are read as zero during read operations.

11.2.2.2 RTI/COP Divider (DV[7:0])—Bits 12–5

When the COPCNT register is read, the RTI /COP divider (DV[7:0]) bits show the current value of the last divider in the RTI clock chain.

11.2.2.3 RTI Prescaler (RP[1:0])—Bits 4–3

When the COPCNT register is read, the RTI prescaler (RP[1:0]) bits show the current value of the prescaler portion of the RTI clock chain.

11.2.2.4 Scaler (SC[2:0])—Bits 2–0

When the COPCNT register is read, the scaler (SC[2:0]) bits show the current value of the scaler portion of the RTI clock chain.

11.2.3 COP Reset (COPRST) Register

The COP reset (COPRST) register is a 16-bit write-only register and is used for two purposes. The first use of this register is for resetting the COP timer before it times out. The COP timer, once enabled, can only be reset by writing a sequence in the correct order to the COPRST register. The required sequence is described in Section 11.3, “Programming the COP and RTI Timers.” This resets the COP timer to its maximum value, and it begins counting down again. The COP timer is the last divide-by-8 or divide-by-64 portion of the counter chain.

NOTE:

Writing to this register does not affect the portion of the timing chain shared by both the RTI and the COP timers.

The second use of this register is to enable writes to the COPCTL register. It is very important that the COPCTL register is not accidentally overwritten if the computer is not operating properly, so writes to this register are enabled only after a correct sequence is written to the COPRST register.

11.3 Programming the COP and RTI Timers

Accidental writes to COPCTL are prevented by requiring the user to write a specific sequence to COPRST before writes to the COPCTL register are enabled.

NOTE:

Writing this sequence also has the effect of resetting the COP timer.

The exact sequence is as follows:

1. Write the value $5555 to the COPRST register.
2. Execute a NOP instruction.
3. Write the value $AAAA to the COPRST register.
4. Execute a NOP instruction.
5. Write the value $5555 to the COPRST register.
6. Execute a NOP instruction.
7. At this point in the sequence, it is now possible to write the COPCTL register. Use the following sequence to modify the bits in the COPCTL register.
8. Write the value $AAAA to the COPRST register. At this point, the COP timer is reset to its maximum value—7 if CT = 0 and 63 if CT = 1. This final write also resets the sequence mechanism and disables writing to the COPCTL register, so it is necessary to begin again at step 1.
9. Execute a NOP instruction.

It is also important to modify the COPCTL bits in the proper order when programming the COPCTL register:

1. Write the correct sequence to the COPRST register to enable writes to the COPCTL register, as shown in the previous procedure.
2. Clear the RTE bit in the COPCTL register using a BFCLR instruction.
3. Change any of the following COPCTL bits as desired: CPE, CT, RTIE, RP, or DV[7:0].
4. Set the RTE bit in the COPCTL register using a BFSET instruction.
5. Disable writes once again to the COPCTL register by writing the value $AAAA to the COPRST register, as described in step 8 in the preceding procedure.

NOTE:

If RTE is set, bits in the COPCTL register can be written, but a malfunction in the COP/RTI module can occur. Always clear the RTE bit before writing to the COPCTL register to ensure proper operation of this module.

11.3.1 COP and RTI Timer Resolution

Table 11-3 shows the resolution and range of the RTI timer for different prescaler clock frequencies.

Table 11-3. COP Timer Range and Resolution

Prescaler Frequency	RP Prescaler	Resolution (Preload = 0)	Range (Preload = $2^8 - 1$)
32.00 kHz	/1	250 ms	64 ms
(31.25 µs)	/4	1 ms	256 ms
38.4 kHz	/1	208.3 µs	53.3 ms
(26.04 µs)	/4	833.3 µs	213.3 ms

The RTI occurs every $2^3 \times 4^{RP} \times (DV[7:0] + 1)$ prescaler clock cycles.

The COP time-out occurs every $2^3 \times 4^{RP} \times (DV[7:0] + 1) \times 8^{(CT + 1)}$ prescaler clock cycles.

11.3.2 COP/RTI Timer Low-Power Operation

The COP/RTI module can be shut off to reduce power consumption when the module is not required by an application. To shut off this module, set the RTE bit in the COPCTL register to 0; see Section 11.2.1, “COP and RTI Control Register (COPCTL).” This gates off all clocks to the COP/RTI module. If either the COP or RTI function is required, the RTE bit must remain set.

It is also possible to run the RTI or COP timer when the chip is in stop mode. When the RTI timer reaches zero, a signal is generated that wakes up the DSP56800 core and brings it out of stop mode. Caution should be taken that the COP timer is not allowed to time out when the DSP56824 is in stop mode. The DSP56824 is reset whenever the COP timer times out, regardless of the operating mode it is in when COP time-out occurs. Both timers can continue to operate in wait mode.

NOTE:

The RTI timer can bring the DSP56824 out of stop mode independently of the value of the RTIE bit in the COPCTL register or of the bits in the IPR (described in Section 3.3.1, “DSP56824 Interrupt Priority Register (IPR),” on page 3-14).

11.3.3 Programming Example

Example 11-1 shows how to set up the COP timer.

Example 11-1. Sending a COP Reset

```
;**********************
;* COP Timer example  *
;* for COP/RTI Module *
;* of DSP56824 chip   *
;**********************
;
START               EQU   $0040         ; Start of program
BCR                 EQU   $FFF9         ; Bus Control Register
COPCTL              EQU   $FFF1         ; COP/RTI Control Register
COPRST              EQU   $FFF0         ; COP Reset Register [write-only]
;PCR0 [unused]      EQU   $FFF2         ; PLL Control Register 0
PCR1                EQU   $FFF3         ; PLL Control Register 1
;**************
;* Vector setup*
;**************
      ORG           P:$0000             ; Cold Boot
      JMP           START               ; also Hardware RESET vector (Mode 0, 1, 3)
      ORG           P:$E000             ; Warm Boot
      JMP           START               ; Hardware RESET vector (Mode 2)
      ORG           P:$E002             ;
      JMP           COPOUT              ; COP Watchdog RESET vector (Mode 2)
      ORG           P:START             ; Start of program
;**************
;* General setup*
;**************
      MOVEP         #$0000,X:BCR        ; External Program memory has 0 wait states.
                                        ; External data memory has 0 wait states.
                                  ; Port A pins tri-stated if no external access.
```

Example 11-1. Sending a COP Reset (Continued)

```
;****************************
;* Prescaler Clock setup   *
;* (source for divider chain)*
;****************************
;
       MOVEP        #$06C0,X:PCR1       ; Configure:
                                        ; (PLLE) PLL disabled (bypassed)
                                        ; -- Oscillator supplies Phi Clock
                                        ; (PLLD) PLL Power Down disabled (PLL active)
                                        ; -- PLL block active for PLL to attain lock
                                        ; (LPST) Low Power Stop disabled
                                        ; (PS[2:0]) Set Prescaler Divider to /256
                                        ; (CS[1:0]) Clockout pin (CLKO) disabled
;      [PCR0 unused]                    ; Feedback Divider unused (PLL bypassed)
;*********************
;
;* COP Timer setup  *
;*********************
;
       MOVEP        #$5555,X:COPRST     ; Begin COP reset sequence
       NOP                              ;
       MOVEP        #$AAAA,X:COPRST     ;
       NOP                              ;
       MOVEP        #$5555,X:COPRST     ;
       NOP                              ; COPCTL write-enabled
       BFCLR        #$0800,X:COPCTL     ; (RTE) RTI Timer disabled
                                        ; -- COP/RTI divider chain disabled, thus
                                        ; -- COP timer (and RTI timer) disabled
       MOVEP        #$C1FF,X:COPCTL     ; Configure:
                                        ; (CPE) COP time-out (chip reset) Enabled
                                        ; (CT) Set COP Timer divider to /64.
                                        ; (RTIE) RTI disabled
                                        ; (RTIF) RTI flag remains.
                                        ; (RP) Set COP/RTI Prescaler to /4.
                                        ; (DV[7:0]) Set COP/RTI Divider to /256.
       BFSET        #$0800,X:COPCTL     ; (RTE) RTI Timer Enabled
                                        ; -- COP/RTI divider chain enabled, thus
                                        ; -- COP Timer (and RTI Timer) enabled
       MOVEP        #$AAAA,X:COPRST     ; Reset (disable) write mechanism for COPCTL
                                        ; and reset COP Timer.
       NOP                              ; -- End COP reset sequence.
;**************
;* Main routine*
;**************
;
                                        ; ...
TEST                                    ; Test Loop
                                        ; ...
                                        ; code to be watched for potentially wayward
                                        ; execution
                                        ; ...
       JSR          COPCLR              ; Let the COP know we have not left town.
       BRA          TEST
COPCLR                                  ; COP CLEAR -- COP Timer Reset Routine
       ;***************************************
       ;* Clear the COP Timer: code execution is still valid*
       ;***************************************
       ;
       MOVEP        #$5555,X:COPRST     ; Begin COP reset sequence.
       NOP                              ;    .
       MOVEP        #$AAAA,X:COPRST     ;    .
       NOP                              ;    .
       MOVEP        #$5555,X:COPRST     ;    .
       NOP                              ; COPCTL write-enabled
       MOVEP        #$AAAA,X:COPRST     ; Reset (disable) write mechanism for COPCTL
                                        ; and reset COP Timer.
       NOP                              ; -- End COP reset sequence.
       RTS
```

Example 11-1. Sending a COP Reset (Continued)

```
COPOUT                                  ; COP time OUT -- COP Watchdog RESET Routine
                                        ; the COP timed out: system failure has
                                        ; occurred
                                        ; ...
                                        ; error recovery code
                                        ; ...
```

Freescale Semiconductor, Inc.
ARCHIVED BY FREESCALE SEMICONDUCTOR, INC. 2005

Chapter 12
OnCE™ Module

The DSP56824 provides board and chip-level testing capability through two on-chip modules that are both accessed through the JTAG/OnCE port. These modules are as follows:

- On-Chip Emulation (OnCE) module
- Test access port (TAP) and 16-state controller, also called the Joint Test Action Group (JTAG) port

The presence of the JTAG/OnCE port allows the user to insert a DSP56824 chip into a target system while retaining debug control. This capability is especially important for devices without an external bus, since it eliminates the need for a costly cable to bring out the footprint of the chip, as required by a traditional emulator system. In addition, the JTAG/OnCE port can be used to program the internal flash memory.

NOTE:

The easiest way to program the DSP56824 flash memory is with the use of the DSP56824 Application Development System (ADS). Contact your Motorola sales representative for more information on this product.

The On-Chip Emulation (OnCE) module is a Motorola-designed module used in DSP chips to debug application software. The module is a separate on-chip block that allows non-intrusive interaction with the DSP and is accessible through the pins of the JTAG interface. The OnCE module makes it possible to examine the contents of registers, memory, or on-chip peripherals in a special debug environment. This avoids sacrificing any user-accessible resources to perform debugging.

The capabilities of the OnCE module include the ability to do the following:

- 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 DSP microprocessor instruction
- Display or modify the contents of any core or memory-mapped peripheral registers
- Display or modify any desired sections of program or data memory
- Trace up to 256 instructions
- Save or restore the current state of the execution 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) that can force 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

See Section 12.3, “OnCE Module Architecture,” for a detailed description of this port.

12.1 Combined JTAG/OnCE Interface Overview

The JTAG and OnCE blocks are tightly coupled. Figure 12-1 shows the block diagram of the JTAG/OnCE port with its two distinct parts. The JTAG port is the master; it must enable the OnCE module before the OnCE module can be accessed.

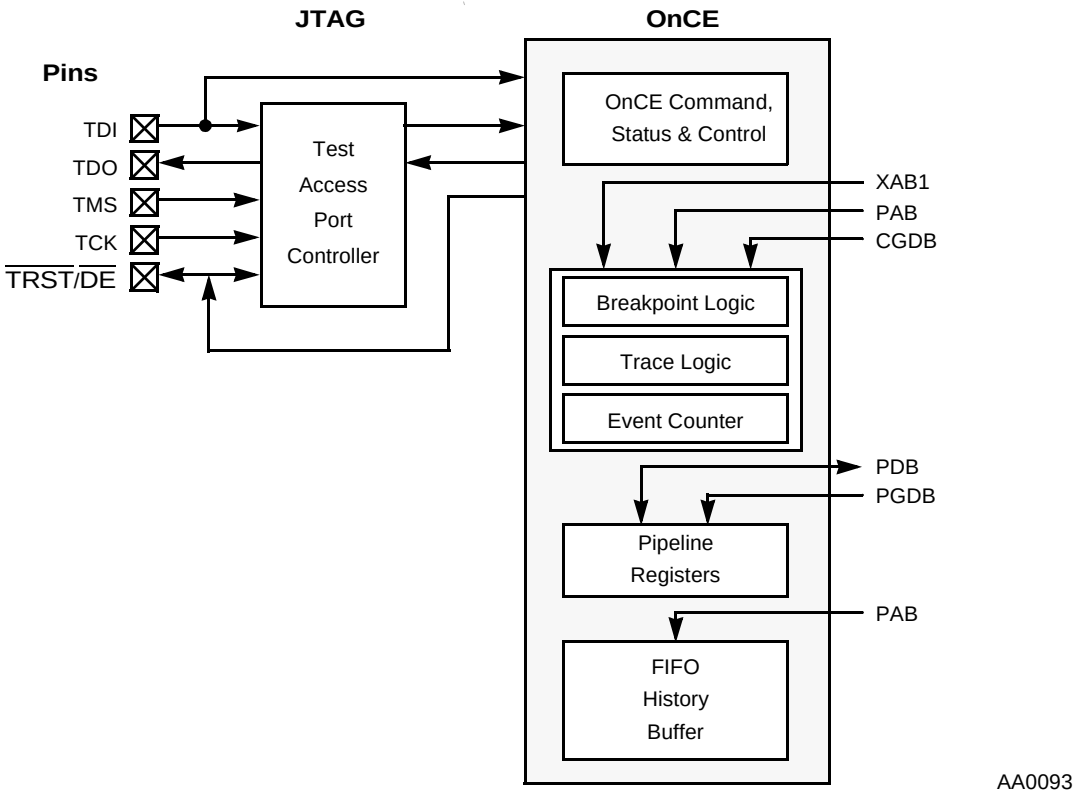


Figure 12-1. JTAG/OnCE Port Block Diagram

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 DSP core
- JTAG programming model—accessible through the JTAG port

The programming models are discussed in more detail in Section 12.3, “OnCE Module Architecture,” and Section 13.2, “JTAG Port Architecture,” on page 13-2.

12.2 JTAG/OnCE Port Pinout

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 DSP56824 also uses the optional test reset ($\overline{\text{TRST}}$) input signal and multiplexes it so that the same pin can support the debug event ($\overline{\text{DE}}$) output signal used by the OnCE interface. The pin functions are described in Table 12-1.

Table 12-1. 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-statable 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 provides a gated clock to synchronize the test logic and shift serial data through the JTAG/OnCE port. The maximum frequency for TCK is one-eighth the maximum frequency of the DSP56824 (that is, 5 MHz for TCK if the maximum CLK input is 40 MHz). 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.
$\overline{\text{TRST}}$/$\overline{\text{DE}}$	**Test Reset/Debug Event**—This bidirectional pin, when configured as an input, provides a reset signal to the TAP controller. When configured as an output, it signals debug events detected on a trigger condition. Pin operation is configured by bit 14 of the OnCE control register (OCR). The $\overline{\text{TRST}}$/$\overline{\text{DE}}$ pin has an on-chip pull-up resistor.

The $\overline{\text{TRST}}$/$\overline{\text{DE}}$ pin can be configured for one of two functions. It can be used as a reset for the JTAG port or can provide a useful event-acknowledge feature. This selection is performed through appropriate control bits in the OCR. The two functions available are as follows:

- $\overline{\text{TRST}}$—When enabled, this input resets the JTAG TAP controller state machine.
- $\overline{\text{DE}}$—When enabled, this open-drain output provides a signal that indicates that 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)

Events cause $\overline{\text{TRST}}$/$\overline{\text{DE}}$ to be asserted only if DE = 1 and DRM = 0 (in the OCR), as shown in Table 12-2.

Table 12-2. DE and DRM Encoding for $\overline{\text{TRST}}$/$\overline{\text{DE}}$ Assertion

DE	DRM	Function
0	X	Input: JTAG reset when $\overline{\text{TRST}}$ pulled low
1	0	Output: pulled low on OnCE events
1	1	Output: disabled (weak on-chip pull-up)

Figure 12-8 on page 12-16 shows the internal configuration of the $\overline{\text{TRST}}$/$\overline{\text{DE}}$ pin. The open-drain output function indicates that a OnCE event has occurred. For example, a trace or breakpoint occurrence causes the pin to go low for a minimum of 8 Phi clocks. This pin is further described in Section 12.4.4.6, “Event Modifier (EM[1:0])—Bits 6–5.”

When the JTAG IR does not contain an ENABLE_ONCE instruction, the OCR control bits are reset only by assertion of $\overline{\text{RESET}}$ or COP timer reset. If the ENABLE_ONCE instruction is in the JTAG IR at the time of reset, the OCR bits are not modified.

NOTE:

Although this implementation is not in strict accordance with IEEE 1149.1a-1993, a JTAG user can confidently use this pin in its $\overline{\text{TRST}}$ mode by ensuring that $\overline{\text{RESET}}$ is asserted on power up. Setting OCR[14] requires a very specific and lengthy JTAG sequence. OCR[14] cannot be set by any other sequence. Similarly, to guarantee full hardware reset, both $\overline{\text{RESET}}$ and $\overline{\text{TRST}}$ should be asserted.

12.3 OnCE Module Architecture

While the JTAG port described in Section 13.2, “JTAG Port Architecture,” on page 13-2 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 a Motorola Application Development Module (ADM) board.

A typical debug environment consists of a target system where the DSP resides in the user-defined hardware. The DSP’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 DSP and its associated circuitry. Refer to your development system’s documentation for information on debugging using the JTAG port interface.

The OnCE module is composed of four different sub-blocks, each of which performs a different task:

- Command, status, and control
- Breakpoint and trace
- Pipeline registers
- FIFO history buffer

12.3.1 OnCE Port Block Diagram

A block diagram of the OnCE module is shown in Figure 12-2 on page 12-5. Information is shifted serially in and out of the OnCE port logic via the TDI and TDO signals. Typically the source and destination of this information is one or more OnCE port registers. An overview of the registers is given in the following section.

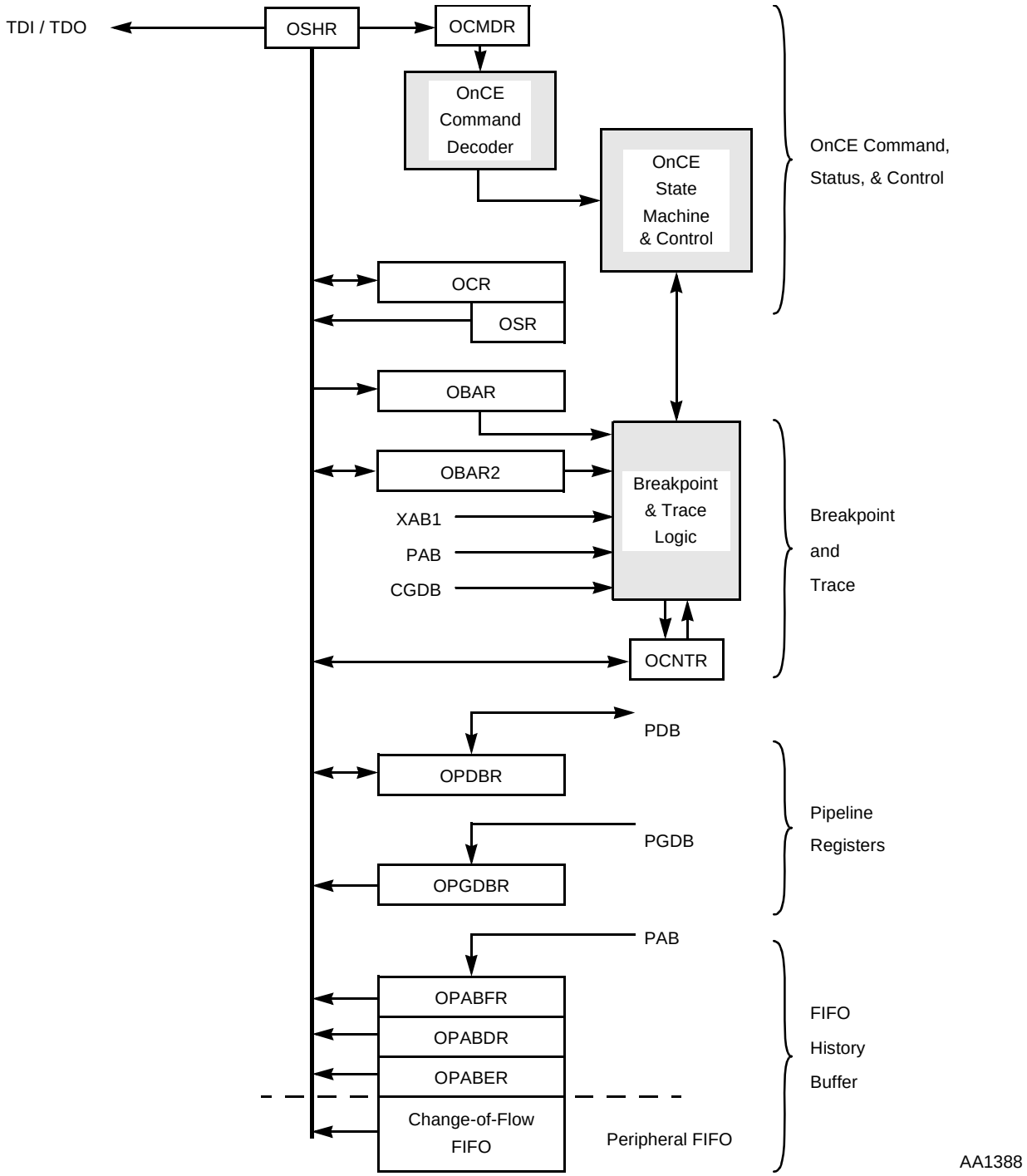
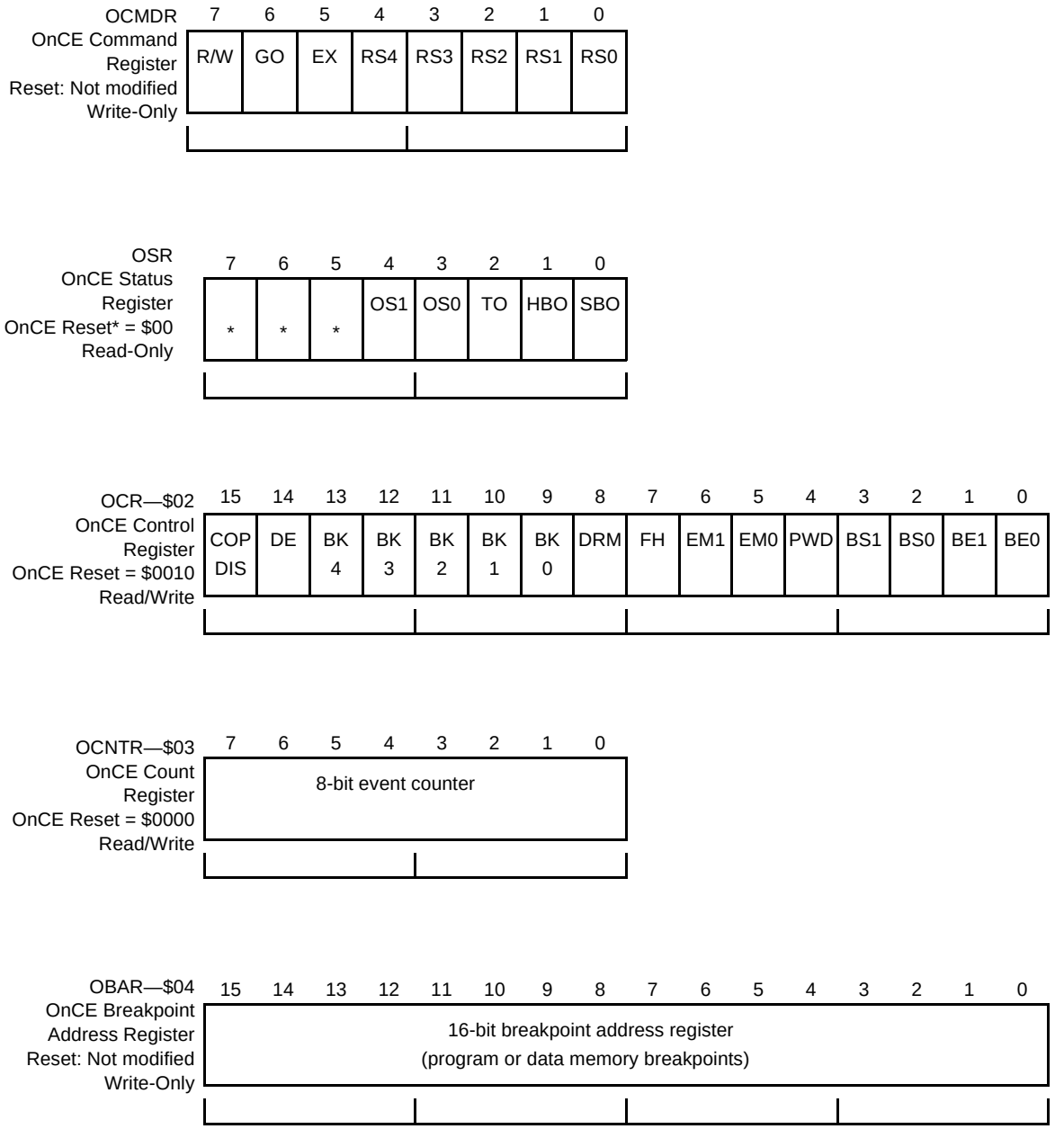


Figure 12-2. DSP56824 OnCE Block Diagram

12.3.2 OnCE Programming Model

The programming model for the OnCE port consists of the registers which control all emulation and debugging tasks. These consist of command, status, and control registers, as well as breakpoint and bus status registers. The registers in the OnCE port programming model are shown in Figure 12-3 and Figure 12-4.



* Indicates reserved bits, written as zero for future compatibility

OnCE reset occurs when hardware or COP reset occurs, and an ENABLE_ONCE instruction is not latched into the JTAG instruction register (IR)

AA1389A

Figure 12-3. OnCE Module Registers Accessed Through JTAG

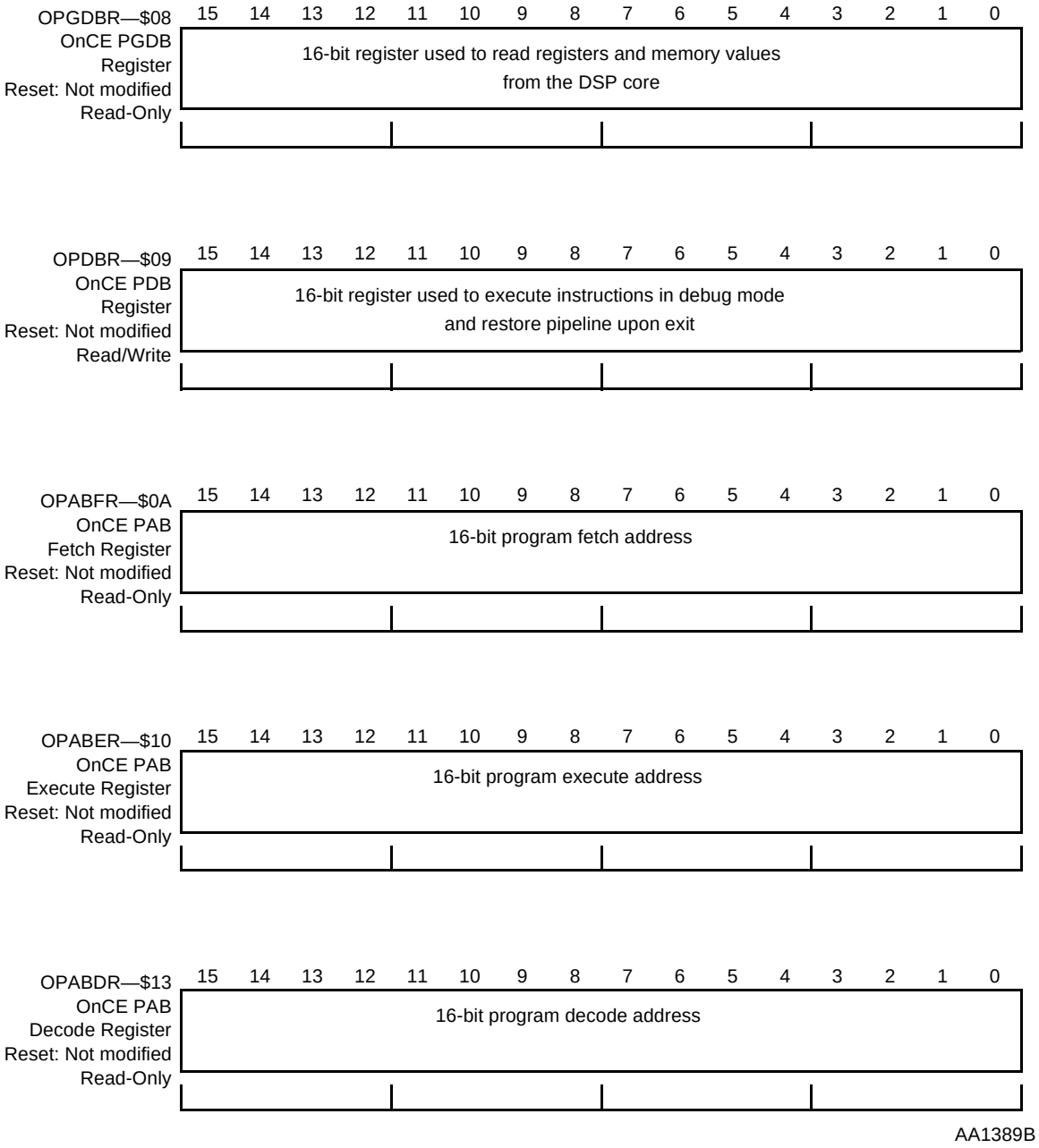
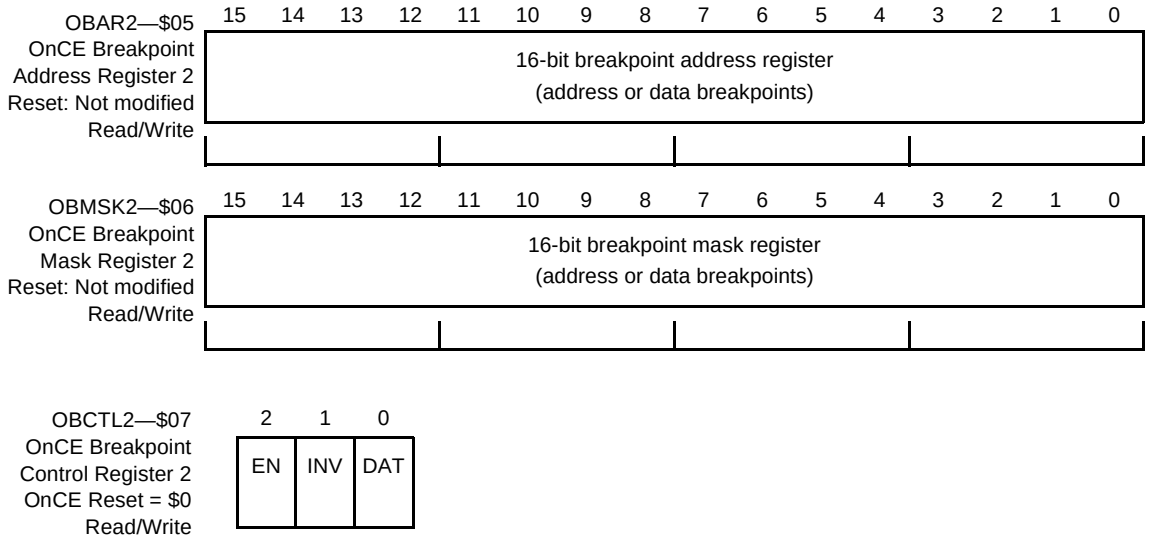


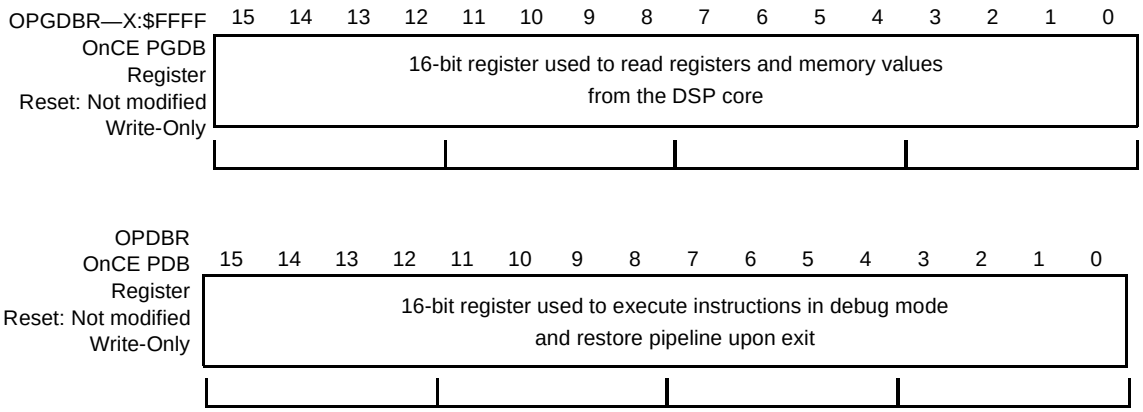
Figure 12-3. OnCE Module Registers Accessed Through JTAG (Continued)



OnCE reset occurs when hardware or Computer Operating Properly (COP) reset occurs, and an ENABLE_ONCE instruction is not latched into the JTAG port instruction register (IR)

AA1389C

Figure 12-3. OnCE Module Registers Accessed Through JTAG (Continued)



OnCE Port interrupt vector:

OnCE TRAP P:$000C

OnCE TRAPs are Level 1 interrupts and not programmable in the IPR

Note: OPGDBR and OPDBR are not dedicated OnCE module registers. They share functionality with the core. If used incorrectly, they can give unexpected results.

AA1390

Figure 12-4. 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 such that a OnCE event other than trace generates a level 1 non-maskable interrupt. This interrupt capability is described in Section 12.4.4.6, “Event Modifier (EM[1:0])—Bits 6–5.”

12.3.3 OnCE State Machine and Control Block

The OnCE state machine has the following states:

- IDLE—Do nothing
- STATCOM—Poll status and get command
- DEC1—Decode 1
- DEC2—Decode 2
- RWREG—Read/write a OnCE register
- WPDBR—Write OPDBR

Figure 12-5 shows the possible paths through the OnCE state machine. Table 12-3 on page 12-11 gives a description of each state transition (as shown in Figure 12-5).

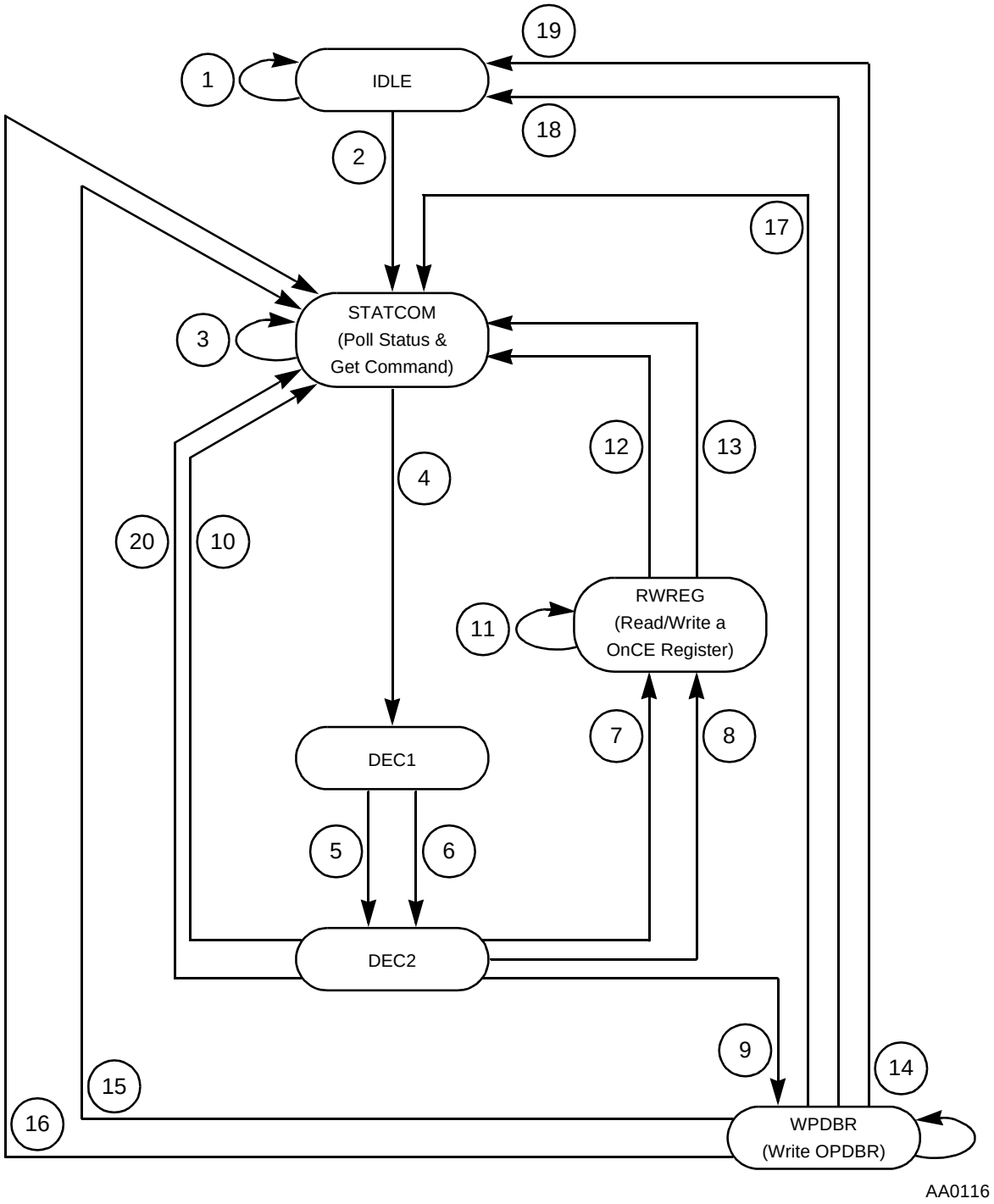


Figure 12-5. OnCE State Machine

Table 12-3. OnCE State Machine Transitions

Transition	Description
1	Chip is in normal mode.
2	Chip enters debug mode. Go to STATCOM state.
3	Get command. Wait for the external controller to finish sending an 8-bit command.
4	External controller has finished sending the command. Start decoding OnCE command.
5	OnCE will not access any registers. The core is to repeat executing the previous instruction.
6	OnCE is to access a dedicated register. This is the default state when the OnCE command selects a reserved option.
7	Read or write any OnCE dedicated register besides the PDB register, OBDBR. The core will not be asked to execute any instruction.
8	Read OPDBR. The core will not be asked to execute any instruction.
9	Write to the OPDBR. The core will eventually be asked to execute an instruction.
10	OnCE is to clear either the OnCE breakpoint counter (OMBC) or the OnCE trace counter (OTC).
11	Read or write the OnCE dedicated register. Wait for 16 input or output bits to be shifted.
12	Finished writing a OnCE register. Send an acknowledge pulse.
13	Finished writing a OnCE register. Do not send an acknowledge pulse.
14	Write to the OPDBR. Wait for the 16-bit core command or operand to be shifted.
15	Finished writing to the OPDBR. Execute a one-word core instruction, but do not exit from debug mode.
16	Finished writing to the OPDBR. Execute a two-word core instruction, but do not exit from debug mode.
17	Finished writing to the OPDBR. Transfer its contents to the PDB. This is the first word of a two-word instruction. Get the second word.
18	Finished writing to the OPDBR. Execute a one-word core instruction while exiting the OnCE debug mode.
19	Finished writing to the OPDBR. Execute a two-word core instruction while exiting the OnCE debug mode.
20	The core has finished executing the current instruction. Get the next OnCE command.

12.4 Command, Status, and Control Registers

The OnCE command, status, and control registers are described in Section 12.4.1, “OnCE Shift Register (OSHR),” through Section 12.4.6, “OnCE Status Register (OSR).” They include these registers:

- OnCE shift register (OSHR)
- OnCE command register (OCMDR)
- OnCE decoder (ODEC) register
- OnCE control register (OCR)
- OnCE breakpoint 2 control (OBCTL2) register
- OnCE status register (OSR)

12.4.1 OnCE Shift Register (OSHR)

The OnCE shift register (OSHR) is a JTAG shift register that samples the TDI pin on the rising edge of TCK in Shift-DR and provides output to the TDO pin on the falling edge of TCK. During OCMDR and OSR transfers, this register is 8 bits wide. During all other transfers, this register is 16 bits wide. The input from TDI is clocked first into the MSB of the OSHR. The TDO pin receives data from the least significant bit (LSB) of the OSHR.

12.4.2 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: read commands that cause the chip to deliver required data, and write commands that transfer data into the chip and write it in one of the on-chip resources. The commands are 8 bits long and have the format shown in Figure 12-6. The lowest 5 bits (RS[4:0]) identify the source for the operation, defined in Table 12-4. Bits 5, 6, and 7 define the exit (EX) command bit (Table 12-5 on page 12-13), the execute (GO) bit (Table 12-6 on page 12-13), and the read/write (R/$\overline{W}$) bit (Table 12-7 on page 12-14), respectively.

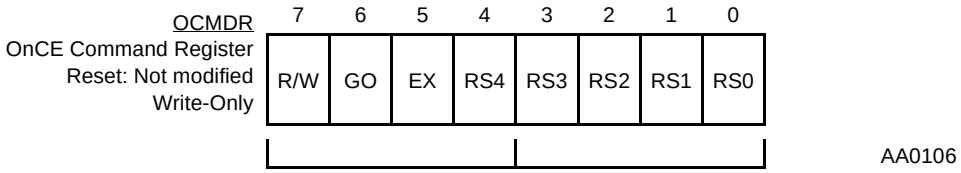


Figure 12-6. OnCE Command Format

Table 12-4. Register Select Encoding

RS[4:0]	Register/Action Selected	Mode	Read/Write
00000	No register selected	All	Not applicable
00001	OnCE breakpoint and trace counter (OCNTR)	All	Read/Write
00010	OnCE debug control register (OCR)	All	Read/Write

Table 12-4. Register Select Encoding (Continued)

RS[4:0]	Register/Action Selected	Mode	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
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 12-5. EX Bit Definition

EX	Action
0	Remain in the debug processing state
1	Leave the debug processing state

Note: Bit 5 in the OnCE command word is the exit command. To leave debug mode and reenter the normal mode, both the EX and GO bits must be asserted in the OnCE input command register.

Table 12-6. GO Bit Definition

GO	Action
0	Inactive—no action taken
1	Execute DSP instruction

Table 12-7. R/$\overline{W}$ Bit Definition

R/$\overline{W}$	Action
0	Write to the register specified by the RS[4:0] bits
1	Read from the register specified by the RS[4:0] bits

12.4.3 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 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.

12.4.4 OnCE Control Register (OCR)

The 16-bit OnCE control register (OCR) contains bit fields that determine how breakpoints are triggered and what action occurs when a OnCE event occurs, and it also controls other miscellaneous OnCE features. Figure 12-7 illustrates the OCR and its fields.

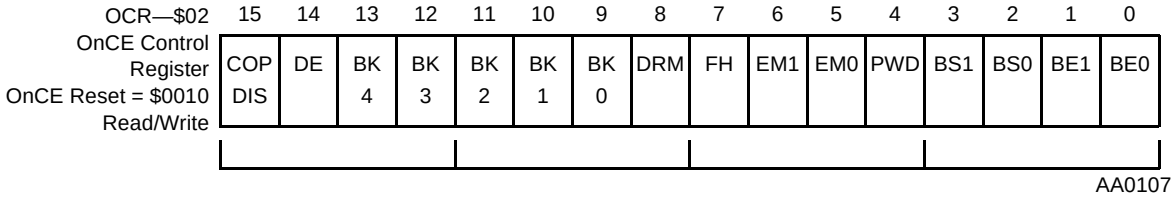


Figure 12-7. OCR Programming Model

12.4.4.1 COP Timer Disable (COPDIS)—Bit 15

The COP timer disable (COPDIS) bit is used to prevent the COP timer from resetting the DSP chip 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. (No COP reset can be generated.) However, the COPDIS bit overrides the CPE bit when both are set. See Section 11.2.1.1, “COP Enable (CPE)—Bit 15,” on page 11-4 for more information.

12.4.4.2 $\overline{DE}$ Pin Output Enable (DE)—Bit 14

The $\overline{DE}$ pin enable (DE) bit configures the $\overline{TRST}$/$\overline{DE}$ pin as an output. When DE is set, the $\overline{TRST}$ capability is disabled. When DE is cleared, the pin is configured as the $\overline{TRST}$ input. To avoid the accidental reset of JTAG, the user should change DE only when DRM = 1. DE and DRM should not be changed simultaneously. See Section 12.4.4.4, “Debug Request Mask (DRM)—Bit 8,” for details on how DE and DRM bits control $\overline{TRST}$/$\overline{DE}$ functionality.

12.4.4.3 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 set up a breakpoint on one address and an interrupt on another address. Table 12-8 lists the different breakpoint combinations available.

Table 12-8. 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 a 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 by trace mode using the instruction count specified in the OCNTR. Then the trigger is generated.
10001	Breakpoint 1 or breakpoint 2 occurs once, followed by trace mode using the instruction count specified in the OCNTR. Then the trigger is generated.
10010	Breakpoint 1 and breakpoint 2 occur simultaneously, followed by trace mode using the instruction count specified in the OCNTR. Then the trigger is generated.
10100	Breakpoint 1 occurs once, followed by 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 by trace mode using the instruction count specified in the OCNTR. Then the trigger is generated.
Note: All other encodings of BK[4:0] are illegal and may cause unpredictable results.	

Breakpoint 2 is a simple address compare. It is unaffected by BS/BE bits, except that BE = 00 disables it. BE = 00 disables all of the preceding BK settings except pure trace mode (BK[4:0] = 10111). See Section 12.4.5, “OnCE Breakpoint 2 Control Register (OBCTL2),” for more information.

12.4.4.4 Debug Request Mask (DRM)—Bit 8

The debug request mask (DRM) bit is used to mask $\overline{DE}$, the external debug request signal. When this bit is cleared, a pulse on the $\overline{DE}$ input pin causes the DSP to enter the debug mode of operation. When this bit is set, the DSP does not enter debug mode when a pulse is placed on the $\overline{DE}$ input.

This bit is also used to enable or disable $\overline{TRST}/\overline{DE}$ pin drive when the DE bit is set. When DRM = 0 and DE = 1, event occurrences assert $\overline{TRST}/\overline{DE}$ low. When DRM = 1, $\overline{TRST}/\overline{DE}$ is not asserted even if DE = 1. Figure 12-8 shows how these 2 bits affect the $\overline{TRST}/\overline{DE}$ pin.

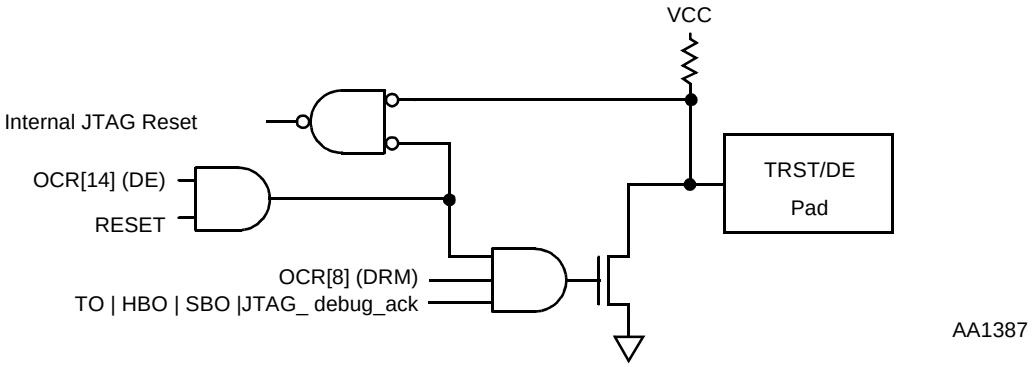


Figure 12-8. Debug Event Pin

12.4.4.5 FIFO Halt (FH)—Bit 7

The FIFO halt (FH) bit allows the user to 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 (that is, it is a control bit, never a status bit). This gives the user 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 that the FIFO can be halted in the middle of instruction execution, leading to incoherent OPFIFO register contents.

12.4.4.6 Event Modifier (EM[1:0])—Bits 6–5

The event modifier (EM[1:0]) bits allow different actions to take place when a OnCE event occurs. 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 DE pin is enabled, each event occurrence drives 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, a 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 via OnCE module transfers.

If EM[1:0] = 01, the core does not halt when an event occurs (that is, debug mode is not entered), but the OPFIFO, OPABFR, OPABDR, and OPABER registers stop capturing. This allows the user to access 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 1 interrupt occurs with a vector at location P:$000C. This allows the user to execute diagnostic subroutines upon event occurrences or even to patch program memory by setting a breakpoint at the beginning of the code to be patched. Note that 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 that 8 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 (driven low) for 8 Phi clock cycles, and then released. If another event occurs within those 8 Phi clock cycles or directly after, the occurrence flag is set immediately 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 user does not wish to change the value of OCR, writing OCR with its current value rearms the event successfully. $\overline{DE}$, if enabled, is released when this 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 that $\overline{DE}$ 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 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 12-9 on page 12-17 summarizes the different EM encodings.

Table 12-9. 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.

Care must be taken when changing the EM bits. It is recommended that the particular event (trace, hardware, or software breakpoint) is disabled first. On the next OCR write, the EM bits can be modified and the event reenabled. 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 that can be used 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.

12.4.4.7 Power Down Mode (PWD)—Bit 4

The power down mode (PWD) bit is a power-saving option that reduces running current for applications that do not use the OnCE module. The user can set or reset the PWD bit by writing to the OCR. On hardware reset (deassertion of the $\overline{RESET}$ signal), this bit is set (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. To ensure proper operation, breakpoints should be completely disabled before setting PWD.

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:

- A 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.

12.4.4.8 Breakpoint Selection (BS[1:0])—Bits 3–2

The 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 inTable 12-10. These bits are used only in determining triggering conditions for breakpoint 1, not for additional future breakpoint comparators.

The BS and BE bits apply only to the breakpoint 1 mechanism. Breakpoint 2 and future breakpoint mechanisms are unaffected by BS or BE bit encodings, except for the fact that all breakpoint mechanisms are disabled when BE[1:0] = 00.

Table 12-10. 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 to determine how the address breakpoint hardware is set up. The decoding scheme for BS[1:0] and BE[1:0] is shown in Table 12-11.

Table 12-11. Breakpoint Programming with the BS[1:0] and BE[1:0] Bits

Function	BS[1:0]	BE[1:0]
Disable all breakpoints[1]	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

1. 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 12.9.2, “Entering Debug Mode.”

12.4.4.9 Breakpoint Enable (BE[1:0])—Bits 1–0

The breakpoint enable (BE[1:0]) control bits enable or disable the breakpoint logic and select the type of memory operations (read, write, or access) upon which the breakpoint logic operates. Access means either a read or write can be taking place. These bits are cleared on hardware reset. Table 12-12 describes the bit functions.

Table 12-12. 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 to determine how the address breakpoint hardware is set up. The decoding scheme for BS[1:0] and BE[1:0] is shown in Table 12-11. Breakpoints should remain disabled until after the OBAR is loaded. See Section 12.5.5, “OnCE Breakpoint and Trace Section,” and Section 12.9.2, “Entering Debug Mode,” for a more complete description of tracing and breakpoints**.** Breakpoints can be disabled or enabled for one memory space.

12.4.5 OnCE Breakpoint 2 Control Register (OBCTL2)

The OnCE breakpoint 2 control register (OBCTL2) is a 3-bit register used to program breakpoint 2. It 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 3 bits of a 16-bit word. The upper bits are reserved and should be written with zero to ensure future compatibility.

12.4.5.1 Reserved OBCTL2 Register Bits

Bits 15–3 are reserved and are read as zero during read operations. These bits should be written with zero to ensure future compatibility.

12.4.5.2 Enable (EN)—Bit 2

The enable (EN) bit is used to enable the second breakpoint unit. When EN is set, the second breakpoint unit is enabled. When EN is cleared, the second breakpoint unit is disabled.

12.4.5.3 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 unit. When INV is set, then the second breakpoint unit inverts the result of the comparison. When INV is cleared, no inversion is performed.

12.4.5.4 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.

12.4.6 OnCE Status Register (OSR)

The OnCE status register (OSR) is shown in Figure 12-9. By observing the values of the 5 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), allowing for efficient status polling. The OSR (and all other OnCE registers) are inaccessible in stop mode.

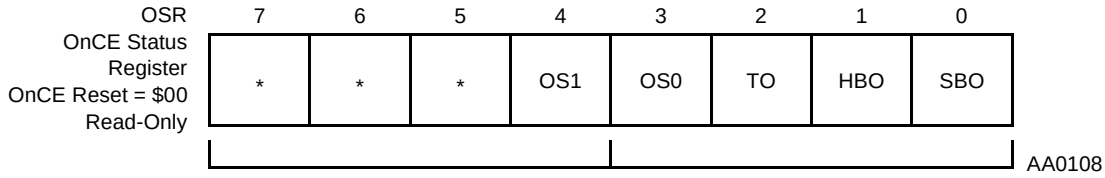


Figure 12-9. OSR Programming Model

12.4.6.1 Reserved OSR Bits—Bits 7–5

Bits 7–5 of the OSR are reserved for future expansion and are read as zero during DSP read operations.

12.4.6.2 OnCE Core Status (OS[1:0])—Bits 4–3

The OnCE core status (OS[1:0]) bits describe the operating status of the DSP core. It is recommended that the user read JTAG IR for OS[1:0] information, because OSR is unreadable in stop mode. Table 12-13 summarizes the OS[1:0] descriptions. On transitions from 00 to 11 and from 11 to 00, there is a small chance that intermediate states (01 or 10) may be captured.

Table 12-13. DSP Core Status Bit Description

OS[1:0]	Instruction	Description
00	Normal	DSP core executing instructions or in reset
01	STOP/WAIT	DSP core in stop or wait mode
10	Busy	DSP is performing external or peripheral access (wait states)
11	Debug	DSP core halted and in debug mode

NOTE:

The OS bits are also captured by the JTAG instruction register (IR). See Section 12.10.3, “Loading the JTAG Instruction Register,” for details.

12.4.6.3 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 JTAG IR and also by event rearm conditions described in Section 12.4.4.6, “Event Modifier (EM[1:0])—Bits 6–5.”

12.4.6.4 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 JTAG IR and also by the event rearm conditions described in Section 12.4.4.6, “Event Modifier (EM[1:0])—Bits 6–5.” See Section 12.4.4.3, “Breakpoint Configuration (BK[4:0])—Bits 13–9,” to determine which encodings are defined to generate hardware breakpoint events.

12.4.6.5 Software Breakpoint Occurrence (SBO)—Bit 0

The read-only software breakpoint occurrence (SBO) status bit is set when a DSP DEBUG instruction is executed (for example, a software breakpoint event) except when PWD = 1 in the OCR.The SBO bit is cleared by hardware reset provided that ENABLE_ONCE is not decoded in the JTAG IR. It is also cleared by the event rearm conditions described in Section 12.4.4.6, “Event Modifier (EM[1:0])—Bits 6–5.” The EM[1:0] bits determine if the core or the FIFO is halted.

12.5 Breakpoint and Trace Registers

Section 12.5.1, “OnCE Breakpoint/Trace Counter Register (OCNTR),” through Section 12.5.5, “OnCE Breakpoint and Trace Section,” describe these OnCE breakpoint and trace registers:

- OnCE breakpoint/trace counter register (OCNTR)
- OnCE memory address latch (OMAL) register
- OnCE breakpoint address register (OBAR)
- OnCE memory address comparator (OMAC)

12.5.1 OnCE Breakpoint/Trace Counter Register (OCNTR)

The OnCE breakpoint/trace counter register (OCNTR) is an 8-bit counter that allows for as many as 256 valid address compares or instruction executions (depending on whether it is configured as a breakpoint or trace counter) to occur before a OnCE event occurs. 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 trace mode is selected and EM[1:0] is not cleared, only *n* – 1 instructions must execute before an event occurs.

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 that ENABLE_ONCE is not decoded in the JTAG IR.

When used as a trace mode counter, the OCNTR allows the user to single step through code, meaning that after each DSP 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 the user has the 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 that do not have a normal flow or are getting hung up in infinite loops. Using the trace counter also enables the user to debug areas of code that are time critical.

It is important to note that 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.

12.5.2 OnCE Memory Address Latch (OMAL) Register

The OnCE memory address latch (OMAL) register is a 16-bit register that latches the PAB or XAB1 on every cycle. This latching is disabled if the OnCE module is powered down with the OCR's PWD bit.

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.

12.5.3 OnCE Breakpoint Address Register (OBAR)

The OnCE breakpoint address register (OBAR) is a 16-bit OnCE register that stores 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 1 only and has no effect on breakpoint 2.

12.5.4 OnCE Memory Address Comparator (OMAC)

The OnCE memory address comparator (OMAC) is a 16-bit comparator that compares the current memory address (stored by OMAL) with the memory address register (OBAR). If OMAC is equal to OMAL, then the comparator delivers a signal indicating that the breakpoint address has been reached.

12.5.5 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 that causes a breakpoint exception. This technique is limiting in that breakpoints can only be set in RAM at the beginning of an opcode and not on an operand. In addition, this technique does not allow breakpoints to be set on data memory locations. The DSP56824 instead provides on-chip address comparison hardware for setting breakpoints on program or data memory accesses. This allows 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. See Section 12.4.4.8, "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 (that is, 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 DSP56824 OnCE module provides a full-speed tracing capability—the capability to execute as many as 256 instructions at full speed and then reenter the debug processing state, or simply generate a OnCE event. This allows the user to single step through a program in the simplest case when the counter is set for one occurrence or to execute many instructions at full speed before returning to the debug mode.

The breakpoint logic and the trace logic have been designed to work together so that it is possible to set up more sophisticated trigger conditions that combine as many as two breakpoints and trace logic. The individual events and conditions that lead to triggering can also be modified.

During debugging, it is sometimes the case that 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.

12.6 Pipeline Registers

The OnCE module provides the user with the ability to halt the DSP core on any instruction boundary. Upon halting the core, the user can execute certain instructions 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 DSP instructions that move values to peripheral global data bus (PGDB), followed by OnCE commands that read the OPGDBR. This register is described in detail in Section 12.6.6, “OnCE PGDB Register (OPGDBR).”

Executing instructions from debug mode destroys the pipeline information. The OnCE module provides a means to preserve the pipeline upon entering debug mode and restoring it upon leaving 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. However, the pipeline can be restored regardless of 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 PDB register (OPDBR)
- OnCE PGDB register (OPGDBR)
- OnCE PAB change-of-flow FIFO (OPFIFO) register

12.6.1 OnCE PAB Fetch Register (OPABFR)

The OnCE PAB fetch register (OPABFR) is a read-only 16-bit latch that stores the address of the last instruction that was fetched before debug mode was entered. 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.

12.6.2 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, which is the program counter (PC) value. This is the instruction that 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.

12.6.3 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.

12.6.4 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 (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 a OPFIFO value.

12.6.5 OnCE PDB Register (OPDBR)

The OnCE PDB register (OPDBR) is a read/write 16-bit latch that stores the value of the program data bus (PDB) generated by the last program memory access of the DSP before debug mode is entered. OPDBR is available for read/write operations only 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.

Immediately upon entering debug mode, OPDBR should be read out via a OnCE command selecting OPDBR for read. This value should then be saved externally if the pipeline is to be restored (which 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 of 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 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 executes. During the execution, OS[1:0] = 00. Upon completion, OS[1:0] = 11 (debug mode). The user can poll JTAG IR to determine whether the instruction has completed. Typically, the period of time that OS[1:0] = 00 is unnoticeably small. By the time JTAG IR polls status and is read, OS[1:0] = 11. The only case in which this is not true is on chips with mechanisms to extend 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, JTAG IR 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 that r0 can be any of the r registers. The execution of other DSP instructions is possible, but only the preceding ones are specified and supported. Three-word instructions cannot be executed from debug mode.

12.6.6 OnCE PGDB Register (OPGDBR)

The OnCE PGDB register (OPGDBR) is a read-only 16-bit latch that stores 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.

The OPGDBR is a temporary register that stores the last value written to the PGDB. Executing `MOVE reg,X:$FFFF` loads the OPGDBR with the correct value. When the next DSP instruction is executed that modifies the PGDB after executing the `move reg,X:$FFFF` instruction, the OPGDBR holds the newly modified PGDB value. An example of an instruction that modifies the PGDB bus is any instruction that writes to any of the peripheral memory-mapped registers on a DSP chip.

The OPGDBR is available for read operations only through the JTAG/OnCE serial interface and only when the chip is in debug mode. Any attempted read of the OPGDBR when the chip is not in 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.

12.6.7 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 (peripheral) change-of-flow FIFO. Figure 12-10 illustrates a block diagram of the OnCE FIFO History Buffer.

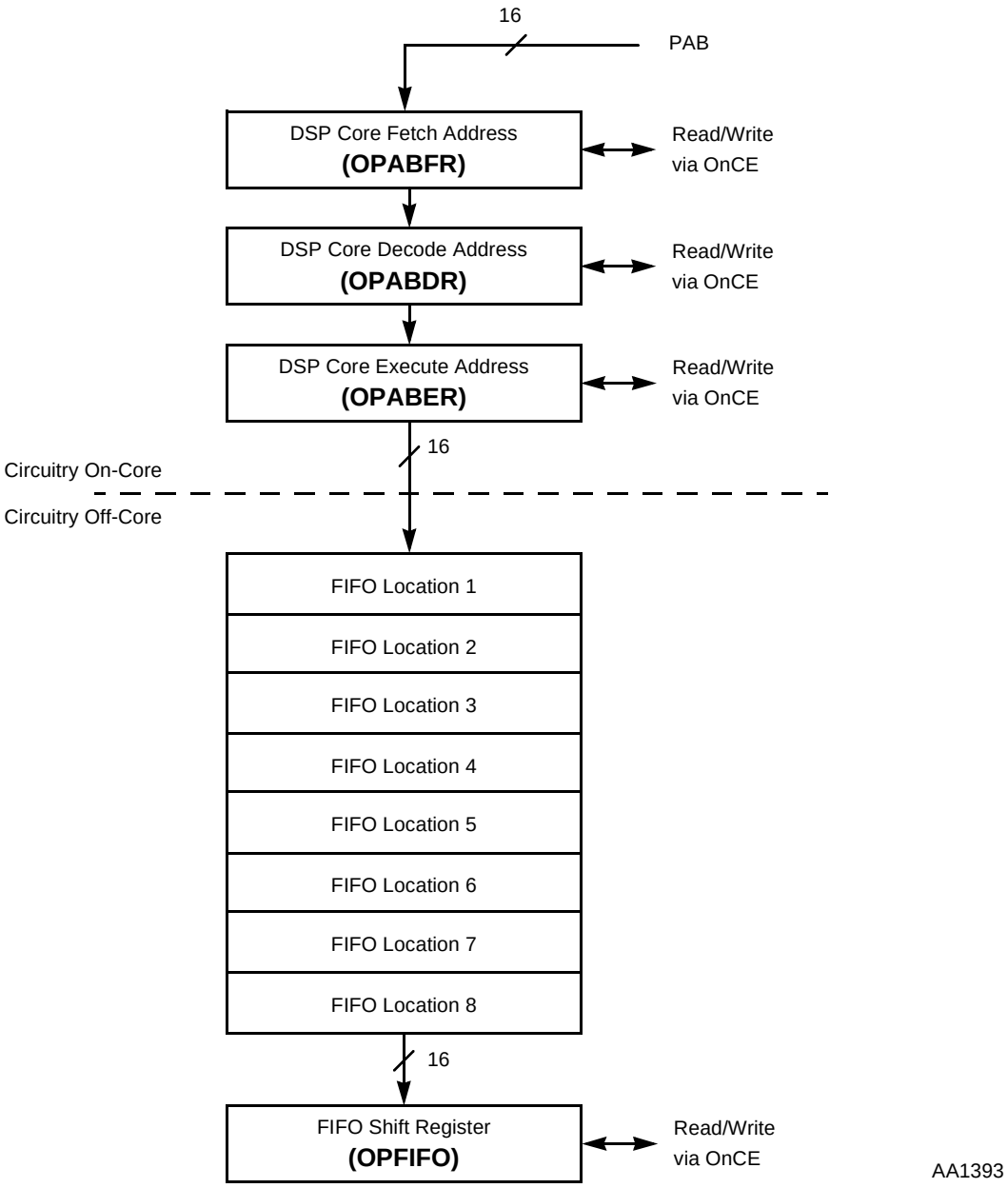


Figure 12-10. OnCE FIFO History Buffer

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 that all OPFIFO locations are read (that is, *n* reads for an *n*-deep OPFIFO) so that the oldest-to-newest ordering is maintained when address capture resumes.

The change-of-flow aspect of the FIFO begins *after* the OPABER and not the fetch, decode, or execute registers (OPABFR, OPABDR, or OPABER). 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. Transfers in progress can be interrupted, meaning that while determinate values are in the registers, these values may not provide entirely coherent information regarding the recent history of program flow.

In addition, the state of the OPFIFO can be different when it is halted due to an event occurring when EM = 01 than when it is halted with the core due to an event occurring when EM = 00.

12.7 Breakpoint 2 Architecture

All DSP56800 chips contain a breakpoint 1 unit. The DSP56824 provides a breakpoint 2 unit that allows greater flexibility in setting breakpoints. Adding a second breakpoint greatly increases the debug capability of the device. It allows the following additional breakpoints for the detection of more complex events:

- On either of two program memory breakpoints (that is, 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 DSP core and enter the debug processing state
- Interrupt the DSP core
- Halt the OnCE FIFO but let the DSP core continue operation
- Rearm the trigger mechanism (and toggle the $\overline{DE}$ pin)

The breakpoint 1 unit is used in conjunction with the breakpoint 2 unit for the detection of more complex trigger conditions. The breakpoint 1 unit is the same as found on other DSP56800 chips. The breakpoint 2 unit allows specifying more complex breakpoint conditions when used in conjunction with the first breakpoint. In addition, it is possible to set up breakpoint 2 for interrupts while leaving breakpoint 1 available to the JTAG/OnCE port. Note that when a breakpoint is set up on the CGDB with the breakpoint 2 unit, the breakpoint condition should be qualified by an X memory access with the breakpoint 1 unit.

Figure 12-11 shows how the two breakpoint units are combined in the breakpoint and trace counter unit to specify more complex triggers and 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.

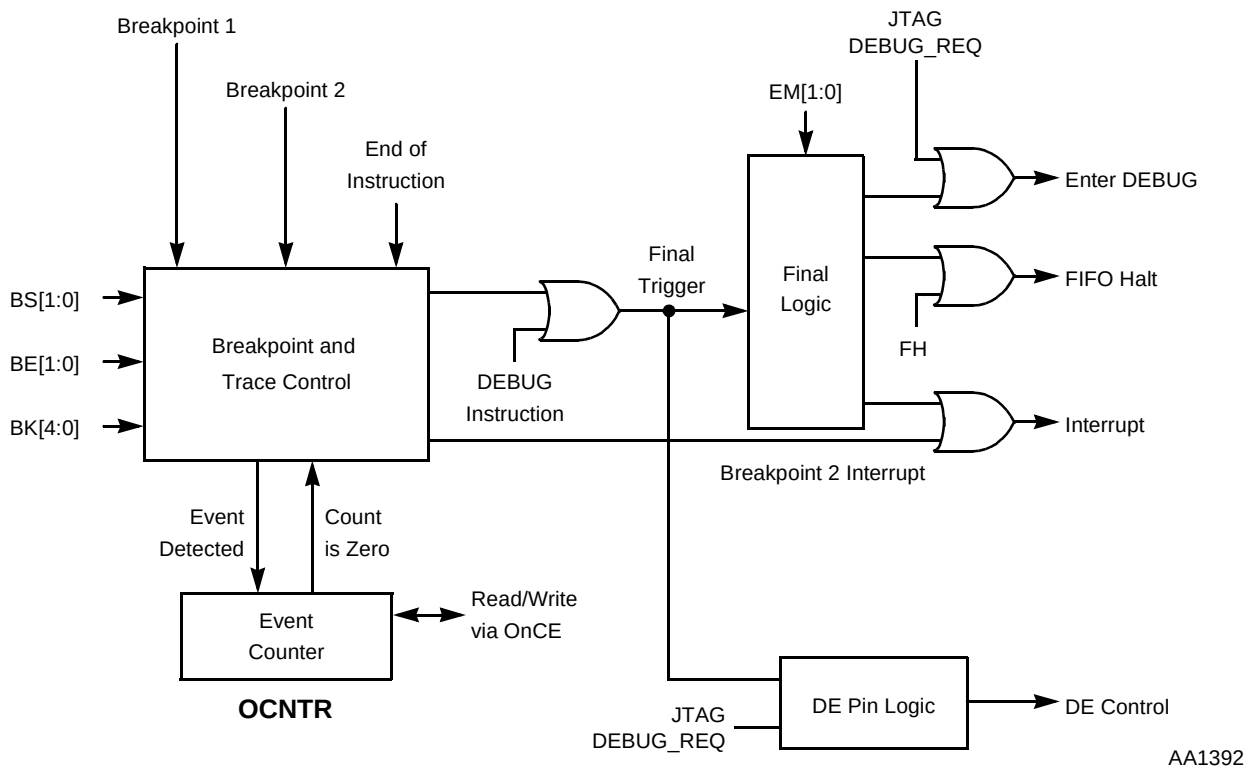


Figure 12-11. Breakpoint and Trace Counter Unit

12.8 Breakpoint Configuration

The breakpoint 1 unit is programmed in the OCR using the BS and BE bits. The breakpoint 2 unit is programmed by the OnCE breakpoint 2 control (OBCTL2) register, located within the breakpoint 2 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 action that is performed when a final trigger is detected is specified by the EM bits in the OCR. Figure 12-12 shows the breakpoint programming model for the dual breakpoint system.

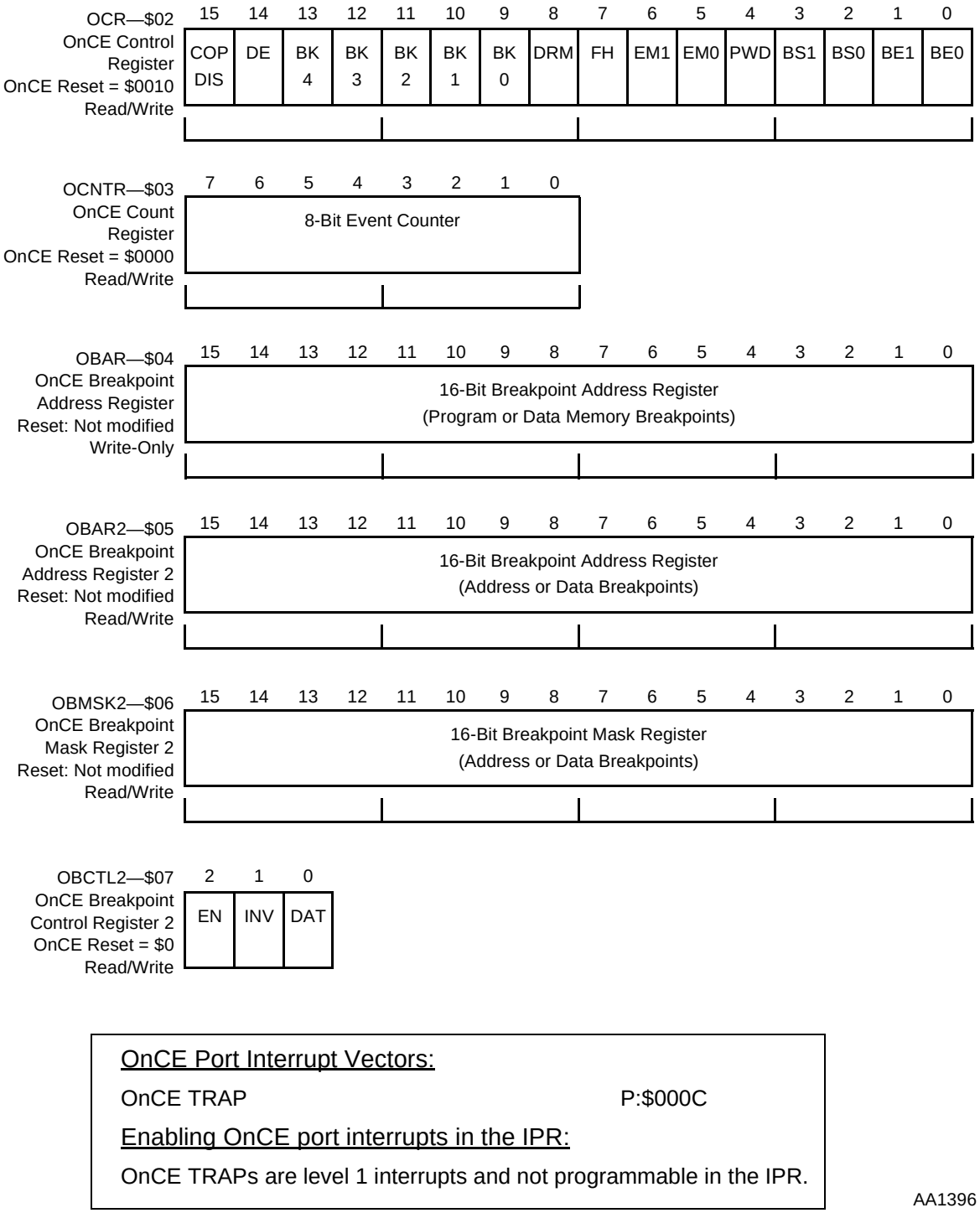


Figure 12-12. OnCE Breakpoint Programming Model

The breakpoint 1 unit's circuitry contains the OMAL, OBAR, OMAC, and 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 1 logic. Figure 12-13 illustrates a block diagram of the breakpoint 1 unit.

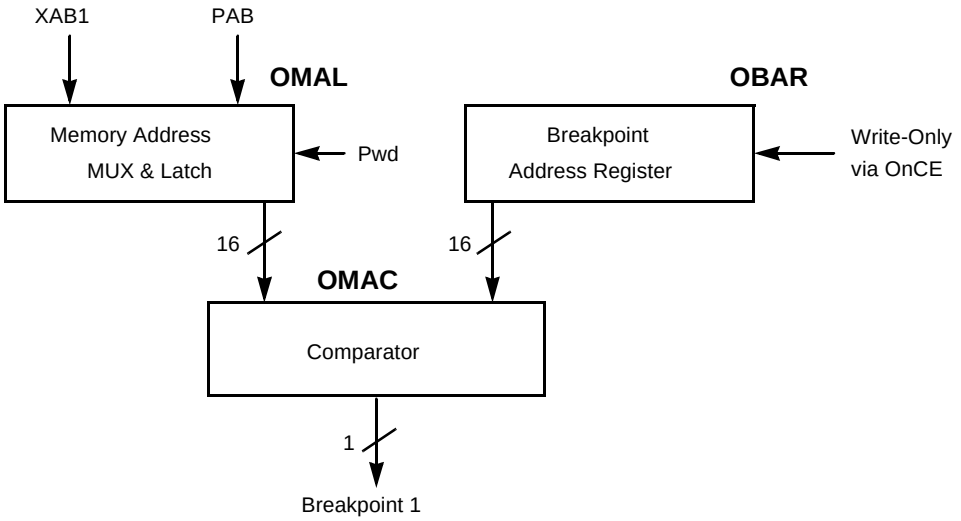


Figure 12-13. Breakpoint 1 Unit

For breakpoint 1, 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 1 can then do a few things based on BK encodings and the state of OCNTR. BK could dictate that the first valid address compare enables trace to decrement OCNTR. BK could also dictate that 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 (if enabled), and the EM bits determine whether to halt the core or only the FIFO.

Figure 12-14 provides a block diagram of the breakpoint 2 unit. This unit also has its own address register OBAR2 (similar to breakpoint 1's OBAR) and address comparator OMAC2 (similar to breakpoint 1's OMAC). If BE = 00, the breakpoint 2 unit is disabled. Other BE encodings and all BS encodings refer only to breakpoint 1 functionality. The BK encoding selects which, if any, breakpoint unit or combination of units (OR/AND) decrements OCNTR. The breakpoint 2 unit operates in a similar manner. A valid breakpoint 2 address compare 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).

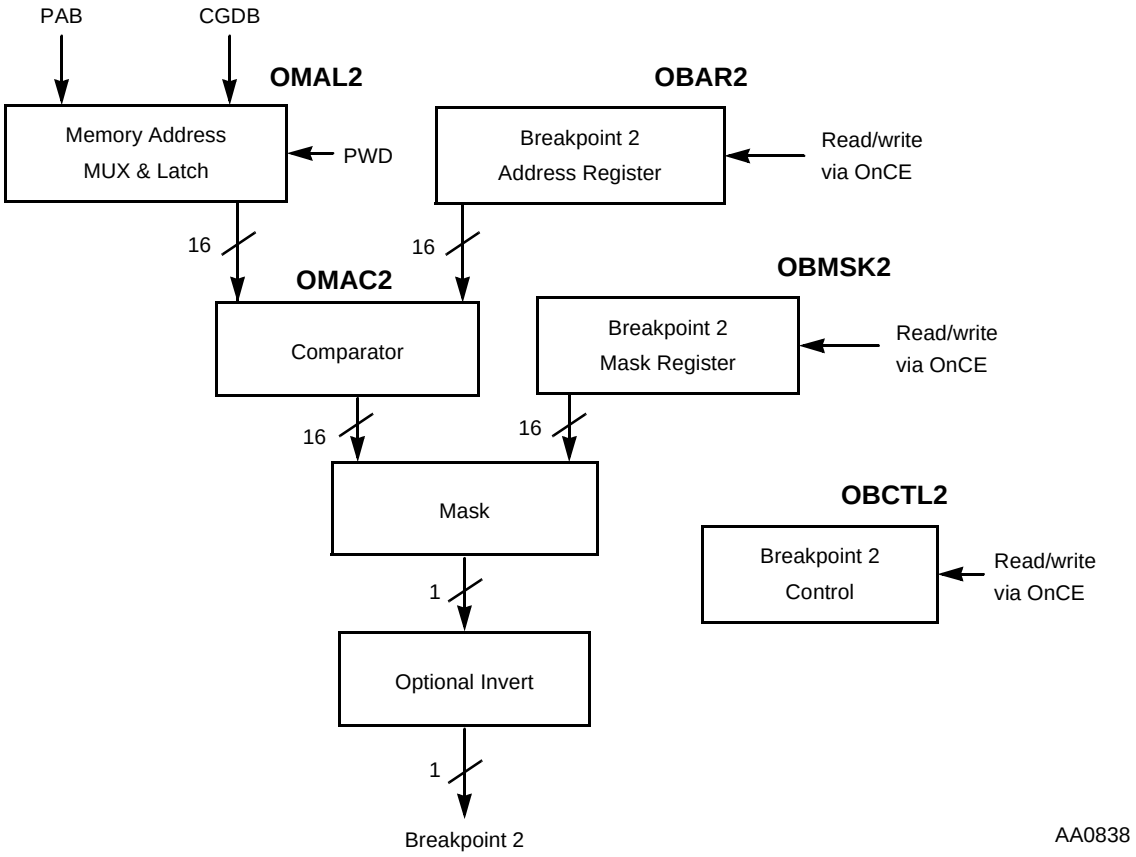


Figure 12-14. Breakpoint 2 Unit

A valid breakpoint 2 address compare 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 1 valid address compares to begin decrementing the OCNTR.
- A valid address compare when OCNTR = 0 can trigger a breakpoint 1 valid address compare to generate an 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 compare can trigger the first valid address compare on breakpoint 1 to allow trace to decrement the OCNTR.

NOTE:

When a breakpoint is set up 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 the user to define the conditions that determine a valid breakpoint. Using these bits, breakpoints could be restricted to occur on first data memory reads or only on fetched instructions that are executed. Again, these bits pertain only to breakpoint 1. See Section 12.4.4.8, “Breakpoint Selection (BS[1:0])—Bits 3–2,” and Section 12.4.4.9, “Breakpoint Enable (BE[1:0])—Bits 1–0,” to understand various encodings.

Perhaps the most important breakpoint capability is the ability to break on up to two program memory locations, on a sequence where a first found breakpoint is followed sequentially by a second breakpoint, on a specified data memory location when a programmed value is read or written as data to that location, and 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 currently available in the OnCE module:

- Halt the DSP core and enter the debug processing state.
- Interrupt the DSP core.
- Halt the OnCE FIFO, but let the DSP core continue operation.
- Rearm the trigger mechanism (and toggle the $\overline{DE}$ pin).

If a breakpoint is set on the last instruction in a DO loop (even if BS = 00 and 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.

12.8.1 Programming the Breakpoints

Breakpoints and trace can be configured while the core is executing DSP 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 user could hold the chip in reset; set OCNTR, OBAR, and OCR such that 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, allowing the user to gather statistical information.

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 reads and 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. The breakpoint address must be written into the OBAR.
4. The value $n - 1$ must be written into the OCNTR, where n is the number of valid address compares that must take place before generating a OnCE event.
5. The OCR must be written to set the BE, BS, and BK bits for the desired breakpoint conditions, EM to choose what happens when an event occurs, and DE to enable or disable the DE pin.

If these steps are completed while in debug mode, debug mode must be exited to restart the core. If these steps are done in user mode, the breakpoint is set immediately. Note that OnCE events can occur even if ENABLE_ONCE is not latched in the JTAG IR. 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 2 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 action which is performed when a final trigger is detected is specified by the EM bits in the OCR.

12.8.2 OnCE Trace Logic Operation

The trace logic is tightly coupled with the breakpoint logic, 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 compare happens. For trace, the event occurs (TO is set) when OCNTR first reaches zero. If the EM bits are set for entry to debug mode, one more instruction is executed after TO is set. Therefore, if the user wants to halt the DSP 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 (that is, EM = 10 acts just like EM = 11 for trace).

Since trace events occur when OCNTR reaches zero and trace mode is enabled by one of the BK settings, rearming trace events acts 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 to disable trace (FIFO begins capturing).
2. Write OCNTR with the desired value.
3. Write OCR to enable trace.
4. Poll for TO = 1.

If the first step is omitted, TO is never reset and the FIFO does not begin capturing because the conditions for valid trace are still present.

Note that there are sequential breakpoints that enable trace mode. Their trace mode operation is identical to the BK[4:0] = 10111 operation, except that the HBO bit is set.

A common use of the trace logic is to execute a single instruction (OCNTR = 0) and then immediately return to 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.

12.9 The Debug Processing State

A DSP56800 chip 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 are referenced in Chapter 7, “Interrupts and the Processing States,” in the *DSP56800 Family Manual*. The last processing mode, the debug mode, is described in this subsection.

The debug mode supports the on-chip emulation features of the chip. In this mode, the DSP core is halted and is set to accept OnCE commands through the JTAG port. Once the OnCE module is set up correctly, the DSP leaves the debug mode and returns control to the user program. The DSP reenters the debug mode when the previously set trigger condition occurs, provided that EM = 00 (OnCE events cause entry to debug mode).

Capabilities available in the debug mode include the following:

- Reading and writing the OnCE registers
- Reading the instruction FIFO
- Resetting the OnCE event counter
- Executing a single DSP instruction and returning to this mode
- Executing a single DSP instruction and exiting this mode

12.9.1 OnCE Normal, Debug, and Stop Modes

The OnCE module has three operational modes: normal, debug, and stop. Whenever a STOP instruction is executed by the DSP, the OnCE module is no longer accessible. The OnCE module is in the normal mode except when the DSP enters the debug mode or is in stop mode. The OnCE module is in debug mode whenever the DSP enters the debug mode. The major difference between the states is register access. The following OnCE module registers can be accessed in normal or debug mode:

- 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. The user can recognize the occurrence by capturing the OS bits in the JTAG IR in Capture-IR. The user can then choose to send a DEBUG_REQUEST to bring the core out of STOP.

12.9.2 Entering Debug Mode

There are seven ways to enter debug mode:

- JTAG DEBUG_REQUEST during hardware reset
- JTAG DEBUG_REQUEST during Stop or Wait
- JTAG DEBUG_REQUEST during wait states
- Software breakpoint (DEBUG) during normal activity with PWD = 0 and EM = 00
- Trigger events (breakpoint and trace modes) when EM = 00
- DSP instruction executed from debug mode with EX = 0
- External $\overline{DE}$ request pin assertion

NOTE:

The $\overline{DE}$ pin, when asserted, causes the DSP to finish the current instruction being executed, save the instruction pipeline information, enter debug mode, and wait for commands to be entered from the JTAG/OnCE serial input line. A request from the $\overline{DE}$ pin is treated the same way as a JTAG DEBUG_REQUEST.

The status bits provide information about the chip status when the debug mode cannot be entered in response to an external request to enter it. Table 12-14 shows the status of the chip as a function of the two status bits OS[1:0].

Table 12-14. Function of OS[1:0]

OS[1:0]	Status
00	Normal mode
01	Stop or wait mode
10	DSP busy state (external accesses with wait state)
11	Debug mode

12.9.2.1 JTAG DEBUG_REQUEST and the Debug Event Pin

The core reacts in the same way to a debug request by a JTAG DEBUG_REQUEST or the assertion of the debug event ($\overline{DE}$) pin. To send a JTAG DEBUG_REQUEST, the 0111 opcode must be shifted into the JTAG IR, and then Update-IR must be passed through. This instructs the core to halt and enter debug mode. Asserting the debug event ($\overline{DE}$) pin produces the same effect. When the DSP enters debug mode in response to these requests, the $\overline{DE}$ pin is asserted (pulled low) if it is enabled.

The JTAG/OnCE interface is accessible when $\overline{RESET}$ is asserted, provided that $\overline{TRST}$ is not asserted (that is, the JTAG port is not being reset). The user can load DEBUG_REQUEST into the JTAG IR while $\overline{RESET}$ is held low. If $\overline{RESET}$ is then deasserted, the chip exits hardware reset directly into debug mode. After sending the DEBUG_REQUEST instruction, the user should poll the JTAG IR to see whether the chip has entered debug mode.

If the chip is in either wait or stop mode, either type of debug request, much like an external interrupt, brings the chip out of these modes. Upon leaving wait or stop mode, the chip enters debug mode. As always, the user should poll JTAG IR for status after sending the debug request. It is important to remember that the OSR cannot be polled during stop mode, since 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 the 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 JTAG IR, meaning that 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 normal mode results in the chip entering debug mode as soon as the instruction currently executing finishes. Again, the JTAG IR should be polled for status. See Section 13.2, “JTAG Port Architecture,” on page 13-2 for information about using the JTAG TAP controller and its instructions.

12.9.2.2 Software Request During Normal Activity

Upon executing the DEBUG instruction, the chip enters the debug processing state provided that PWD = 0 and EM = 00.

12.9.2.3 Trigger Events (Breakpoints and Trace Modes)

The DSP56824 allows the user to configure 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 or trace:

- EM = 00.
- If OCNTR = 0, breakpoints are enabled (BE is not 00) and 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) and the next instruction executed causes the core to halt.

12.9.2.4 Reentering Debug Mode with EX = 0

If a DSP instruction is executed from debug mode with EX = 0, debug mode is automatically reentered after the instruction finishes executing. When the instruction is being executed, the core is not in debug mode, and 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 JTAG IR is recommended to guarantee that the chip is in debug mode.

12.9.2.5 Exiting Debug Mode

There are three ways to exit debug mode:

- Restore the pipeline by writing the 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 a `jmp` opcode to OPDBR with GO = EX = 0 and then writing the target address to OPDBR with GO = 1 and EX = 0. Next, write a NOP to OPDBR with GO = EX = 1.
- Apply a hardware reset (assert $\overline{\text{RESET}}$), which brings the chip out of debug mode provided DEBUG_REQUEST is not decoded in the JTAG IR.

12.10 Accessing the OnCE Module

This section describes useful example sequences involving the JTAG/OnCE interface. The sequences are described in a hierarchical manner. Low-level sequences describe basic operations (for example, 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.

12.10.1 Primitive JTAG Sequences

The JTAG/OnCE serial protocol is identical to the protocol described in the *IEEE Standard Test Access Port and Boundary-Scan Architecture* (1149.1a-1993). It involves the control of four 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. It is multiplexed with the DE output function.

The following descriptions refer to states in the JTAG state machine diagram in Figure 13-5 on page 13-13. Please refer to this diagram or to the IEEE 1149.1a-1993 document.

12.10.2 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 that TDO is disabled, no shifting is taking place, and the JTAG IR 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. In addition, 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 12-15. See the *DSP56824 Technical Data Sheet* for minimum assertion pulse widths. $\overline{\text{TRST}}$ can change at any time with respect to TCK.

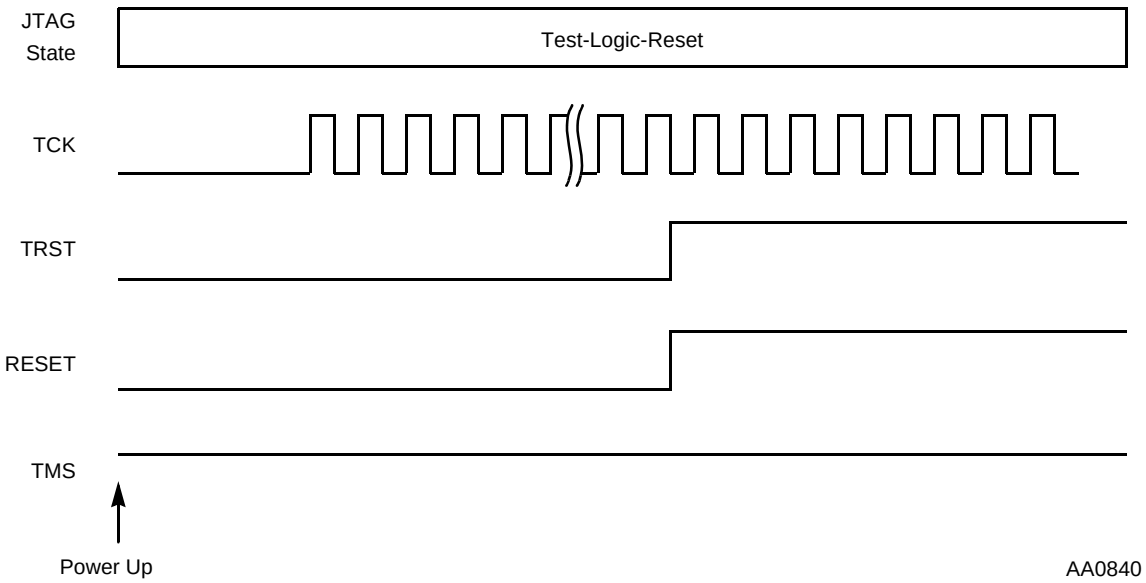


Figure 12-15. Entering the JTAG Test-Logic-Reset State

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 12-16. 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 the 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.

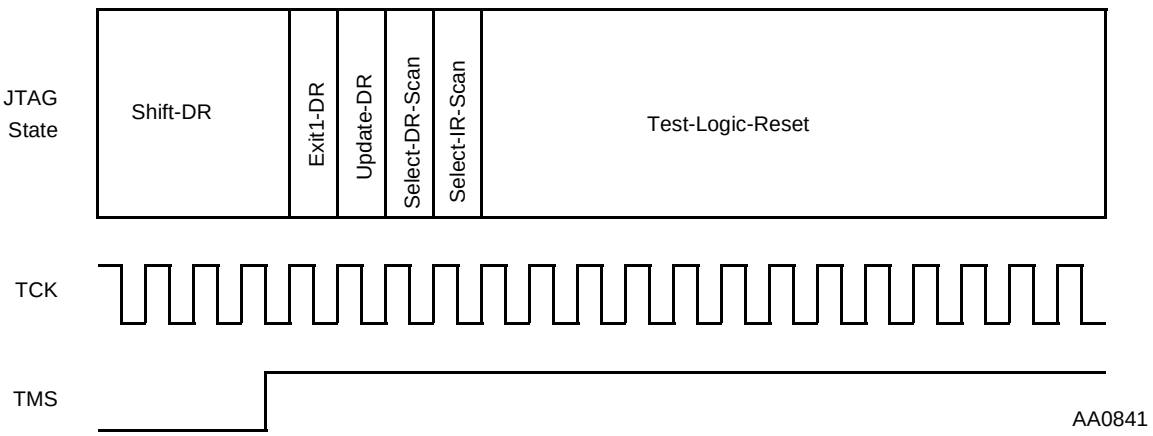


Figure 12-16. Holding TMS High to Enter Test-Logic-Reset State

12.10.3 Loading the JTAG Instruction Register

JTAG instructions are loaded in via the IR path in the state machine. Shifting takes place in the Shift-IR path, while the actual instruction register update occurs on Update-IR.

NOTE:

The bit order for JTAG/OnCE shifting is always as shown in Figure 12-17.

Figure 12-17. Bit Order for JTAG/OnCE Shifting

The following sequence shows how to load the instruction DEBUG_REQUEST into the JTAG IR, as shown in Figure 12-18.

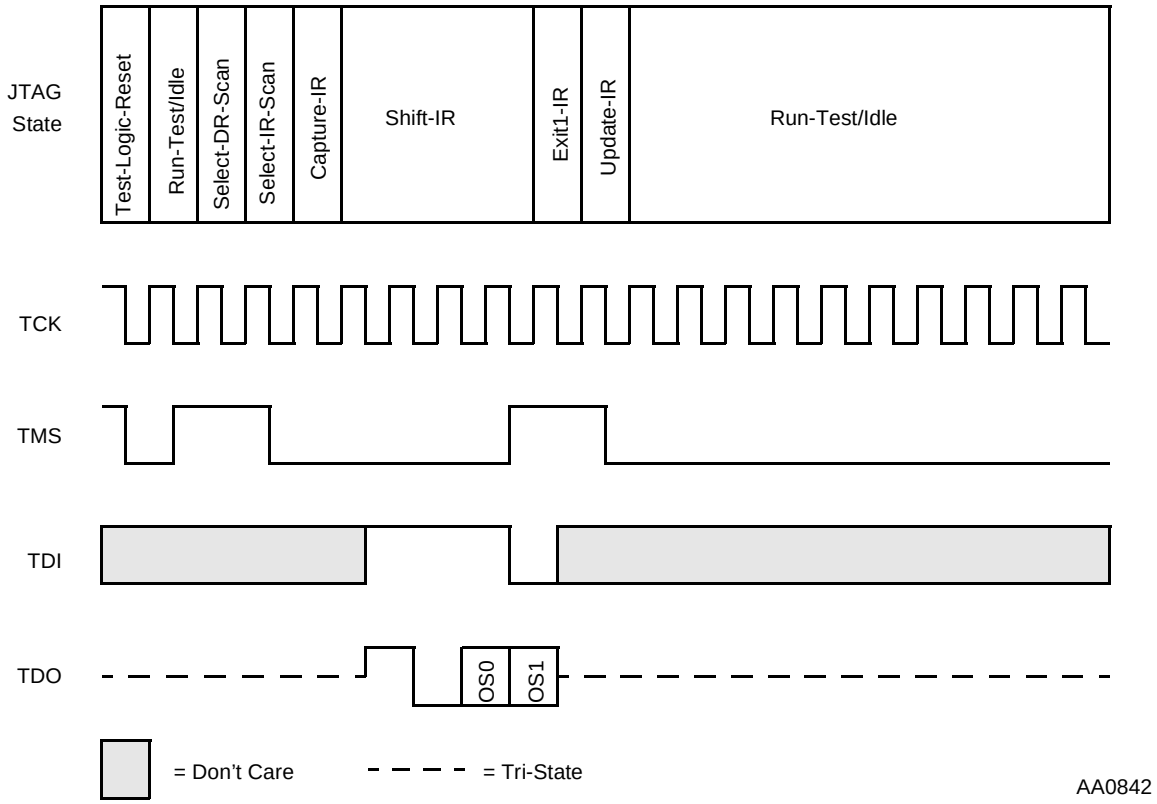


Figure 12-18. Loading DEBUG_REQUEST

During Shift-IR, a 4-bit shifter is connected between TDI and TDO. The opcode shifted in is loaded into the JTAG IR 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, and is last sampled on the TCK rising edge when entering

Exit1-IR. TDO changes on the falling edges of TCK in Shift-IR. It switches back to tri-state on the falling edge of TCK in Exit1-IR. The first 2 bits shifted out of TDO are constant: first 1, then 0. The following 2 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.

12.10.4 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 JTAG IR. Data is captured (if applicable) in the selected register on Capture-DR and 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 that BYPASS has been loaded into the JTAG IR and that the state machine is in the run-test/idle state. In BYPASS, a 1-bit register is selected as the data register. The following sequence shows how data can be shifted through the BYPASS register, as shown in Figure 12-19.

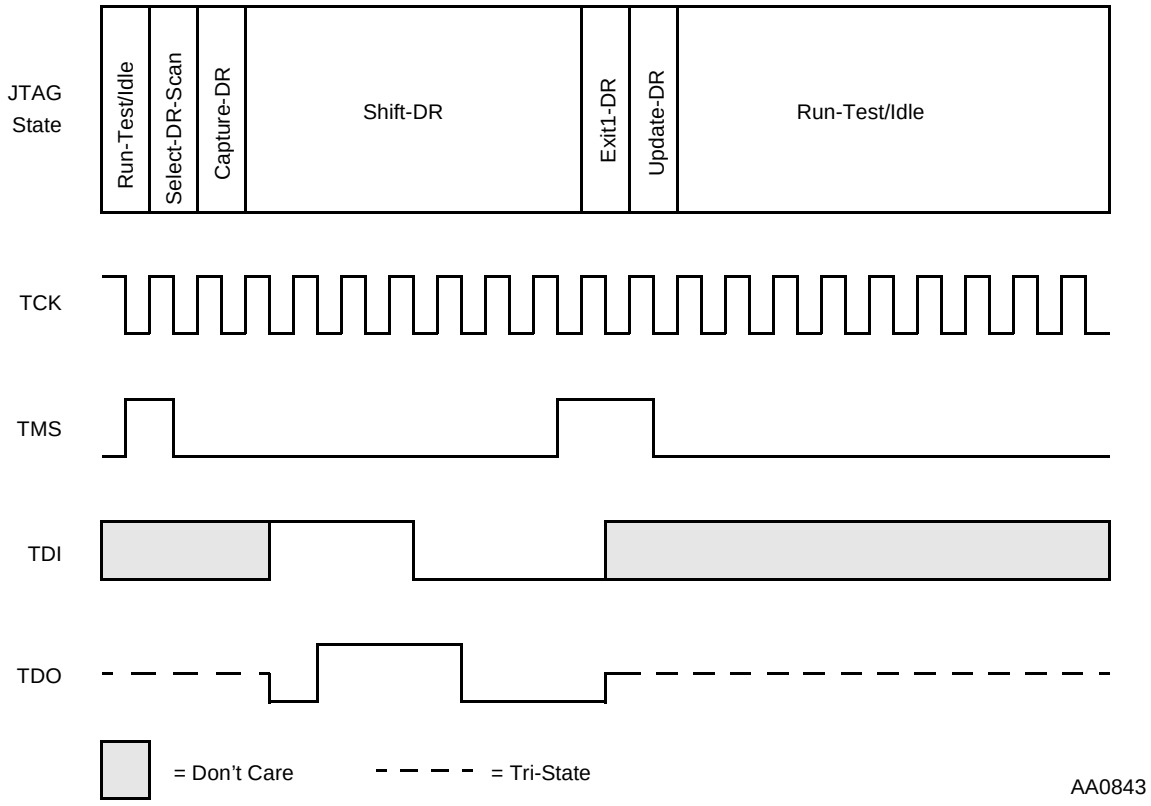


Figure 12-19. Shifting Data through the BYPASS Register

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.

12.10.4.1 JTAG/OnCE Interaction: Basic Sequences

JTAG controls the OnCE module via two basic JTAG instructions: DEBUG_REQUEST and ENABLE_ONCE. DEBUG_REQUEST provides a simple way to halt the DSP core. The halt request is latched in the OnCE module so that a new JTAG instruction can be shifted in without waiting for the request to be granted. After DEBUG_REQUEST has been shifted in, JTAG IR status polling takes place to see if the request has been granted. This polling sequence is described in Section 12.10.4.5, “JTAG IR 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 1-bit BYPASS register is selected. Values shifted into the BYPASS register have no effect on the OnCE logic.

ENABLE_ONCE is decoded in the JTAG IR 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 8 or 16 bits long. The shifter is 8 bits long for OCMDR and OSR accesses and 16 bits long for all other register accesses. This means that 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 then loaded into the OCMDR has a 16-bit read or 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 in Figure 12-20.

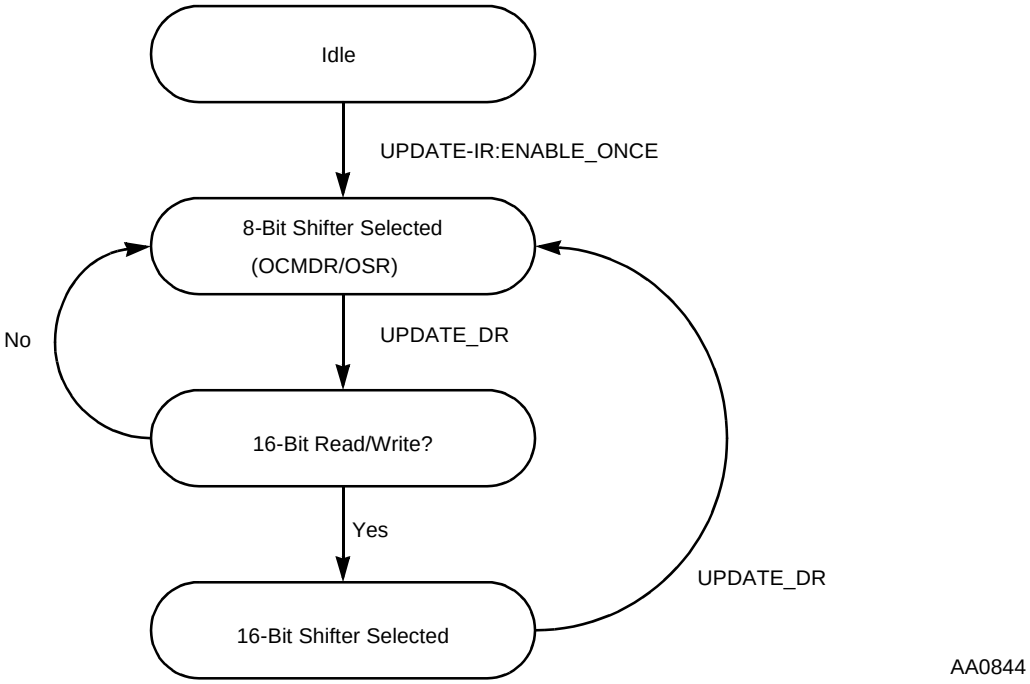


Figure 12-20. OnCE Shifter Selection State Diagram

As long as ENABLE_ONCE is decoded in JTAG IR, one of the two shifters is available for shifting. If a different JTAG instruction is shifted in, the BYPASS register is selected.

12.10.4.2 Executing a OnCE Command by Reading the OCR

The following sequence shows how to read the OCR, assuming that ENABLE_ONCE is being decoded in JTAG IR, the JTAG state machine is at run-test/idle, and the DR path has not yet been entered, meaning that the OnCE module has selected the 8-bit shifter. See Figure 12-21.

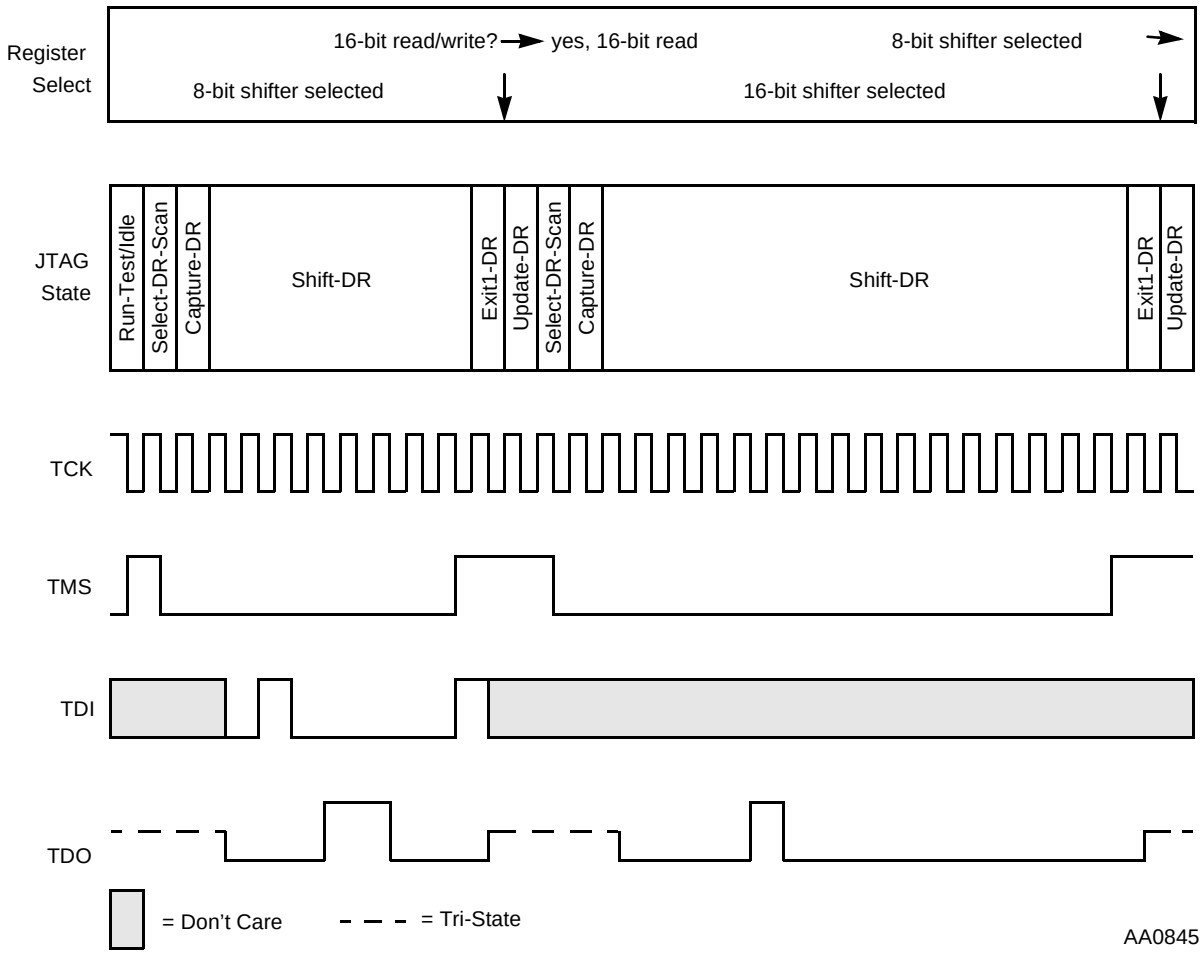


Figure 12-21. Executing a OnCE Command by Reading the OCR

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 that the DSP 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.

12.10.4.3 Executing a OnCE Command by Writing the OCNTR

In this sequence, the 8-bit OCNTR is written. First the Write OCR opcode is entered, followed by a 16-bit shift sequence, even though the OCNTR is only 8 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 setup is identical to the previous example. See Figure 12-22.

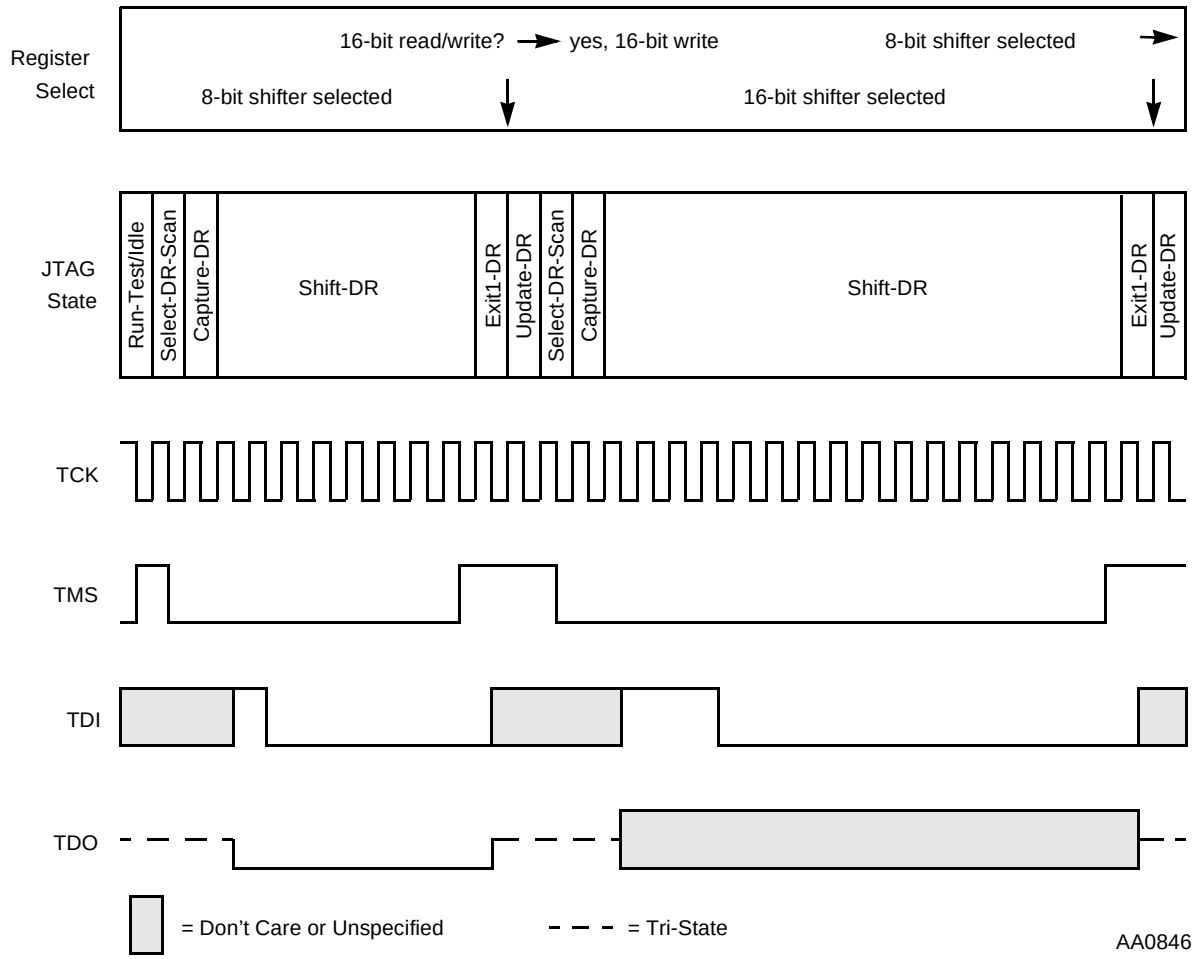


Figure 12-22. Executing a OnCE Command by Writing the OCNTR

In this sequence, $00 was read out from the OSR, indicating that the chip is in normal mode (the DSP core is running). The OnCE opcode shifted in is $01, which corresponds to Write OCNTR. In the ensuing 16-bit shift, $0003 is shifted in. Since the OCNTR is only 8 bits wide, it loads the 8 least significant bits, or $03. The bits coming out of TDO are unspecified during 16-bit writes.

12.10.4.4 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 that a breakpoint has been set up to halt the core. See Figure 12-23.

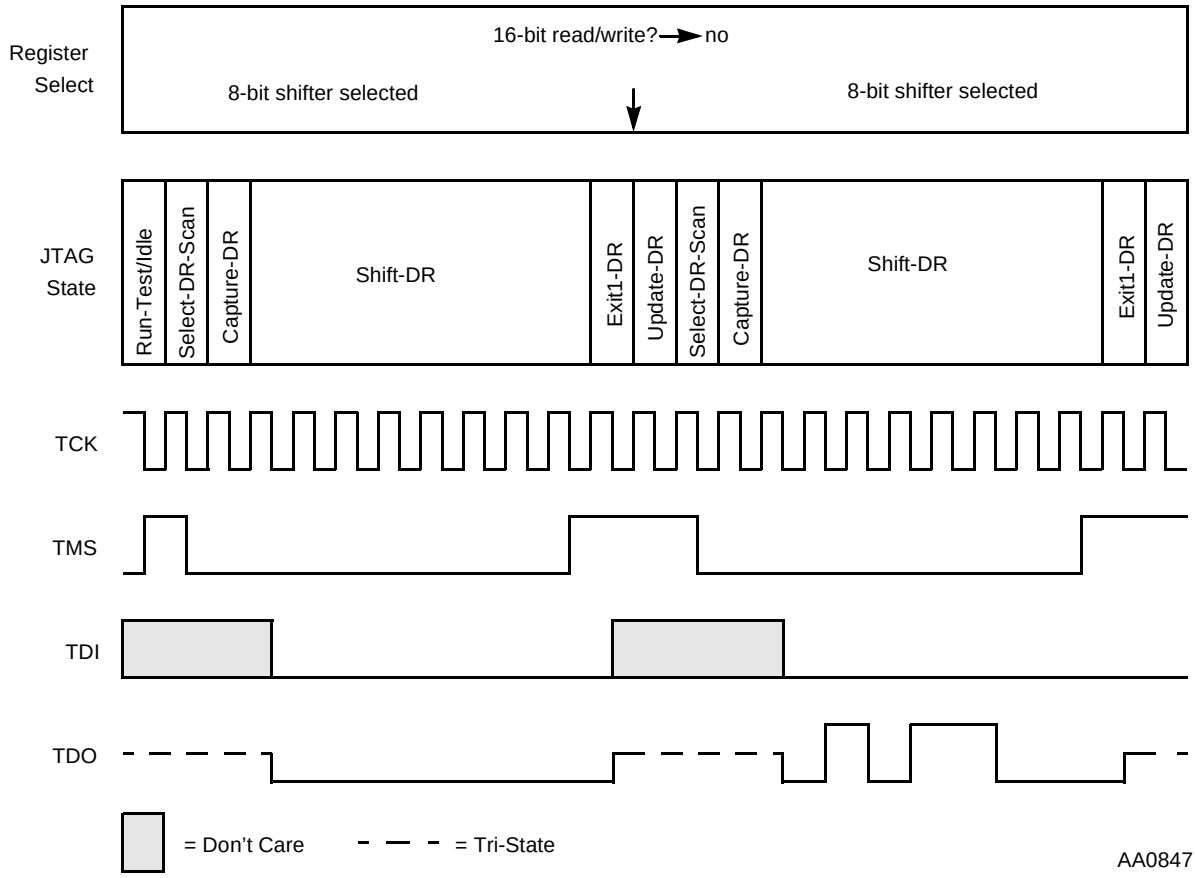


Figure 12-23. OSR Status Polling

On the first 8-bit sequence, $00 is shifted into the OCMDR, which corresponds to no register selected. Next, $00 is read out from the OSR, indicating the DSP 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, indicating that the chip is in debug mode and a hardware breakpoint has occurred.

12.10.4.5 JTAG IR Status Polling

OSR status polling has some disadvantages. First, the OSR is not accessible when the DSP 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. Second, 8-bit shifts are required. Polling the JTAG IR provides a more efficient and more reliable means of gathering status information. The following sequence shows how the JTAG IR can be polled. Again, assume a breakpoint has been set up to halt the core. See Figure 12-24.

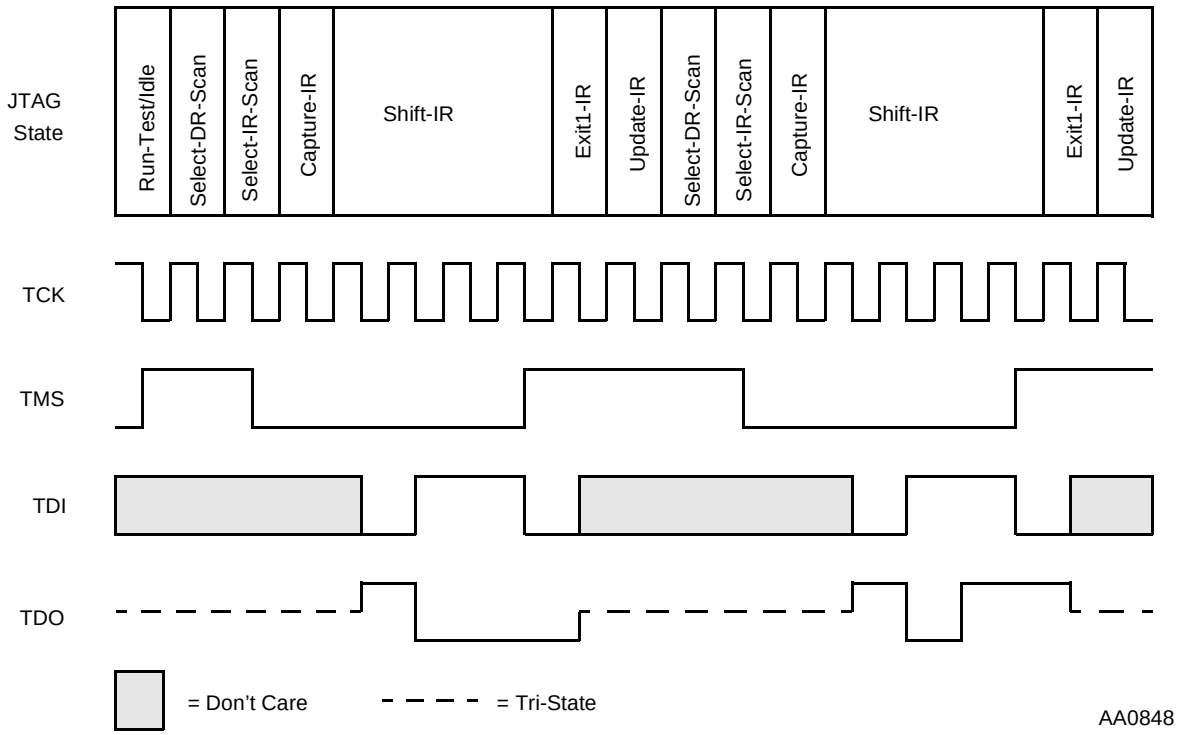


Figure 12-24. JTAG IR Status Polling

On the first 4-bit shift sequence, $6 (ENABLE_ONCE) is shifted into the JTAG IR. The first 2 bits coming out of TDO are the standard constants, and the last two are output shifter (OS) bits. OS is 00, indicating that the DSP is in normal mode. On the second 4-bit shift sequence, ENABLE_ONCE is again shifted in. The OS bits are now 11, indicating that the chip is in debug mode. ENABLE_ONCE does not have to be shifted in for JTAG IR polling. After reading proper status, the DR path can be entered directly for OnCE register accesses.

12.10.4.6 $\overline{\text{TRST}}/\overline{\text{DE}}$ Pin Polling

The $\overline{\text{TRST}}/\overline{\text{DE}}$ pin can also be used to provide information about the processor state. When the DE output function is enabled, $\overline{\text{TRST}}/\overline{\text{DE}}$ goes low upon entry into debug mode. The pin is not released until debug mode is exited.

12.10.5 Serial Protocol Description

In order to permit an efficient means of communication between the command controller and the DSP chip, the following protocol has been adopted. Before starting any debugging activity, the command controller has to wait for an acknowledge from the chip that informs the command controller that it has entered the debug mode.

NOTE:

In case of a breakpoint, trace, or software DEBUG/DEBUGcc instruction, the acknowledge itself initiates the debug session. The command controller communicates with the chip by sending 8-bit commands that may be accompanied by 16-bit data. After sending a command, the command processor starts waiting for the chip to acknowledge execution of the command. The command processor may send a new command only after the chip has acknowledged execution of the previous command.

12.10.5.1 Entering Debug Mode from User Mode

When shifting in the ENABLE_ONCE command, the OS bits will be shifted out so the user can determine if the core has acknowledged the request and halted. To enter the debug mode from the JTAG port, the user must issue the JTAG DEBUG_REQUEST instruction and then enter the ENABLE_ONCE instruction to begin programming the OnCE module.

12.10.5.2 Entering Debug Mode from DSP Reset

The JTAG state machine will be reset via a POR signal. JTAG will therefore be accessible soon after the DSP powers up. The OnCE state machine will be placed at the IDLE state at reset assertion by a pulse. The pulse would be valid for only a few cycles just after reset assertion, allowing for access to the OnCE state machine when the DSP is still in reset. The user can then execute the JTAG instruction DEBUG_REQUEST followed by ENABLE_ONCE. After de-asserting DSP reset, the chip will be in debug mode.

NOTE:

Providing OnCE access in DSP reset allows the user to set breakpoints while in reset. This aids in debugging when the DSP has problems leaving reset.

12.10.5.3 Polling for OnCE Status

It is necessary to poll for OnCE status in the following three situations:

- After loading the JTAG instruction DEBUG_REQUEST
- After issuing a DSP instruction while in debug mode
- When breakpoints are enabled in user mode

In these situations the user must have some way of knowing if the core has halted. Since the TDO pin (which replaces the existing DSO pin) can no longer provide the acknowledge pulse, it has been proposed that the new OSR contain the OS[1:0] bits (as well as the JTAG IR on Capture-IR) that provide this sort of status information.

NOTE:

If the core is executing a STOP or WAIT instruction, the OSR will not be readable (loss of internal clocks) and status must be read via the JTAG IR. Alternately, the user can use the DE output to indicate that a debug event has occurred and that the OnCE module needs servicing.

The user can poll the OSR in debug or user mode provided that the OnCE state machine is in the STATCOM state and the JTAG state machine is in Shift-DR. The OSR will capture new status only when traversing to the STATCOM state from a state other than STATCOM itself; see Section 12.3.3, “OnCE State Machine and Control Block.”

12.10.5.4 Setting Breakpoints in User Mode

Setting breakpoints in user mode is done the same way as in debug mode, except that it is recommended that the user disable breakpoints before writing to the OMAC, OBAR, or the BK4–BK0 bits in OCR. This avoids generating any indeterminate results that could arise if a breakpoint is occurring while the preceding registers are being modified.

12.10.5.5 Reading Pipeline Information in User Mode

The user can access pipeline information while the program is still executing. One way of doing this is for the user to follow these steps:

1. Make sure breakpoints and trace are disabled.
2. Write 01 to the EM bits in OCR.
3. Set the OCNTR to 1.
4. Set BK4-BK0 in the OCR to 10111 to enable trace mode.
5. Poll the OSR until TO equals 1.
6. Read the FIFO.

12.10.5.6 Displaying a Specified Register

The multiple DSP configuration shown in Figure 12-25 is used for Example 12-1. The user must assume before the procedure begins that all three DSPs are running and that their JTAG ports are executing the BYPASS instruction. The user wishes to display the value of a register (“reg”) of DSP#2.

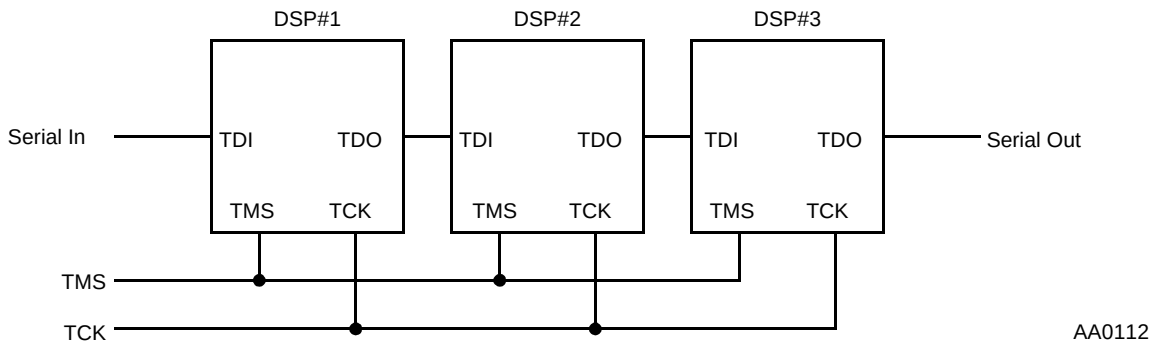


Figure 12-25. Multiple DSP Configuration for Example Procedures

NOTE:

"Shift-IR," "Shift-DR," "Update-IR," and "Update-DR" in the following example refer to operating states in the JTAG module. The TAP controller in the JTAG module uses the current operating state of the TAP controller combined with the level of the TMS input to transition between the operating states. A detailed description of the TAP controller state machine is presented in Section 13.3, "TAP Controller."

BYPASS, DEBUG_REQUEST, and ENABLE_ONCE are JTAG instructions. (See Table 13-2 on page 13-5 for a list of JTAG instructions.)

Example 12-1. Display a Specified Register—Serial

1. Enter Shift-IR, shift in BYPASS to DSP#1 and DSP#3, and DEBUG_REQUEST to DSP#2 (12 clocks in Shift-IR). Pass through Update-IR. OnCE state machine of DSP#2 is in STATCOM state.
2. Enter Shift-IR, ENABLE_ONCE into DSP#2 and BYPASS into others. Pass through Update-IR. OnCE state machine is in STATCOM state.
3. Enter Shift-DR. Begin polling to see if DSP#2 has entered debug mode (or poll via JTAG IR if STOP or WAIT may be executing). When status information indicates the core has halted and is in debug mode, enter Shift-DR, shift in `WRITE OPDBR with GO and no EX` (OnCE command = 01001001). Pass through Update-DR. OnCE state machine is in WPDBR state.
4. Enter Shift-DR. Send the 16-bit opcode: `MOVE reg, x:OGDB` (17 clocks in Shift-DR). Pass through Update-DR to actually write the OPDBR and begin execution of the MOVE instruction. OnCE state machine is in STATCOM state to allow for polling.
5. Enter Shift-DR and begin polling for OS[1:0] = 11 again. While polling, shift in `READ OPGDBR` (OnCE command = 11001000). When OS[1:0] = 11, pass through Update-DR. In this case, polling really would not be necessary unless internal wait states are inserted. If they are, the user should reduce JTAG clock speed enough such that polling still is not necessary. OnCE state machine is in RWREG state in preparation for reading of OPGDBR.
6. Enter Shift-DR. Clock 17 times to display the value in the OGDBR (that is, register contents). Pass through Update-DR. OnCE state machine is in STATCOM state in preparation for further OnCE commands to be shifted in.

12.10.5.7 Displaying X Memory Area Starting at Address xxxx

For Example 12-2, the user should assume the multiple DSP configuration shown in Figure 12-25 on page 12-50. Before the procedure begins, all three DSPs are in user mode and their JTAG ports are executing the BYPASS instruction. The user wishes to display memory contents in DSP#2.

NOTE:

"Shift-IR," "Shift-DR," "Update-IR," and "Update-DR" in the following example refer to operating states in the JTAG module. The TAP controller in the JTAG module uses the current operating state of the TAP controller combined with the level of the TMS input to transition between the operating states. A detailed description of the TAP controller state machine is presented in Section 13.3, "TAP Controller."

BYPASS, DEBUG_REQUEST, and ENABLE_ONCE are JTAG instructions. (See Table 13-2 on page 13-5 for a list of JTAG instructions.)

Example 12-2. Display X Memory Area from Address xxxx—Serial

1. Perform steps 1–6 of the procedure in Example 12-1 on page 12-51; the register in this case is R0. Since R0 will be used in displaying the memory area, its original value should be saved and later restored just before the debug mode is left. OnCE state machine is in STATCOM state.
2. Enter Shift-DR, shift `WRITE OPDBR with no GO and no EX` (OnCE command = 00001001) into DSP#2 (9 clocks in Shift-DR). Pass through Update-DR. OnCE state machine is in WPDBR state.
3. Enter Shift-DR. Send the 16-bit opcode of the two-word DSP instruction: `MOVE #$xxxx, R0` (17 clocks in Shift-DR). Pass through Update-DR to actually write the OPDBR. OnCE state machine is in STATCOM state.
4. Enter Shift-DR, shift `WRITE OPDBR with GO and no EX` (OnCE command = 01001001) into DSP#2 (9 clocks in Shift-DR). Pass through Update-DR. OnCE state machine is in WPDBR state.
5. Enter Shift-DR. Send the 16-bit data of the two-word DSP instruction: `MOVE #$xxxx, R0` (the xxxx field). Pass through Update-DR to actually write the OPDBR and begin execution of the MOVE instruction. OnCE state machine is in STATCOM state to allow for polling.
6. Enter Shift-DR and begin polling for OS[1:0] = 11. While polling, shift in `WRITE OPDBR with no GO and no EX` (OnCE command = 00001001). When OS[1:0] = 11, indicating that the MOVE instruction has completed and the core has halted once again, R0 has been loaded with the base address of the memory area to be displayed. Pass through Update-DR. OnCE state machine is in WPDBR state.
7. Enter Shift-DR. Send the 16-bit opcode: `MOVE X:(R0)+, x:OGDB` (17 clocks in Shift-DR). Pass through Update-DR to actually write the OPDBR and begin execution of the MOVE instruction. OnCE state machine is in STATCOM state to allow for polling.
8. Enter Shift-DR and begin polling for OS[1:0] = 11. While polling, shift in `READ OPDBR` (OnCE command = 11001001). When OS[1:0] = 11, the MOVE instruction has completed, the core has halted once again, the contents of the first memory location have been placed in OGDBR, and R0 has been incremented such that it points to the next memory location. Pass through Update-DR. OnCE state machine is in RWREG state.

Example 12-2. Display X Memory Area from Address xxxx—Serial (Continued)

9. Enter Shift-DR. Clock 17 times to display the value in the OGDBR (that is, the contents of memory location xxxx). Pass through Update-DR. OnCE state machine is in STATCOM state.
10. Enter Shift-DR. Shift in `NO SELECTION with GO and no EX` (OnCE command = 11000000). Pass through Update-DR to execute `MOVE X:(R0)+, x:OGDB` again. Go to step eight to continue reading memory contents. When finished, restore original value of R0.

12.10.5.8 Returning from Debug Mode to Normal Mode

For Example 12-3, the user should assume the multiple DSP configuration shown in Figure 12-25 on page 12-50. The user wishes to bring DSP#2 back into user mode from debug mode. DSP#1 and DSP#3 are in user mode and their JTAG ports are executing the BYPASS instruction.

NOTE:

"Shift-IR," "Shift-DR," "Update-IR," and "Update-DR" in the following example refer to operating states in the JTAG module. The TAP controller in the JTAG module uses the current operating state of the TAP controller combined with the level of the TMS input to transition between the operating states. A detailed description of the TAP controller state machine is presented in Section 13.3, "TAP Controller."

NOTE:

BYPASS, DEBUG_REQUEST, and ENABLE_ONCE are JTAG instructions. (See Table 13-2 on page 13-5 for a list of JTAG instructions.)

Example 12-3. Return from Debug Mode to Normal Mode—Serial

1. Enter Shift-DR, shift `WRITE OPDBR with no GO and no EX` (OnCE command = 00001001) into DSP#2 (9 clocks in Shift-DR). Pass through Update-DR. PAB is driven with value in OPABDR, which is the value latched from the PAB just before entering debug mode. OnCE state machine is in WPDBR state.
2. Enter Shift-DR. Send the saved contents of the OPDBR (17 clocks in Shift-DR). Pass through Update-DR to actually load the OPDBR. OPDBR will drive the PDB, and the instruction latch loads the value on the PDB. OnCE state machine is in the STATCOM state.
3. Enter Shift-DR, shift `WRITE OPDBR with GO and EX` (OnCE command = 01101001) into DSP#2 (9 clocks in Shift-DR). Pass through Update-DR. OnCE state machine is in WPDBR state.
4. Enter Shift-DR. Send the saved contents of the OPDBR (17 clocks in Shift-DR). Pass through Update-DR to actually load the OPDBR. The OnCE state machine goes into the IDLE state then back to STATCOM as long as ENABLE_ONCE is executing in JTAG machine. DSP#2's core pipeline is restored to its state prior to debug, and the DSP is back in user mode.

12.10.5.9 Recovering from STOP or WAIT Execution

If a STOP or WAIT instruction is executed while the user is accessing OnCE in user mode, problems will occur since few or no internal clocks are running anymore. This possibility should be avoided; the user can recognize the occurrence by capturing the OS bits in the JTAG IR in Capture-IR. The user can then choose to send a DEBUG_REQUEST to bring the core out of STOP or WAIT.

12.11 Using the OnCE Port

Following are example OnCE port command sequence for performing common debugging tasks. Entering and exiting debug mode, reading registers and memory, and restarting execution at a new address are discussed.

12.11.1 Beginning Debug Activity

Debug activity begins on an instruction boundary after the debug request pin is asserted, a DEBUGcc opcode is executed, a trigger event occurs, or a JTAG DEBUG_REQUEST is executed. If the instruction executing when the debug request pin is asserted is a REP instruction or the instruction following a REP instruction, then the debug activity begins after the instruction following the REP instruction finishes its repetitions. The first ACK indicates that the OnCE controller is ready to receive commands and data. Most of the debug activities will have the beginning specified in Example 12-4.

Example 12-4. Begin Debug Activity

Wait for acknowledge on $\overline{DS}$ line

1. Save pipeline information:
   a) Send command `Read OPDBR` (OnCE command = 11001001)
   b) Wait for acknowledge on $\overline{DS}$ line.
   c) Issue 16 clocks to read out data from selected register.
   d) Send command `READ OPGDR` (OnCE command = 11001000)
   e) Wait for acknowledge on $\overline{DS}$ line.
   f) Issue 16 clocks to read out data from selected register.
2. Read PAB FIFO and fetch/decode info (this step is optional):
   a) Send command `READ OPABFR` (OnCE command = 11001010)
   b) Wait for acknowledge on $\overline{DS}$ line.
   c) Issue 16 clocks to read out data from selected register.
   d) Send command `READ OPABDR` (OnCE command = 11010011)
   e) Wait for acknowledge on $\overline{DS}$ line.
   f) Issue 16 clocks to read out data from selected register.

Example 12-4. Begin Debug Activity (Continued)

g) Send command `READ OPFIFO` (OnCE command = 11010001)
h) Wait for acknowledge on $\overline{DS}$ line.
i) Issue 16 clocks to read out data from selected register.
j) Send command `READ OPFIFO` (OnCE command = 11010001)
k) Wait for acknowledge on $\overline{DS}$ line.
l) Issue 16 clocks to read out data from selected register.
m) Send command `READ OPFIFO` (OnCE command = 11010001)
n) Wait for acknowledge on $\overline{DS}$ line.
o) Issue 16 clocks to read out data from selected register.
p) Send command `READ OPFIFO` (OnCE command = 11010001)
q) Wait for acknowledge on $\overline{DS}$ line.
r) Issue 16 clocks to read out data from selected register.
s) Send command `READ OPFIFO` (OnCE command = 11010001)
t) Wait for acknowledge on $\overline{DS}$ line.
u) Issue 16 clocks to read out data from selected register.

12.11.2 Displaying a Specified Register

Example 12-5 shows the procedure for displaying a specified register.

Example 12-5. Display a Specified Register—OnCE

1. Send the command `WRITE OPDBR with GO and no EX` (OnCE command = 01001001); ODEC selects OPDBR as the destination for serial data.
2. Wait for acknowledge on $\overline{DS}$ line.
3. Send the 16-bit opcode: `MOVE reg, x:OGDB`
   After all 16 bits have been received, the PDB drives the OPDBR. ODEC generates PRNEW and releases the chip from the halt state, and the contents of the register specified in the instruction is loaded in the OPGDBR. The PRCYC1 signal (an internal signal) that marks the end of the instruction brings the chip again to the halt state, and an acknowledge is issued to the command controller.
4. Wait for acknowledge on $\overline{DS}$ line.
5. Send the command `READ OPGDBR` (OnCE command = 11001000); ODEC selects PGDB as the source for serial data, and an acknowledge is issued to the command controller.
6. Wait for acknowledge on $\overline{DS}$ line.
7. Issue 16 clocks to read out data from selected register.

12.11.3 Displaying X Memory Area Starting at Address xxxx

Example 12-6 shows the procedure for displaying X memory area; this procedure uses Rn to minimize serial traffic.

Example 12-6. Display X Memory Area from Address xxxx—OnCE

1. Send the command `WRITE OPDBR with GO and no EX` (OnCE command = 01001001); ODEC selects OPDBR as the destination for serial data.
2. Wait for acknowledge on $\overline{DS}$ line.
3. Send the 16-bit opcode: `MOVE R0,x:OGDB`
   After all 16 bits have been received, the PDB drives the OPDBR. ODEC generates PRNEW and releases the chip from the halt state, and the contents of R0 are loaded in the OPGDBR. The PRCYC1 signal that marks the end of the instruction brings the chip again to the halt state, and an acknowledge is issued to the command controller.
4. Wait for acknowledge on $\overline{DS}$ line.
5. Send the command `READ OPGDBR` (OnCE command = 11001000). ODEC selects PGDB as the source for serial data, and an acknowledge is issued to the command controller.
6. Wait for acknowledge on $\overline{DS}$ line.
7. The command controller generates 16 clocks that shift out the contents of the OPGDBR. The value of R0 is thus saved and will be restored before exiting the debug mode.
8. Send the command `WRITE OPDBR with no GO and no EX` (OnCE command = 00001001); ODEC selects OPDBR as destination for the serial data.
9. Wait for acknowledge on $\overline{DS}$ line.

10. Send the 16 bits of opcode: `MOVE #$xxxx,R0`. After all 16 bits have been received, the PDB drives the OPDBR. ODEC generates PRNEW, so the PILR is loaded with the opcode. An acknowledge is issued to the command controller.
11. Wait for acknowledge on $\overline{DS}$ line.
12. Send the command `WRITE OPDBR with GO and no EX` (OnCE command = 01001001); ODEC selects OPDBR as the destination for serial data.
13. Wait for acknowledge on $\overline{DS}$ line.
14. Send the 16 bits of the second word of: `MOVE #$xxxx,R0` (the xxxx field) where xxxx is the address to be read.
    After all 16 bits have been received, the PDB drives the OPDBR. ODEC releases the chip from the halt state, and the instruction starts execution. The PRCYC1 signal that marks the end of the instruction brings the chip again to the halt state, and an acknowledge is issued to the command controller.
15. Wait for acknowledge on $\overline{DS}$ line.
16. Send the command `WRITE OPDBR with GO and no EX` (OnCE command = 01001001); ODEC selects OPDBR as the destination for serial data.
17. Wait for acknowledge on $\overline{DS}$ line.
18. Send the 16 bit opcode: `MOVE X:(R0)+,x:OGDB.` After all 16 bits have been received, the PDB drives the OPDBR. ODEC generates PRNEW and releases the chip form the halt state, and the contents of X:(R0) are loaded in the OPGDBR. The PRCYC1 signal that marks the end of the instruction brings the chip again to the halt state, and an acknowledge is issued to the command controller.
19. Wait for acknowledge on $\overline{DS}$ line.
20. Send the command `READ OPGDBR` (OnCE command = 11001000). ODEC selects PGDB as the source for serial data and an acknowledge is issued to the command controller.
21. Wait for acknowledge on $\overline{DS}$ line.
22. Issue 16 clocks to read out data from selected register.
23. Send the command `NO SELECTION with GO and no EX` (OnCE command 11000000); ODEC releases the chip from the halt state, and the instruction is executed once again (in a REPEAT-like fashion). The PRCYC1 signal that marks the end of the instruction brings the chip again to the halt state, and an acknowledge is issued to the command controller.
24. Wait for acknowledge on $\overline{DS}$ line.
25. Send the command `READ OPGDBR` (OnCE command = 11001000). ODEC selects GDB as the source for serial data and an acknowledge is issued to the command controller.
26. Wait for acknowledge on $\overline{DS}$ line.
27. Issue 16 clocks to read out data from selected register.
28. Repeat from step 23 onward until the entire memory area is examined. At the end of the process, R0 must be restored.

12.11.4 Returning from Debug Mode to Normal Mode

There are two cases for returning from the debug mode. In Example 12-7, control will be returned to the program that was running before debug was initiated, and in Example 12-8, the registers will be changed to jump to a different program. There is *no acknowledgment* on the DSO pin when the chip leaves the OnCE mode following a GO, EX. This is a special case of the “write a register” option.

Example 12-7. Returning from Debug Mode to Normal Mode—OnCE

1. Send the command `WRITE OPDBR with no GO and no EX` (OnCE command = 00001001). ODEC selects the OPDBR as the destination for serial data. Also, ODEC selects the on-chip PAB register as the source for the PAB bus. After the PAB was driven, an acknowledge is issued to the command controller.
2. Wait for acknowledge on $\overline{\text{DS}}$ line.
3. Send the 16 bits of the saved PGDB value. After all 16 bits have been received, the PDB drives the OPDBR. ODEC generates PRNEW, so the entire chip loads the opcode. An acknowledge is issued to the command controller.
4. Wait for acknowledge on $\overline{\text{DS}}$ line.
5. Send the command `WRITE OPDBR with GO and EX` (OnCE command = 01101001). ODEC selects OPDBR as the destination for serial data.
6. Wait for acknowledge on $\overline{\text{DS}}$ line.
7. Send the 16 bits of the saved OPDBR value. After all 16 bits have been received, the PDB drives the OPDBR. ODEC releases the chip form the halt state, and the debug mode bit in OSR is cleared. The chip continues to execute instructions until a debug mode condition occurs.

Example 12-8. Jump to a New Program—Go from Address $xxxx

1. Send the command `WRITE OPDBR with no GO and no EX` (OnCE command = 00001001). ODEC selects OPDBR as the destination for serial data.
2. Wait for acknowledge on $\overline{\text{DS}}$ line.
3. Send 16 bits of the opcode of a two-word jump instruction instead of the saved OPGDBR (instruction latch) value. After all the 16 bits have been received, the PDB drives the OPDBR. ODEC causes the DSP to load the opcode. An acknowledge is issued to the command controller.
4. Wait for acknowledge on $\overline{\text{DS}}$ line.
5. Send the command `WRITE OPDBR with GO and EX` (OnCE command = 01101001). ODEC selects OPDBR as the destination for serial data.
6. Wait for acknowledge on $\overline{\text{DS}}$ line.

Example 12-8. Jump to a New Program—Go from Address $xxxx (Continued)

7. Send 16 bits of the target absolute address ($xxxx). The chip will resume fetching from the target address (with no pipeline problems). The trace counter will count this instruction, so the current trace counter may need to be corrected if the trace mode enable bit in the OSCR has been set.
   In other words, after 16 bits have been received, the PDB drives the OPDBR. ODEC releases the chip from the halt state, and the debug mode bit in OSCR is cleared. The chip executes first the jump instruction and then fetches the instruction from the target address. The chip continues to execute instructions from that address until a debug mode condition occurs.

12.12 OnCE Unit Low-Power Operation

If the 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.

12.13 Resetting the Chip Without Resetting the OnCE Unit

The DSP can be reset without resetting the OnCE module. This allows setting breakpoints on an application's final target hardware, which may have a power-on reset circuit.

The OnCE module reset is disabled using the following technique. If an ENABLE_ONCE instruction is in the JTAG IR when a hardware or COP timer reset occurs, then the OnCE module unit is *not* reset. All OnCE module registers retain their current values, and all valid breakpoints remain enabled. If any other instruction is in the JTAG IR when a chip reset occurs, the OnCE module is reset. This capability allows for the following sequence, which is useful for setting breakpoints on final target hardware:

1. Reset the DSP 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 DSP chip and enter the debug mode.
4. Program the desired breakpoint(s) and leave the ENABLE_ONCE instruction in the JTAG IR.
5. Reset the DSP chip 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. The DSP chip then correctly triggers in the application code when the desired breakpoint condition is detected.
7. Poll the JTAG port to detect the occurrence of this breakpoint.

Another technique for loading breakpoints is accomplished as follows:

1. Reset the DSP chip by asserting both the $\overline{\text{RESET}}$ pin and the $\overline{\text{TRST}}$ pin.
2. Release the $\overline{\text{TRST}}$ pin.
3. Shift an ENABLE_ONCE instruction into the JTAG IR.
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. The DSP chip then correctly triggers in the application code when the desired breakpoint condition is detected.
7. Poll the JTAG port to detect the occurrence of this breakpoint.

Another useful sequence is to enter the debug mode directly from reset. This is accomplished as follows:

1. Reset the DSP chip by asserting both the $\overline{\text{RESET}}$ pin and the $\overline{\text{TRST}}$ pin.
2. Release the $\overline{\text{TRST}}$ pin.
3. Shift a DEBUG_REQUEST instruction into the JTAG IR.
4. Release the $\overline{\text{RESET}}$ pin.
5. Come out of reset and directly enter the debug mode.

The OnCE module reset can still be forced on DSP chip reset even if there is an ENABLE_ONCE instruction in the JTAG IR. This is accomplished by asserting the $\overline{\text{TRST}}$ pin in addition to the $\overline{\text{RESET}}$ pin, which guarantees reset of the OnCE module.

Chapter 13
JTAG Port

The JTAG port is a dedicated, user-accessible test access port (TAP) that is compatible with the *IEEE Standard Test Access Port and Boundary-Scan Architecture* (1149.1a-1993). 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 DSP56824 supports circuit-board test strategies based on this standard.

Five dedicated pins interface to the TAP, which contains 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 scanning 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:

- Performing boundary-scan operations to test circuit-board electrical continuity
- Bypassing the DSP for a given circuit-board test by replacing the boundary scan register (BSR) with a single bit register
- Sampling the DSP system pins during operation and transparently shift out the result in the BSR; pre-loading values to output pins prior to invoking the EXTEST instruction
- Disabling the output drive to pins during circuit-board testing
- Providing a means of accessing the OnCE module controller and circuits to control a target system
- Querying identification information (manufacturer, part number, and version) from a DSP chip
- Forcing test data onto the outputs of a DSP IC while replacing its BSR in the serial data path with a single bit register
- Enabling a weak pull-up current device on all input signals of a DSP IC (helping to ensure deterministic test results in the presence of a continuity fault during interconnect testing)

This section includes aspects of the JTAG implementation that are specific to the DSP56824. It is intended to be used with IEEE 1149.1a-1993. The discussion includes those items required by the standard to be defined and, in certain cases, provides additional information specific to the DSP56824. For internal details and applications of the standard, refer to IEEE 1149.1a-1993.

13.1 JTAG/OnCE Port Pinout

As described in IEEE 1149.1a-1993, the JTAG port requires a minimum of four pins to support TDI, TDO, TCK, and TMS signals. The DSP56824 also uses the optional $\overline{\text{TRST}}$ input signal and multiplexes it so that the same pin can support the debug event ($\overline{\text{DE}}$) output signal used by the OnCE module interface. The pin functions are described in Table 13-1.

Table 13-1. 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 module. It is sampled on the rising edge of TCK and has an on-chip pull-up resistor.
TDO	**Test Data Output**—This tri-statable output pin provides a serial output 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 pin proves 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 *DSP56824 Technical Data*. When accessing the OnCE module through the JTAG TAP, the maximum frequency for TCK is one-eighth the maximum frequency specified for the DSP56824 core (that is, 8.75 MHz for TCK if the maximum CLK input is 70 MHz). 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.
$\overline{\text{TRST}}$/$\overline{\text{DE}}$	**Test Reset/Debug Event**—This bidirectional pin, when configured as an input, provides a reset signal to the JTAG TAP controller. When configured as an output, it signals debug events detected on a trigger condition. The operational mode of the pin is configured by bit 14 of the OnCE control register (OCR). The $\overline{\text{TRST}}$/$\overline{\text{DE}}$ pin has an on-chip pull-up resistor.
Note: Implementation of the $\overline{\text{TRST}}$/$\overline{\text{DE}}$ pin is not fully compliant with IEEE 1149.1a-1993.	

13.2 JTAG Port Architecture

The TAP controller is a simple state machine used to sequence the JTAG port through its valid operations, including the following:

- Serially shifting in or out a JTAG port command
- Updating (and decoding) the JTAG port instruction register (IR)
- Serially inputting or outputting a data value
- Updating 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.

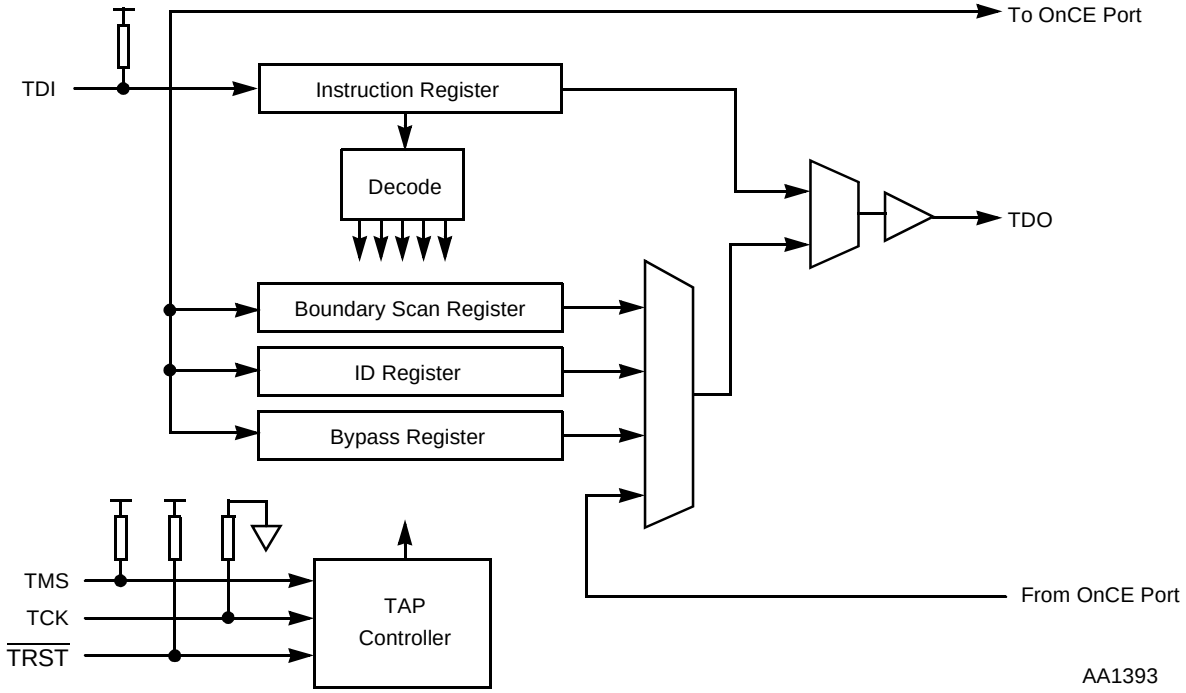


Figure 13-1. JTAG Block Diagram

A block diagram of the JTAG port is shown in Figure 13-1. The JTAG port has four read/write registers: the IR, BSR, device identification register, and bypass register. Access to the OnCE registers is described in **Chapter 12, "OnCE™ Module."**

The TAP controller provides access to the JTAG IR through the JTAG port. The other JTAG registers must be individually selected by the JTAG IR. Figure 13-2 shows the programming models for the JTAG registers on the DSP56824.

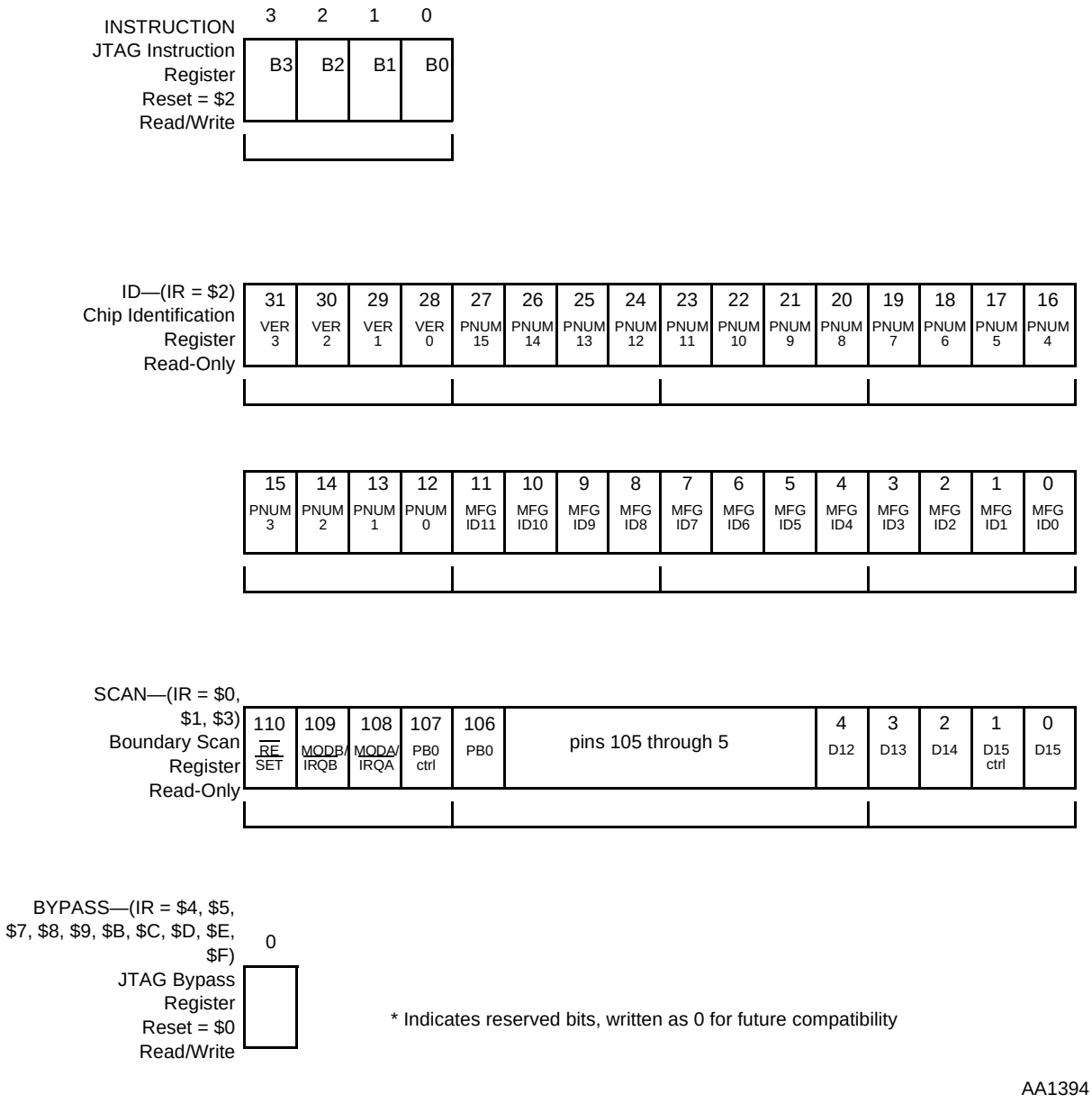


Figure 13-2. JTAG Port Programming Model

13.2.1 JTAG Instruction Register (IR) and Decoder

The TAP controller contains a 4-bit instruction register (IR). The instruction is presented to an instruction decoder during the Update-IR state. See Section 13.3, “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 DSP56824 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 4 bits B[3:0] of the IR decode the nine instructions, as shown in Table 13-2. All other encodings are reserved.

WARNING:

Reserved JTAG instruction encodings should not be used. Hazardous operation of the chip could occur if these instructions are used.

Table 13-2. JTAG IR 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 JTAG IR 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 1 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 shifted into the instruction shift register stage on Shift-IR state.

13.2.1.1 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. The EXTEST instruction asserts internal system reset for the DSP system logic for the duration of EXTEST in order to force a predictable internal state while performing external boundary-scan operations.

13.2.1.2 SAMPLE/PRELOAD (B[3:0] = 0001)

The SAMPLE/PRELOAD instruction enables the BSR between TDI and TDO. When this instruction is selected, the operation of the test logic has no effect on the operation of the on-chip system logic or on the flow of a signal between the system pin and the on-chip system logic, as specified by IEEE 1149.1a-1993. 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 to ensure proper system operation. Consequently, the same is true for the SAMPLE/PRELOAD functionality. The data latched into the BSR during the Capture-DR controller state may not match the drive state of the package signal if the system-required 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 the selection of the CLAMP, EXTEST, or EXTEST_PULLUP instruction. This initialization ensures that 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), the user must provide some form of external synchronization to achieve meaningful results when sampling system values using the SAMPLE/PRELOAD instruction.

13.2.1.3 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 one into its LSB. Since the bypass register loads a logic zero 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.

13.2.1.4 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 one 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: traversing into the run-test/idle state for extra TCK periods of charging delay, or limiting the maximum TCK frequency (slow down the TCK) so that two TCK periods are adequate. The EXTEST_PULLUP instruction asserts internal system reset for the DSP system logic for the duration of EXTEST_PULLUP in order to force a predictable internal state while performing external boundary-scan operations.

13.2.1.5 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 DSP system logic for the duration of HIGHZ in order to force a predictable internal state while performing external boundary-scan operations.

13.2.1.6 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 can be facilitated by setting up guarding signal conditions that control the operation of logic not involved in the test 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 that while the signals continue to supply the guarding inputs to the in-circuit test site, the bypass mode is enabled, thus 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 DSP system logic for the duration of CLAMP in order to force a predictable internal state while performing external boundary-scan operations.

13.2.1.7 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 Motorola 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* for more information.

13.2.1.8 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 Motorola 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* for more information.

13.2.1.9 BYPASS (B[3:0] = 1111)

The BYPASS instruction enables the single-bit bypass register between TDI and TDO, as shown in Figure 13-3. 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 DSP 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 1149.1a-1993.

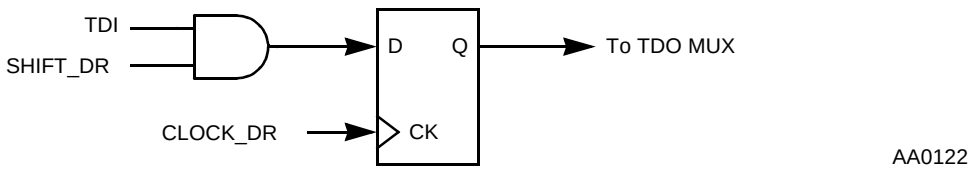


Figure 13-3. Bypass Register

13.2.2 JTAG Chip Identification (CID) Register

The chip identification (CID) register is a 32-bit register that provides a unique JTAG ID for the DSP56824. It is provided as a public instruction to allow the manufacturer, part number, and version of a component to be determined through the TAP. Figure 13-4 shows the CID register configuration.

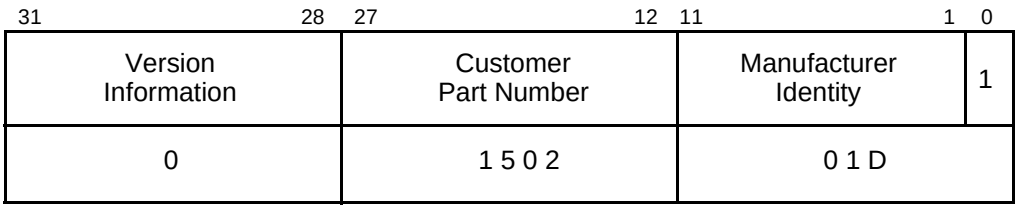


Figure 13-4. Chip Identification Register Configuration

For the initial release of the DSP56824, the device ID number is $0150201D. The version code is $0. Future revisions increment this version code. The value of the customer part number code is $1502.

Motorola's Manufacturer Identity is 00000001110. The customer part number consists of two parts: Motorola Design Center number (bits 27:22) and Design Center Assigned Sequence number (bits 21:12). DSP's Design Center number is 000101. 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 ID code is given in Table 13-3.

Table 13-3. Device ID Register Bit Assignment

Bit No.	Code Use	Value for DSP56824
31–28	Version number	0000 (for initial version only—these bits may vary)
27–22	Motorola Design Center ID	00 0101
21–12	Family and part ID	01 0000 0010
11–0	Motorola Manufacturer ID	0000 0001 1101

13.2.3 JTAG Boundary Scan Register (BSR)

The boundary scan register (BSR) is used to examine or control the scannable pins on the DSP56824. The BSR for the DSP56824 contains 111 bits. Each scannable pin has at least one BSR bit associated with it. Table 13-4 gives the contents of the BSR for the DSP56824.

Table 13-4. BSR Contents for DSP56824

Bit Number	Pin/Bit Name	Pin Type	BSR Cell Type	Pin Number (TQFP Package)
0	D15	Input/output	BC_6	45
1	D15 control	Control	BC_2	N/A
2	D14	Input/output	BC_6	44
3	D13	Input/output	BC_6	41

Table 13-4. BSR Contents for DSP56824 (Continued)

Bit Number	Pin/Bit Name	Pin Type	BSR Cell Type	Pin Number (TQFP Package)
4	D12	Input/output	BC_6	40
5	D11	Input/output	BC_6	39
6	D10	Input/output	BC_6	38
7	D9	Input/output	BC_6	37
8	D8	Input/output	BC_6	36
9	D7	Input/output	BC_6	35
10	D6	Input/output	BC_6	34
11	D5	Input/output	BC_6	33
12	D4	Input/output	BC_6	30
13	D3	Input/output	BC_6	29
14	D2	Input/output	BC_6	28
15	D1	Input/output	BC_6	27
16	D0	Input/output	BC_6	26
17	A0	Output	BC_2	25
18	A1	Output	BC_2	24
19	A2	Output	BC_2	23
20	A3	Output	BC_2	22
21	A4	Output	BC_2	21
22	$\overline{DS}$	Output	BC_2	18
23	$\overline{DS}$ control	Control	BC_2	N/A
24	$\overline{PS}$	Output	BC_2	17
25	$\overline{PS}$ control	Control	BC_2	N/A
26	A5	Output	BC_2	14
27	A6	Output	BC_2	13
28	A7	Output	BC_2	12
29	A8	Output	BC_2	11
30	A9	Output	BC_2	8
31	A10	Output	BC_2	7
32	A11	Output	BC_2	6

Table 13-4. BSR Contents for DSP56824 (Continued)

Bit Number	Pin/Bit Name	Pin Type	BSR Cell Type	Pin Number (TQFP Package)
33	A12	Output	BC_2	5
34	A13	Output	BC_2	4
35	A14	Output	BC_2	3
36	A15	Output	BC_2	2
37	A15 control	Control	BC_2	N/A
38	$\overline{RD}$	Output	BC_2	1
39	$\overline{RD}$ control	Control	BC_2	N/A
40	$\overline{WR}$	Output	BC_2	100
41	$\overline{WR}$ control	Control	BC_2	N/A
42	PC15/TIO2	Input/output	BC_6	99
43	PC15/TIO2 control	Control	BC_2	N/A
44	PC14/TIO01	Input/output	BC_6	98
45	PC14/TIO01 control	Control	BC_2	N/A
46	PC13/SRFS	Input/output	BC_6	97
47	PC13/SRFS control	Control	BC_2	N/A
48	PC12/SRCK	Input/output	BC_6	96
49	PC12/SRCK control	Control	BC_2	N/A
50	PC11/STFS	Input/output	BC_6	95
51	PC11/STFS control	Control	BC_2	N/A
52	PC10/STCK	Input/output	BC_6	94
53	PC10/STCK control	Control	BC_2	N/A
54	PC9/SRD	Input/output	BC_6	93
55	PC9/SRD control	Control	BC_2	N/A
56	PC8/STD	Input/output	BC_6	92
57	PC8/STD control	Control	BC_2	N/A
58	PC7/$\overline{SS1}$	Input/output	BC_6	91
59	PC7/$\overline{SS1}$ control	Control	BC_2	N/A
60	PC6/SCK1	Input/output	BC_6	88
61	PC6/SCK1 control	Control	BC_2	N/A

Table 13-4. BSR Contents for DSP56824 (Continued)

Bit Number	Pin/Bit Name	Pin Type	BSR Cell Type	Pin Number (TQFP Package)
62	PC5/MOSI1	Input/output	BC_6	87
63	PC5/MOSI1 control	Control	BC_2	N/A
64	PC4/MISO1	Input/output	BC_6	86
65	PC4/MISO1 control	Control	BC_2	N/A
66	PC3/$\overline{SS0}$	Input/output	BC_6	85
67	PC3/$\overline{SS0}$ control	Control	BC_2	N/A
68	PC2/SCK0	Input/output	BC_6	84
69	PC2/SCK0 control	Control	BC_2	N/A
70	PC1/MOSI0	Input/output	BC_6	83
71	PC1/MOSI0 control	Control	BC_2	N/A
72	PC0/MISO0	Input/output	BC_6	82
73	PC0/MISO0 control	Control	BC_2	N/A
74	EXTAL	Input	BC_4	77
75	CLKO	Output	BC_2	74
76	PB15	Input/output	BC_6	73
77	PB15 control	Control	BC_2	N/A
78	PB14	Input/output	BC_6	72
79	PB14 control	Control	BC_2	N/A
80	PB13	Input/output	BC_6	71
81	PB13 control	Control	BC_2	N/A
82	PB12	Input/output	BC_6	70
83	PB12 control	Control	BC_2	N/A
84	PB11	Input/output	BC_6	67
85	PB11 control	Control	BC_2	N/A
86	PB10	Input/output	BC_6	66
87	PB10 control	Control	BC_2	N/A
88	PB9	Input/output	BC_6	65
89	PB9 control	Control	BC_2	N/A
90	PB8	Input/output	BC_6	64

Table 13-4. BSR Contents for DSP56824 (Continued)

Bit Number	Pin/Bit Name	Pin Type	BSR Cell Type	Pin Number (TQFP Package)
91	PB8 control	Control	BC_2	N/A
92	PB7	Input/output	BC_6	63
93	PB7 control	Control	BC_2	N/A
94	PB6	Input/output	BC_6	62
95	PB6 control	Control	BC_2	N/A
96	PB5	Input/output	BC_6	61
97	PB5 control	Control	BC_2	N/A
98	PB4	Input/output	BC_6	58
99	PB4 control	Control	BC_2	N/A
100	PB3	Input/output	BC_6	57
101	PB3 control	Control	BC_2	N/A
102	PB2	Input/output	BC_6	56
103	PB2 control	Control	BC_2	N/A
104	PB1	Input/output	BC_6	55
105	PB1 control	Control	BC_2	N/A
106	PB0	Input/output	BC_6	54
107	PB0 control	Control	BC_2	N/A
108	MODA/$\overline{\text{IRQA}}$	Input	BC_4	53
109	MODB/$\overline{\text{IRQB}}$	Input	BC_4	52
110	$\overline{\text{RESET}}$	Input	BC_2	51

13.2.4 JTAG Bypass Register

The JTAG bypass register is a 1-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 together in a daisy chain. Individual DSPs or other devices can be programmed with the BYPASS instruction so that 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.

13.3 TAP Controller

The TAP controller is a synchronous finite state machine that contains 16 states, as illustrated in Figure 13-5. 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.

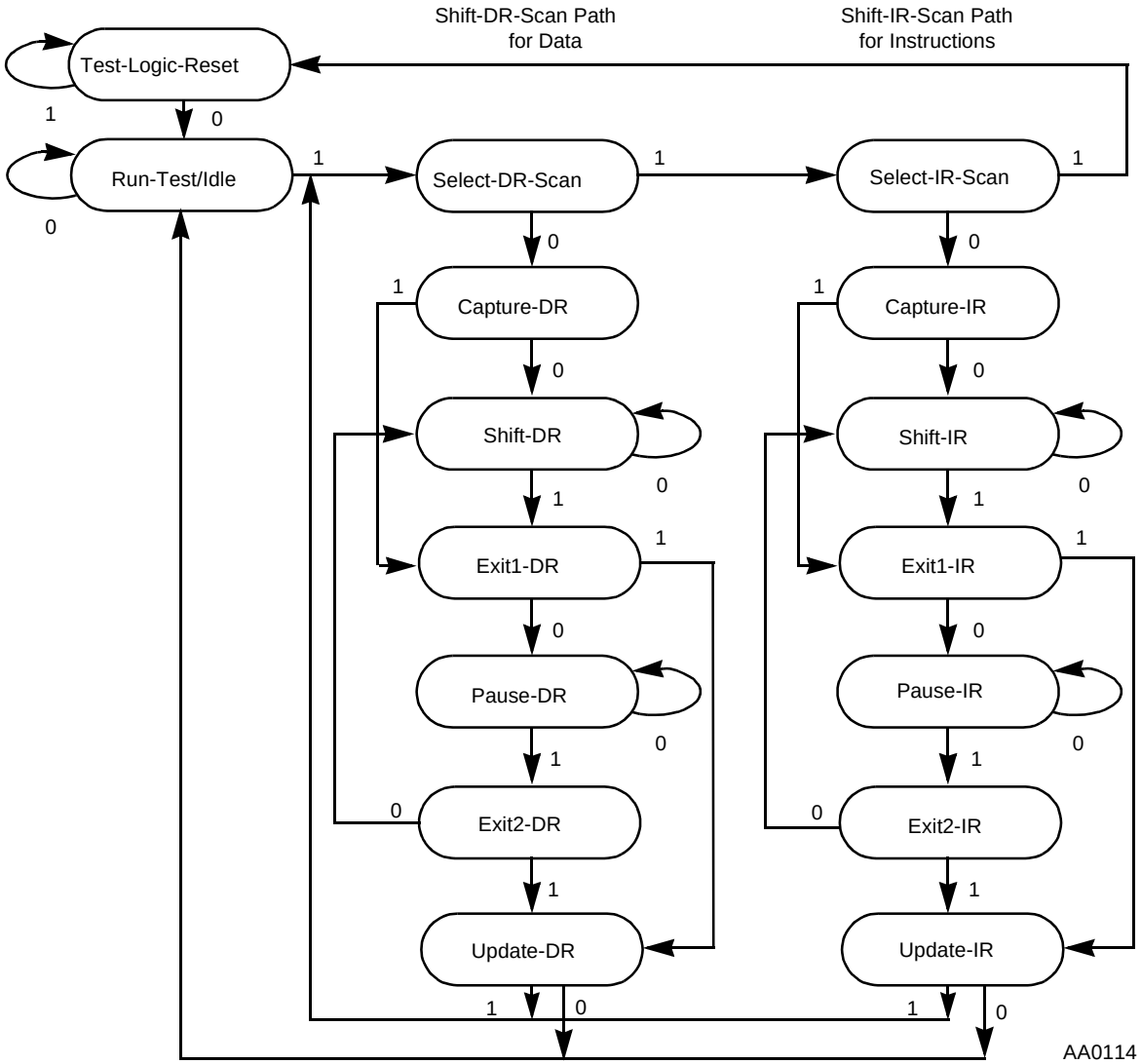


Figure 13-5. TAP Controller State Diagram

There are two paths through the 16-state machine. The Shift-IR-Scan path captures and loads JTAG instructions into the JTAG IR. 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 the JTAG port is used to access OnCE module registers, accesses are first enabled by shifting the ENABLE_ONCE instruction into the JTAG IR. 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.

13.4 DSP56824 Restrictions

The $\overline{\text{TRST}}$ pin shares functionality with the $\overline{\text{DE}}$ output function of the OnCE module. This implementation is not fully compliant with the JTAG standard. The function of this pin is controlled by the OnCE control register (OCR). The 2 bits in the OCR that determine how the pin functions, DE and DRM, are cleared on $\overline{\text{RESET}}$ assertion, forcing the $\overline{\text{TRST}}$/$\overline{\text{DE}}$ pin to its input JTAG reset function. The bits may be set either by an explicit write to the OCR, which involves a lengthy serial sequence, or on power up. To guarantee that $\overline{\text{TRST}}$ functions as a JTAG reset, it is recommended that both $\overline{\text{TRST}}$ and $\overline{\text{RESET}}$ be asserted at least once after powering up.

The control afforded by the output enable signals using the BSR and the EXTEST instruction requires a compatible circuit-board test environment to avoid device-destructive configurations. The user must avoid situations in which the DSP56824 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 powering up is concluded, TMS must be sampled as a logic one 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.

The DSP56824 features a low-power stop mode that is 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.

Since all DSP56824 clocks are disabled during stop state, the JTAG interface provides the means of polling the device status (sampled in the Capture-IR state).

Appendix A
Bootstrap Program

This section describes the bootstrap program contained in the DSP56824 program ROM. This program can load the on-chip program RAM from an external memory through the external memory interface (Port A) or from the serial peripheral interface 0 (SPI0), and then transfer program control to a starting address in program RAM.

The program RAM locations that are available for loading are the 128 internal program RAM locations from P:$0000 to P:$007F, or the 32K of external program RAM locations from P:$8000 to P:$FFFF. It is not possible to load any external program RAM locations from P:$0000 to P:$7FFF as these are only accessible in Mode 3. Figure 3-1 on page 3-2 provides a picture of the DSP56824 memory map.

The bootstrap program begins at P:$7F80, which is the Mode 1 reset vector location. The following sequence describes the bootstrap program flow.

1. External memory location P:$C000 is read to determine the source of the data to load into program RAM. If bit 15 (D[15]) of P:$C000 is zero, program RAM is loaded with values read through SPI0. If bit 15 (D[15]) is one, program RAM is loaded with values received through the lower byte of Port A.
2. Once the program determines the bootstrap source, data is read from that source. The first word (2 bytes) read contains the number of program words that are loaded. The second word (also 2 bytes) contains the starting address to which the program words are loaded; this address is also where program control is transferred to after exiting the bootstrap program. The least significant byte of each word should be transmitted first, followed by the most significant byte.
3. After the bootstrap program determines the number of words and the starting address, the remaining words are loaded byte-wise into contiguous program RAM. This continues until the specified number of words have been loaded.
4. When the loading of the program RAM has completed, the bootstrap program terminates and transfers program control to the starting address. At this point, the program the user has downloaded begins to execute.

NOTE:

This routine loads data starting with the least significant byte.

A.1 Design Considerations

Section A.1.1, “Boot Source Selection,” through Section A.1.3, “No Load Option,” describe hardware and software considerations involving choice of boot source and bootstrap operation when the COP (Computer Operating Properly) is enabled. In addition, a special No Load option is described.

A.1.1 Boot Source Selection

The DSP56824 bootstrap program provides a simple and flexible manner of selecting the boot source, which can be either SPI0 or Port A.

A.1.1.1 Bootstrapping from SPI0

To select SPI0 as the boot source, bit 15 (D[15]) of P:$C000 should be sampled as zero during the bootstrap program execution. If no external memory is mapped at location P:$C000, a pull-down resistor connected to pin D15 provides the correct signal.

In some cases, data can be loaded through SPI0 to external SRAM mapped to the top 32K of external program space, including location P:$C000. In this case, a pull-down resistor will not work since it will be overridden by the value read from location P:$C000 in SRAM. To solve this problem, the bootstrap program writes zero to this location before reading it, so no additional external logic is required.

A.1.1.2 Bootstrapping from Port A

To select Port A as the boot source, bit 15 (D[15]) of P:$C000 should be sampled as one during the bootstrap program execution. If no external memory is mapped to the upper byte at location P:$C000, a pull-up resistor connected to pin D15 provides the correct signal. This should be done if byte-wide EPROM is used as the source from which data is downloaded.

A.1.2 COP Reset

Special attention must be taken if the COP timer is to be used in an application where the DSP56824 comes out of reset in Mode 1. The COP timer is typically used as a watchdog-like timer to guard processor activity, providing an automatic reset signal if a failure occurs. Once started, the COP timer must be reset by software on a regular basis so that it never reaches its time-out value.

If the COP timer reaches its time-out value, an internal reset is generated within the chip and the COP reset vector is fetched. On a COP reset, the original operating mode that was latched from the pins on hardware reset specifies the location of the COP reset vector. For example, if the chip initially comes out of hardware reset in Mode 1, and later the mode is switched to Mode 3, the COP reset vector will be the Mode 1 COP reset vector, P:$7F82, which was the original mode when exiting reset.

COP reset in Mode 1 is essentially identical to hardware reset, as the boot ROM code is initiated. In the case when bootstrapping originally occurred via SPI0, the system needs a way to detect that a COP reset has occurred so the program is resent to the DSP via SPI0.

A.1.3 No Load Option

It is also possible to use the bootstrap program to jump to a particular location in program memory to begin execution without loading any program RAM. This can be accomplished by insuring that the first 2 bytes read, which select the number of program words to be loaded, are both zero.

A.2 Bootstrap Listing

The bootstrap program listing is shown in Example A-1.

Example A-1. DSP56824 Bootstrap Code

```
; ***************************************************************************
; * boot_56824.asm
; *
; * Bootstrap source code for the Motorola DSP56824.
; * (C) Copyright 1999 Motorola Inc.
; *
; * This is the source code for the bootstrap program contained in the 56824.
; * This program is executed on hardware reset in Mode 1. The program can load
; * the internal program memory from one of two external sources. The
; * program reads P:$C000 to decide which external source to access.
; * The sources include:
; *
; *    - Bootstrapping from external byte-wide memory (lower byte of data bus)
; *    - Bootstrapping through the SPI serial port (SPI0)
; *
; * The decision for which port is selected is shown here:
; *       if (bit 15 of P:$C000 == 0)
; *          load program RAM through the SPI (SPI0)
; *       if (bit 15 of P:$C000 == 1)
; *          load program RAM from consecutive byte-wide P: memory locations
; *          (typically EPROM), starting at P:$C000 (lower byte of data bus)
; *
; * The number of words to load and the starting load address:
; *    The first 2 bytes read from external memory or the SPI
; *    indicate the number of program words which will
; *    be loaded. The next 2 bytes indicate the starting load
; *    address; the boot loader will also jump to this address
; *    after all of the words have been loaded. Usually the load
; *    address is 0 (internal PRAM), but the user has the option
; *    to load external program ram if desired. Also, note that both of
; *    these parameters should be passed least-significant-byte first:
; *
; *    <1st byte read>  lower byte of number of words to be loaded
; *    <2nd byte read>  upper byte of number of words to be loaded
; *    <3rd byte read>  lower byte of load address
; *    <4th byte read>  upper byte of load address

; * Loading PRAM from external byte-wide memory:
; *    The boot ROM code will load bytes from the external P memory
; *    space beginning at P:$C004 (bits 7-0). Each two bytes will be condensed
; *    into a 16-bit word and stored in contiguous internal PRAM memory
; *    locations. Note that routine loads data starting with the least
; *    significant byte:
```

Example A-1. DSP56824 Bootstrap Code (Continued)

```
; *          P:$C000      lower byte of number of words to be loaded
; *          P:$C001      upper byte of number of words to be loaded
; *          P:$C002      lower byte of load address
; *          P:$C003      upper byte of load address
; *          P:$C004      lower byte of first word to be loaded
; *          P:$C005      upper byte of first word to be loaded
; *          P:$C006      lower byte of second word to be loaded
; *          P:$C007      upper byte of second word to be loaded
; *             .                and so on.
; *
; * Loading PRAM from the SPI port
; *    The boot ROM code loads the internal pram from the SPI port.
; *    Each two bytes will be condensed into a 16-bit word and stored in
; *    contiguous PRAM memory locations. Note that when using the SPI,
; *    the routine loads data starting with the least significant byte
; *    first. For the SPI, the maximum frequency allowed on the serial
; *    clock pin is 10 MHz.
; *
; * NOTE:
; *    If this program is entered in software by jumping to p:$7f80,
; *    make certain that the M01 register has been set to $FFFF (linear
; *    addressing). In addition, make sure the BCR register is set to $xxxF
; *    to support slow external EPROMs for the bootstrapping operation.
; *    Also, make sure the MB:MA bits in the OMR register are set to Mode 1.
; *
; * Registers used by the program:
; *    R0  - pointer used to write 16-bit instructions to on-chip PRAM
; *    R1  - points to load/jump address
; *    R2  - pointer used to read 8-bit data from off-chip P memory when
; *          bootstrapping through the External Bus
; *    Y0  - contents of P:$C000, used to save MSB which selects SPI vs. EXT BUS
; *    A   - loop counter
; *    B   - register which holds an 8-bit sample from one of the 2 ports
; *    SP -  used for the jump to the load address. SP=$3F after exiting
; *          the boot loader.
; *
; * Memory used by the program:
; *    X:$40 and X:$41 are used by the stack
; *
; ***************************************************************************
; *
             page    255
            opt     cc,now,mu

BOOT                EQU    $C000  ; The location in P: memory where the external
                                  ; byte-wide EPROM is located when loading from
                                  ; Port A. Also, the MSB of the value read at
                                  ; this location is used to select SPI vs. EXT BUS
PCC                 EQU     $FFED             ; Port C control register
SPCR0               EQU     $FFE2             ; SPI0 control register
SPSR0               EQU     $FFE1             ; SPI0 status register
SPSR0               EQU     $FFE1             ; SPI0 status register
```

Example A-1. DSP56824 Bootstrap Code (Continued)

```
; ***
; *** Initial entry into the bootstrap program:
; ***    - Initialize R2 to address of external byte-wide ROM ($C000)
; ***    - read the contents of P:$C000 (external memory)
; ***    - if bit 15 is one, bypass SPI initialization
; ***
            ORG    P:$7F80                  ; Mode 1 reset vector

            nop                         ; Two NOPs to account for the case when
            nop                         ; COP reset occurs and transfers control
                                        ; to the COP reset vector at P:$7F82
            move   #$3f,sp                  ; Initialize the stack pointer
                                            ;   above page 0
            MOVE   #BOOT,R2                 ; Setup R2 as P: address of
                                            ;   external byte-wide ROM
            clr    a
            movem  a,p:(r2)+                ; Handle case where there's
            lea    (r2)-                    ;   SRAM at P:$C000
            MOVE   P:(R2)+,Y0               ; Get P:$C000
            LEA    (R2)-                    ; Adjust pointer, because in the case
                                            ; where bootstrapping from external
                                            ; P memory, P:$C000 will be read again
            BRSET  #$8000,Y0,_BYPASS_SPI    ; if (P:$C000's MSB == 1)
                                            ;    bypass SPI
                                            ; else
                                            ;    initialize SPI
; ****************** Initialize the SPI if necessary *************
_INIT_SPI
            BFSET  #$0004,X:SPCR0           ; Set up SPIO's SPCR control register:
                                            ;    - interrupts disabled
                                            ;    - Wired-OR mode disabled
                                            ;    - Slave mode
                                            ;    - CPL set to 0
                                            ;    - CPH set to 1
                                            ;    - SPR2-0 not used since Slave mode
            BFSET  #$0040,X:SPCR0           ; Enable SPI via setting SPE bit
            BFSET  #$000F,X:PCC             ; Set PC0-PC3 to MISO0,MOSI0,SCK0,SS0

; ****************** End of SPI initialization *************

_BYPASS_SPI
; *** Determine the number of words to be loaded*************
             jsr   get_word
            ; received word is now in b0
            move   b0,a                     ; put number of words in a
; ********************************************************
; ******************** Determine the load address*************
            jsr    get_word
            ; received word is now in b0
            move   b0,r0                    ; put in r0
            move   b0,r1                    ; and save in r1 for jmp
; ********************************************************
            tstw   a
            beq    _exit                    ; if 0 words are to be loaded
                                            ;  branch to _exit
```

Example A-1. DSP56824 Bootstrap Code (Continued)

```
; ***************** Main Loop *************************
_main_loop
             jsr    get_word
             move   b0,b
             move   b,p:(r0)+                ; put word into PRAM
             decw   a                        ; decrement number of words counter
             bgt    _main_loop
; ******************** End of Main Loop ********************
; ***
; *** Exit bootstrap program
; ***         - goto P:(r1)
; ***
_exit
             ; goto to r1 by pushing it onto the stack and issuing an rts
             lea    (sp)+
             move   r1,x:(sp)+
             ANDC   #$FF00,SR                ; clear SR as if HW reset
             move   sr,x:(sp)
             rts

; ******* Shared subroutine to pack 2 bytes read into a word *******
get_word
             do     #2,end_get_word
             brset  #$8000,y0,_read_from_ext_mem
_read_from_spi
             BRCLR  #$0080,X:SPSR0,*         ; - Wait for SPIF flag to go high
             MOVE   X:SPSR0,B1               ; Read of SPSR0 used to clear SPIF flag
             MOVE   X:SPDR0,B1               ; Put SPI's byte into lower byte of B1
             bra    _pack_pram
_read_from_ext_mem
             move   p:(r2)+,b1               ; put external mem into low bype of B1
_pack_pram
             rep    #8
             asr    B                        ; shift into B0
             nop
end_get_word
             rts
; ***************** End of subroutine *********************************
             END
```

Freescale Semiconductor, Inc.
ARCHIVED BY FREESCALE SEMICONDUCTOR, INC. 2005

Appendix B
Boundary Scan Description Language Listing

This section provides the boundary scan description language (BSDL) listing for the DSP56824.

B.1 DSP56824 BSDL Listing—100-Pin TQFP

Example B-1 provides the BSDL listing for the DSP56824 in the 100-pin thin quad flat pack (TQFP) package.

Example B-1. DSP56824 BSDL Listing—100-Pin TQFP

```
-- M O T O R O L A   S S D T   J T A G   S O F T W A R E
-- BSDL File Generated: Thu Sep 26 10:44:42 1996
--
-- Revision History:
--

entity FALCON is
       generic (PHYSICAL_PIN_MAP : string := "BU");

       port (TRSTZ:in      bit;
             TCK:          in     bit;
             TMS:          in     bit;
             TDI:          in     bit;
             TDO:          out    bit;
             D:            inout  bit_vector(0 to 15);
             A:            out    bit_vector(0 to 15);
             DSZ:          out    bit;
             PSZ:          out    bit;
             RDZ:          out    bit;
             WRZ:          out    bit;
             PC15_TIO2:    inout  bit;
             PC14_TIO01:   inout  bit;
             PC13_SRFS:    inout    bit;
             PC12_SRCK:    inout    bit;
             PC11_STFS:    inout    bit;
             PC10_STCK:    inout    bit;
             PC9_SRD:      inout    bit;
             PC8_STD:      inout    bit;
             PC7_SSZ1:     inout  bit;
             PC6_SCK1:     inout  bit;
             PC5_MOSI1:    inout  bit;
             PC4_MISO1:    inout  bit;
             PC3_SSZ0:     inout  bit;
             PC2_SCK0:     inout  bit;
```

Example B-1. DSP56824 BSDL Listing—100-Pin TQFP (Continued)

```
                PC1_MOSI0:    inout  bit;
                PC0_MISO0:    inout  bit;
                EXTAL:        in     bit;
                CLKO:         buffer bit;
                PB:           inout  bit_vector(0 to 15);
                MODA_IRQAZ:   in     bit;
                MODB_IRQBZ:   in     bit;
                RESETZ:       in     bit;
                XTAL:         linkage bit;
                SXFC:         linkage bit;
                VSSD0:        linkage bit;
                VSSD1:        linkage bit;
                VDDD0:        linkage bit;
                VDDD1:        linkage bit;
                VSSA0:        linkage bit;
                VSSA1:        linkage bit;
                VDDA0:        linkage bit;
                VDDA1:        linkage bit;
                VSSQ0:        linkage bit;
                VSSQ1:        linkage bit;
                VDDQ0:        linkage bit;
                VDDQ1:        linkage bit;
                VSSPLL:       linkage bit;
                VDDPLL:       linkage bit;
                VSSCLK:       linkage bit;
                VDDCLK:       linkage bit;
                VSSPC:        linkage bit;
                VDDPC:        linkage bit;
                VSSPB:        linkage bit;
                VDDPB:        linkage bit);

        use STD_1149_1_1994.all;

        attribute PIN_MAP of FALCON : entity is PHYSICAL_PIN_MAP;

        constant BU : PIN_MAP_STRING :=
                "RDZ:         1, " &
                "A:           (25, 24, 23, 22, 21, 14, 13, 12, 11, 8, 7, 6, 5, 4, 3, 2),
 " &
                "VDDA0:        9, " &
                "VSSA0:       10, " &
                "VSSQ1:       15, " &
                "VDDQ1:       16, " &
                "PSZ:         17, " &
                "DSZ:         18, " &
                "VSSA1:       19, " &
                "VDDA1:       20, " &
                "D:(26, 27, 28, 29, 30, 33, 34, 35, 36, 37, 38, 39, 40, 41, 44, 45), " &
                "VDDD0:       31, " &
                "VSSD0:       32, " &
                "VSSD1:       42, " &
                "VDDD1:       43, " &
                "TDO:         46, " &
                "TMS:         47, " &
                "TCK:         48, " &
                "TRSTZ:       49, " &
                "TDI:         50, " &
                "RESETZ:      51, " &
                "MODB_IRQBZ:  52, " &
                "MODA_IRQAZ:  53, " &
                "PB:(54, 55, 56, 57, 58, 61, 62, 63, 64, 65, 66, 67, 70, 71, 72, 73)," &
                "VDDPB:       59, " &
```

Example B-1. DSP56824 BSDL Listing—100-Pin TQFP (Continued)

```
                "VSSPB:       60, " &
                "VSSQ0:       68, " &
                "VDDQ0:       69, " &
                "CLKO:        74, " &
                "VSSCLK:      75, " &
                "XTAL:        76, " &
                "EXTAL:       77, " &
                "VDDCLK:      78, " &
                "SXFC:        79, " &
                "VDDPLL:      80, " &
                "VSSPLL:      81, " &
                "PC0_MISO0:   82, " &
                "PC1_MOSI0:   83, " &
                "PC2_SCK0:    84, " &
                "PC3_SSZ0:    85, " &
                "PC4_MISO1:   86, " &
                "PC5_MOSI1:   87, " &
                "PC6_SCK1:    88, " &
                "VSSPC:       89, " &
                "VDDPC:       90, " &
                "PC7_SSZ1:    91, " &
                "PC8_STD:     92, " &
                "PC9_SRD:     93, " &
                "PC10_STCK:   94, " &
                "PC11_STFS:   95, " &
                "PC12_SRCK:   96, " &
                "PC13_SRFS:   97, " &
                "PC14_TIO01:  98, " &
                "PC15_TIO2:   99, " &
                "WRZ:         100 " ;

        attribute TAP_SCAN_IN     of     TDI : signal is true;
        attribute TAP_SCAN_OUT    of     TDO : signal is true;
        attribute TAP_SCAN_MODE   of     TMS : signal is true;
        attribute TAP_SCAN_RESET  of     TRSTZ : signal is true;
        attribute TAP_SCAN_CLOCK  of     TCK : signal is (20.0e6, BOTH);
        attribute INSTRUCTION_LENGTH of FALCON : entity is 4;
        attribute INSTRUCTION_OPCODE of FALCON : entity is
                "EXTEST             (0000)," &
                "SAMPLE             (0001)," &
                "IDCODE             (0010)," &
                "CLAMP              (0101)," &
                "HIGHZ              (0100)," &
                "EXTEST_PULLUP      (0011)," &
                "ENABLE_ONCE        (0110)," &
                "DEBUG_REQUEST      (0111)," &
                "PLLRES_DISABLE     (1000)," &
                "BYPASS             (1111)";

        attribute INSTRUCTION_CAPTURE of FALCON : entity is "XX01";
        attribute INSTRUCTION_PRIVATE of FALCON : entity is "ENABLE_ONCE,
DEBUG_REQUEST, PLLRES_DISABLE";

        attribute IDCODE_REGISTER   of FALCON : entity is
                "0010"              & -- version
                "000101"            & -- manufacturer's use
                "0100000000"        & -- sequence number
                "00000001110"       & -- manufacturer identity
                "1";                 -- 1149.1 requirement
```

Example B-1. DSP56824 BSDL Listing—100-Pin TQFP (Continued)

```
attribute REGISTER_ACCESS of FALCON : entity is
      "BOUNDARY   (EXTEST_PULLUP)," &
      "BYPASS   (ENABLE_ONCE,DEBUG_REQUEST,PLLRES_DISABLE)" ;

attribute BOUNDARY_LENGTH of FALCON : entity is 111;

attribute BOUNDARY_REGISTER of FALCON : entity is
-- num     cell   port    func    safe  [ccell dis rslt]
      "0      (BC_6, D(15),  bidir,    X,   1,    0,   Z)," &
      "1      (BC_2, *,      control,  0)," &
      "2      (BC_6, D(14),  bidir,    X,   1,    0,   Z)," &
      "3      (BC_6, D(13),  bidir,    X,   1,    0,   Z)," &
      "4      (BC_6, D(12),  bidir,    X,   1,    0,   Z)," &
      "5      (BC_6, D(11),  bidir,    X,   1,    0,   Z)," &
      "6      (BC_6, D(10),  bidir,    X,   1,    0,   Z)," &
      "7      (BC_6, D(9),   bidir,    X,   1,    0,   Z)," &
      "8      (BC_6, D(8),   bidir,    X,   1,    0,   Z)," &
      "9      (BC_6, D(7),   bidir,    X,   1,    0,   Z)," &
      "10     (BC_6, D(6),   bidir,    X,   1,    0,   Z)," &
      "11     (BC_6, D(5),   bidir,    X,   1,    0,   Z)," &
      "12     (BC_6, D(4),   bidir,    X,   1,    0,   Z)," &
      "13     (BC_6, D(3),   bidir,    X,   1,    0,   Z)," &
      "14     (BC_6, D(2),   bidir,    X,   1,    0,   Z)," &
      "15     (BC_6, D(1),   bidir,    X,   1,    0,   Z)," &
      "16     (BC_6, D(0),   bidir,    X,   1,    0,   Z)," &
      "17     (BC_2, A(0),   output3,  X,   37,   0,   Z)," &
      "18     (BC_2, A(1),   output3,  X,   37,   0,   Z)," &
      "19     (BC_2, A(2),   output3,  X,   37,   0,   Z)," &
      "20     (BC_2, A(3),   output3,  X,   37,   0,   Z)," &
      "21     (BC_2, A(4),   output3,  X,   37,   0,   Z)," &
      "22     (BC_2, DSZ,    output3,  X,   23,   0,   Z)," &
      "23     (BC_2, *,      control,  0)," &
      "24     (BC_2, PSZ,    output3,  X,   25,   0,   Z)," &
      "25     (BC_2, *,      control,  0)," &
      "26     (BC_2, A(5),   output3,  X,   37,  0,   Z)," &
      "27     (BC_2, A(6),   output3,  X,   37,  0,   Z)," &
      "28     (BC_2, A(7),   output3,  X,   37,  0,   Z)," &
      "29     (BC_2, A(8),   output3,  X,   37,  0,   Z)," &
      "30     (BC_2, A(9),   output3,  X,   37,  0,   Z)," &
      "31     (BC_2, A(10),  output3,  X,   37,  0,   Z)," &
      "32     (BC_2, A(11),  output3,  X,   37,  0,   Z)," &
      "33     (BC_2, A(12),  output3,  X,   37,  0,   Z)," &
      "34     (BC_2, A(13),  output3,  X,   37,  0,   Z)," &
      "35     (BC_2, A(14),  output3,  X,   37,  0,   Z)," &
      "36     (BC_2, A(15),  output3,  X,   37,  0,   Z)," &
      "37     (BC_2, *,      control,  0)," &
      "38     (BC_2, RDZ,    output3,  X,   39,  0,   Z)," &
      "39     (BC_2, *,      control,  0)," &
      "40     (BC_2, WRZ,    output3,  X,   41,  0,   Z)," &
      "41     (BC_2, *,      control,  0)," &
      "42     (BC_6, PC15_TIO2,bidir,  X,   43,  0,   Z)," &
      "43     (BC_2, *,      control,  0)," &
      "44     (BC_6, PC14_TIO01,bidir, X,   45,  0,   Z)," &
      "45     (BC_2, *,      control,  0)," &
      "46     (BC_6, PC13_SRFS, bidir, X,   47,  0,   Z)," &
      "47     (BC_2, *,      control,  0)," &
      "48     (BC_6, PC12_SRCK,bidir,  X,   49,  0,   Z)," &
      "49     (BC_2, *,      control,  0)," &
      "50     (BC_6, PC11_STFS, bidir, X,   51,  0,   Z)," &
      "51     (BC_2, *,      control,  0)," &
      "52     (BC_6, PC10_STCK, bidir, X,   53,  0,   Z)," &
```

Example B-1. DSP56824 BSDL Listing—100-Pin TQFP (Continued)

```
"53     (BC_2, *,     control,   0)," &
"54     (BC_6, PC9_SRD, bidir,   X,   55,  0,   Z)," &
"55     (BC_2, *,      control,  0)," &
"56     (BC_6, PC8_STD,bidir,    X,   57,  0,   Z)," &
"57     (BC_2, *,      control,  0)," &
"58     (BC_6, PC7_SSZ1, bidir,  X,   59,  0,   Z)," &
"59     (BC_2, *,      control,  0)," &
"60     (BC_6, PC6_SCK1, bidir,  X,   61,  0,   Z)," &
"61     (BC_2, *,      control,  0)," &
"62     (BC_6, PC5_MOSI1, bidir, X,   63,  0,   Z)," &
"63     (BC_2, *,      control,  0)," &
"64     (BC_6, PC4_MISO1, bidir, X,   65,  0,   Z)," &
"65     (BC_2, *,      control,  0)," &
"66     (BC_6, PC3_SSZ0, bidir,  X,   67,  0,   Z)," &
"67     (BC_2, *,      control,  0)," &
"68     (BC_6, PC2_SCK0, bidir,  X,   69,  0,   Z)," &
"69     (BC_2, *,      control,  0)," &
"70     (BC_6, PC1_MOSI0, bidir, X,   71,  0,   Z)," &
"71     (BC_2, *,      control,  0)," &
"72     (BC_6, PC0_MISO0, bidir, X,   73,  0,   Z)," &
"73     (BC_2, *,      control,  0)," &
"74     (BC_4, EXTAL,  input,    X)," &
"75     (BC_2, CLKO,   output2,  X)," &
"76     (BC_6, PB(15), bidir,    X,   77,  0,   Z)," &
"77     (BC_2, *,      control,  0)," &
"78     (BC_6, PB(14), bidir,    X,   79,  0,   Z)," &
"79     (BC_2, *,      control,  0)," &
"80     (BC_6, PB(13), bidir,    X,   81,  0,   Z)," &
"81     (BC_2, *,      control,  0)," &
"82     (BC_6, PB(12), bidir,    X,   83,  0,   Z)," &
"83     (BC_2, *,      control,  0)," &
"84     (BC_6, PB(11), bidir,    X,   85,  0,   Z)," &
"85     (BC_2, *,      control,  0)," &
"86     (BC_6, PB(10), bidir,    X,   87,  0,   Z)," &
"87     (BC_2, *,      control,  0)," &
"88     (BC_6, PB(9),  bidir,    X,   89,  0,   Z)," &
"89     (BC_2, *,      control,  0)," &
"90     (BC_6, PB(8),  bidir,    X,   91,  0,   Z)," &
"91     (BC_2, *,      control,  0)," &
"92     (BC_6, PB(7),  bidir,    X,   93,  0,   Z)," &
"93     (BC_2, *,      control,  0)," &
"94     (BC_6, PB(6),  bidir,    X,   95,  0,   Z)," &
"95     (BC_2, *,      control,  0)," &
"96     (BC_6, PB(5),  bidir,    X,   97,  0,   Z)," &
"97     (BC_2, *,      control,  0)," &
"98     (BC_6, PB(4),  bidir,    X,   99,  0,   Z)," &
"99     (BC_2, *,      control,  0)," &
"100    (BC_6, PB(3),  bidir,    X,   101, 0,   Z)," &
"101    (BC_2, *,      control,  0)," &
"102    (BC_6, PB(2),  bidir,    X,   103, 0,   Z)," &
"103    (BC_2, *,      control,  0)," &
"104    (BC_6, PB(1),  bidir,    X,   105, 0,   Z)," &
"105    (BC_2, *,      control,  0)," &
"106    (BC_6, PB(0),  bidir,    X,   107, 0,   Z)," &
"107    (BC_2, *,      control,  0)," &
"108    (BC_4, MODA_IRQAZ, input,X)," &
"109    (BC_4, MODB_IRQBZ, input,X)," &
"110    (BC_2, RESETZ, input,    X)";

end FALCON;
```

Freescale Semiconductor, Inc.
ARCHIVED BY FREESCALE SEMICONDUCTOR, INC. 2005

Appendix C
Programmer's Sheets

The following pages provide a set of reference tables and programming sheets that are intended to simplify programming the DSP56824. The programming sheets provide room to write in the value of each bit and the hexadecimal value for each register. The programmer can photocopy these sheets.

C.1 Instruction Set Summary

The following tables provide a brief summary of the instruction set for the DSP56824. Table C-1 summarizes the instruction set. Table C-2 on page C-5 and Table C-3 on page C-6 provide a key to the abbreviations in the summary table. For complete instruction set details, see Appendix A of the *DSP56800 Family Manual*.

Table C-1. 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	—	—	—	—	—	—	—	—

Table C-1. Instruction Set Summary (Continued)

					CCR							
Mnemonic	Syntax	Parallel Moves	Prog. Word	Clock Cycles	SZ	L	E	U	N	Z	V	C
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	—	—	—	—	—	—	—	?
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	—

Table C-1. Instruction Set Summary (Continued)

					CCR							
Mnemonic	**Syntax**	**Parallel Moves**	**Prog. Word**	**Clock Cycles**	**SZ**	**L**	**E**	**U**	**N**	**Z**	**V**	**C**
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	?
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	*	*	—	—	—	—	—	—

Table C-1. Instruction Set Summary (Continued)

					CCR							
Mnemonic	**Syntax**	**Parallel Moves**	**Prog. Word**	**Clock Cycles**	**SZ**	**L**	**E**	**U**	**N**	**Z**	**V**	**C**
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	—	*	—	—	—	—	—	—
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	—	?	?	?	?	?	?	?

Table C-1. Instruction Set Summary (Continued)

					CCR							
Mnemonic	**Syntax**	**Parallel Moves**	**Prog. Word**	**Clock Cycles**	**SZ**	**L**	**E**	**U**	**N**	**Z**	**V**	**C**
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	?
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 to the case in which 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 DSP 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.

Table C-2. Condition Code Register (CCR) Symbols (Standard Definitions)

Symbol	Description
SZ	Size bit indicating data growth detection
L	Limit bit indicating arithmetic overflow, data limiting, or both
E	Extension bit indicating if the integer portion is in use
U	Unnormalized bit indicating if the result is unnormalized

Table C-2. Condition Code Register (CCR) Symbols (Standard Definitions) (Continued)

Symbol	Description
N	Negative bit indicating if bit 35 (or 31) of the result is set
Z	Zero bit indicating if the result equals zero
V	Overflow bit indicating if arithmetic overflow has occurred in the result
C	Carry bit indicating if a carry or borrow occurred in the result

Table C-3. CCR Notation

Notation	Description
*	Set according to the standard definition by the result of the operation
—	Not affected by the operation
0	Cleared
1	Set
U	Undefined
?	Set according to the special computation definition by the result of the operation

C.2 Interrupt, Vector, and Address Tables

Table C-4. Interrupt Priority Structure

Priority	Exception	IPR Bits
Level 1 (Non-maskable)		
Highest	Hardware $\overline{\text{RESET}}$	—
	COP Timer RESET	—
	Illegal instruction trap	—
	Hardware stack overflow	—
	OnCE module instruction trap	—
Lower	SWI	—
Level 0 (Maskable)		
Higher	$\overline{\text{IRQA}}$ (external interrupt)	2, 1
	$\overline{\text{IRQB}}$ (External interrupt)	5, 4

Table C-4. Interrupt Priority Structure (Continued)

Priority	Exception	IPR Bits
	Channel 6 peripheral interrupt—SSI	9
	Channel 5 peripheral interrupt—Reserved	10
	Channel 4 peripheral interrupt—Timer module	11
	Channel 3 peripheral interrupt—SPI1	12
	Channel 2 peripheral interrupt—SPI0	13
	Channel 1 peripheral interrupt—Real-time timer	14
Lowest	Channel 0 peripheral interrupt—Port B GPIO	15

Table C-5. Reset and Interrupt Vectors

Interrupt Starting Address	Interrupt Priority Level	Interrupt Source
$0000	—	Hardware $\overline{\text{RESET}}$
$0002	—	COP timer RESET
$0004	—	(Reserved)
$0006	1	Illegal instruction trap
$0008	1	Software interrupt (SWI)
$000A	1	Hardware stack overflow
$000C	1	OnCE trap
$000E	1	(Reserved)
$0010	0	$\overline{\text{IRQA}}$
$0012	0	$\overline{\text{IRQB}}$
$0014	0	Port B GPIO interrupt
$0016	0	Real-time interrupt
$0018	0	Timer 0 overflow
$001A	0	Timer 1 overflow
$001C	0	Timer 2 overflow
$001E	0	(Reserved)
$0020	0	SSI receive data with exception status
$0022	0	SSI receive data

Table C-5. Reset and Interrupt Vectors (Continued)

Interrupt Starting Address	Interrupt Priority Level	Interrupt Source
$0024	0	SSI transmit data with exception status
$0026	0	SSI transmit data
$0028	0	SPI1 serial system
$002A	0	SPI0 serial system
$002C	0	(Reserved)
$002E	0	(Reserved)
$0030	0	(Reserved)
$0032	0	(Reserved)
$0034	0	(Reserved)
$0036	0	(Reserved)
$0038	0	(Reserved)
$003A	0	(Reserved)
$003C	0	(Reserved)
$003E	0	(Reserved)
$0040	0	(Reserved)
$0042	0	(Reserved)
$007C	0	(Reserved)
$007E	0	(Reserved)

Table C-6. DSP56824 I/O and On-Chip Peripheral Memory Map

Address	Register
X:$FFFF	OPGDBR—OnCE PGDB register
X:$FFFE	(Reserved)
X:$FFFD	(Reserved)
X:$FFFC	(Reserved)
X:$FFFB	IPR—Interrupt priority register
X:$FFFA	(Reserved)
X:$FFF9	BCR—Bus control register (Port A)

Table C-6. DSP56824 I/O and On-Chip Peripheral Memory Map (Continued)

Address	Register
X:$FFF8	(Reserved)
X:$FFF7	(Reserved)
X:$FFF6	(Reserved)
X:$FFF5	(Reserved)
X:$FFF4	(Reserved)
X:$FFF3	PCR1—PLL control register 1
X:$FFF2	PCR0—PLL control register 0
X:$FFF1	COPCTL—COP control register
X:$FFF0	COPCNT—COP/RTI count register (read-only) COPRST—COP reset register (write-only)
X:$FFEF	PCD—Port C data register
X:$FFEE	PCDDR—Port C data direction register
X:$FFED	PCC—Port C control register
X:$FFEC	PBD—Port B data register
X:$FFEB	PBDDR—Port B data direction register
X:$FFEA	PBINT—Port B interrupt register
X:$FFE9	(Reserved)
X:$FFE8	(Reserved)
X:$FFE7	(Reserved)
X:$FFE6	SPCR1—SPI1 control register
X:$FFE5	SPSR1—SPI1 status register
X:$FFE4	SPDR1—SPI1 data register
X:$FFE3	(Reserved)
X:$FFE2	SPCR0—SPI0 control register
X:$FFE1	SPSR0—SPI0 status register
X:$FFE0	SPDR0—SPI0 data register
X:$FFDF	TCR01—Timer 0 and 1 control register
X:$FFDE	TPR0—Timer 0 preload register
X:$FFDD	TCT0—Timer 0 count register
X:$FFDC	TPR1—Timer 1 preload register

Table C-6. DSP56824 I/O and On-Chip Peripheral Memory Map (Continued)

Address	Register
X:$FFDB	TCT1—Timer 1 count register
X:$FFDA	TCR2—Timer 2 control register
X:$FFD9	TPR2—Timer 2 preload register
X:$FFD8	TCT2—Timer 2 count register
X:$FFD7	(Reserved)
X:$FFD6	(Reserved)
X:$FFD5	STSR—SSI time slot register (write-only)
X:$FFD4	SCRRX—SSI receive control register
X:$FFD3	SCRTX—SSI transmit control register
X:$FFD2	SCR2—SSI control register 2
X:$FFD1	SCSR—SSI control/status register
X:$FFD0	SRX—SSI receive register (read-only) STX—SSI transmit register (write-only)
X:$FFCF	(Reserved)
X:$FFCE	(Reserved)
X:$FFCD	(Reserved)
X:$FFCC	(Reserved)
X:$FFCB	(Reserved)
X:$FFCA	(Reserved)
X:$FFC9	(Reserved)
X:$FFC8	(Reserved)
X:$FFC7	(Reserved)
X:$FFC6	(Reserved)
X:$FFC5	(Reserved)
X:$FFC4	(Reserved)
X:$FFC3	(Reserved)
X:$FFC2	(Reserved)
X:$FFC1	(Reserved)
X:$FFC0	(Reserved)

C.3 Programmer's Sheets

The following pages provide programmer's sheets that are intended to simplify programming the various registers in the DSP56824. The programmer's sheets provide room to write in the value of each bit and the hexadecimal value for each register. The programmer can photocopy these sheets.

The programmer's sheets are provided in the same order as the sections in this document. Table C-7 lists the sets of programmer's sheets, the registers described in the sheets, and the pages in this appendix where the sheets are located.

Table C-7. List of Programmer's Sheets

Type of Register	Register	Page
CPU	JTAG instruction register	C-13
	JTAG bypass register	C-13
	JTAG ID register	C-13
	JTAG boundary scan register	C-14
	SR—Status register (includes mode register and condition code register)	C-15
	OMR—Operating mode register	C-16
	IPR—Interrupt priority register	C-17
Memory	BCR—Bus control register	C-18
Port B GPIO	PBINT—Port B interrupt register	C-19
	PBD—Port B data register	C-19
	PBINT—Port B data direction register	C-19
Port C GPIO	PCC—Port C control register	C-20
	PCD—Port C data register	C-20
	PCDDR—Port C data direction register	C-20
SPI0	SPCR0—SPI 0 control register	C-21
	SPSR0—SPI 0 status register	C-22
	SPDR0—SPI 0 data register	C-22
SPI1	SPCR1—SPI 1 control register	C-23
	SPCS1—SPI 1 status register	C-24
	SPDR1—SPI 1 data register	C-24

Table C-7. List of Programmer's Sheets (Continued)

Type of Register	Register	Page
SSI	SCRRX—SSI receive control register	C-25
	SCRTX—SSI transmit control register	C-25
	SCR2—SSI control register 2	C-26
	SCSR—SSI control/status register	C-27
	STSR—SSI time slot register	C-28
	STX—SSI transmit register	C-28
	STX—SSI receive register	C-28
Timer 0	TCR01—Timer 0/1 control register	C-29
	TPR0—Timer 0 preload register	C-29
	TCT0—Timer 0 count register	C-29
Timer 1	TCR01—Timer 0/1 control register	C-30
	TPR1—Timer 1 preload register	C-30
	TCT1—Timer 1 count register	C-30
Timer 2	TCR2—Timer 2 control register	C-31
	TPR2—Timer 2 preload register	C-31
	TCT2—Timer 2 count register	C-31
PLL	PCR1—PLL control register 1	C-32
	PCR0—PLL control register 0	C-32
COP/RTI	COPCTL—COP control register	C-33
	COPCNT—COP count register	C-33

Application:_____ Date:_____

_____ Programmer:_____

CPU

JTAG Instruction Register
Reset = $2
Read/Write

3	2	1	0
B3	B2	B1	B0

JTAG Bypass Register
Reset = $0
Read/Write

0

JTAG ID Register
Read-Only

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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
				1				5				0			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PNUM 3	PNUM 2	PNUM 1	PNUM 0	MFG ID11	MFG ID10	MFG ID9	MFG ID8	MFG ID7	MFG ID6	MFG ID5	MFG ID4	MFG ID3	MFG ID2	MFG ID1	MFG ID0
2				0				1				D			

Bitfield	Usage
VER[3:0]	Version number of chip mask (varies with mask)
PNUM[15:0]	Customer part number (specifies DSP56824)
MFG ID[11:0]	Specifies manufacturer identity

Application: _____ Date: _____

_____ Programmer: _____

CPU

JTAG Boundary Scan Register Read-Only

110	109	108	107	106	105	104	103	102	101	100	99	98	97	96
$\overline{\text{RE}}$SET	MODB/IRQB	MODA/IRQA	PB0 ctrl	PB0	PB1 ctrl	PB1	PB2 ctrl	PB2	PB3 ctrl	PB3	PB4 ctrl	PB4	PB5 ctrl	PB5

95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80
PB6 ctrl	PB6	PB7 ctrl	PB7	PB8 ctrl	PB8	PB9 ctrl	PB9	PB10 ctrl	PB10	PB11 ctrl	PB11	PB12 ctrl	PB12	PB13 ctrl	PB13

79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64
PB14 ctrl	PB14	PB15 ctrl	PB15	CLKO	EX	PC0 MISO0 ctrl	PC0 MISO0	PC1/ MOSI0 ctrl	PC1/ MOSI0	PC2/ SCK0 ctrl	PC2/ SCK0	$\overline{\text{PC3}}$/ $\overline{\text{SS0}}$ ctrl	$\overline{\text{PC3}}$/ $\overline{\text{SS0}}$	PC4/ MISO1 ctrl	PC4/ MISO1

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
PC5/ MOSI1 ctrl	PC5/ MOSI1	PC6/ SCK1 ctrl	PC6/ SCK1	$\overline{\text{PC7}}$/ $\overline{\text{SS1}}$ ctrl	$\overline{\text{PC7}}$/ $\overline{\text{SS1}}$	PC8/ STD ctrl	PC8 STD	PC9/ SRD ctrl	PC9/ SRD	PC10/ STCK ctrl	PC10/ STCK	PC11/ STFS ctrl	PC11/ STFS	PC12/ SRCK ctrl	PC12/ SRCK

47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
PC13/ SRFS ctrl	PC13/ SRFS	PC14/ TIO01 ctrl	PC14/ TIO01	PC15/ TIO2 ctrl	PC15/ TIO2	$\overline{\text{WR}}$ ctrl	$\overline{\text{WR}}$	$\overline{\text{RD}}$ ctrl	$\overline{\text{RD}}$	A15 ctrl	A15	A14	A13	A12	A11

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
A10	A9	A8	A7	A6	A5	$\overline{\text{PS}}$ ctrl	$\overline{\text{PS}}$	$\overline{\text{DS}}$ ctrl	$\overline{\text{DS}}$	A4	A3	A2	A1	A0	D0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D1	D2	D3	D4	D5	D6	D7	D8	D9 ctrl	D10	D11	D12	D13	D14	D15 ctrl	D15

Application: _____ Date: _____

_____ Programmer: _____

Sheet 3 of 6

CPU

Carry
Overflow
Zero
Negative
Unnormalized
Extension
Limit
Size

I1	I0	Exceptions Permitted	Exceptions Masked
0	0	(Reserved)	(Reserved)
0	1	IPL0, IPL1	None
1	0	(Reserved)	(Reserved)
1	1	IPL1	IPL0

LF	Exceptions Masked
0	Loop not running
1	Loop is running

Status Register (SR)
Reset = $0300
Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LF	*	*	*	*	*	I1	I0	S	L	E	U	N	Z	V	C
	0	0	0	0	0										

MR
Mode Register

CCR
Condition Code Register

* = Reserved, program as zero

Application: _____ Date: _____

_____ Programmer: _____

Sheet 4 of 6

CPU

MB	MA	Description
0	0	Mode 0—Single Chip
0	1	(Reserved)
1	0	Mode 2—Normal Expanded
1	1	Mode 3—Development

EX	Description
0	External X memory disabled
1	External X memory enabled

SA	Description
0	Saturation mode disabled
1	Saturation mode enabled

NL	Description
0	Nested loop not executing
1	Nested loop is executing

R	Description
0	Convergent rounding used
1	Two's-complement rounding used

SD	Description
0	Stop delay enabled
1	Stop delay disabled

CC	Description
0	Condition codes generated based on 36-bit MAC result
1	Condition codes generated based on 32-bit MAC result

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Operating Mode Register (OMR)	NL	*	*	*	*	*	*	CC	*	SD	R	SA	EX	*	MB	MA
Reset = $0000		0	0	0	0	0	0		0					0		
Read/Write																

* = Reserved, program as zero

Application: _____ Date: _____

_____ Programmer: _____

Sheet 5 of 6

CPU

IBL1	IBL0	IRQ B Mode
0	0	IRQ B disabled
0	1	IRQ B enabled, level-sensitive trigger
1	0	IRQ B disabled
1	1	IRQ B enabled, edge-sensitive trigger

IAL1	IAL0	IRQ A Mode
0	0	IRQ A disabled
0	1	IRQ A enabled, level-sensitive trigger
1	0	IRQ A disabled
1	1	IRQ A enabled, edge-sensitive trigger

IxL1	IxINV	Trigger Mode
0	0	Low-level sensitive
0	1	High-level sensitive
1	0	Falling-edge sensitive
1	1	Rising-edge sensitive

Channel 6 IPL
Channel 5 IPL
Channel 4 IPL
Channel 3 IPL
Channel 2 IPL
Channel 1 IPL
Channel 0 IPL

Set to 1 to Enable Interrupt Channels

Interrupt Priority Register (IPR)
X:$FFFB
Reset = $0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CH0	CH1	CH2	CH3	CH4	CH5	CH6	*	*	*	IBL1	IBL0	IB INV	IAL1	IAL0	IA INV
							0	0	0						

* = Reserved, Program as 0

Interrupt Channel	Associated Peripheral
6	SSI
5	(Reserved)
4	Timer module
3	SPI1
2	SPI0
1	Real-time timer
0	Port B GPIO

Application: _____ Date: _____

_____ Programmer: _____

Sheet 6 of 6

CPU

Wait state field for external X (data) memory

Wait state field for external P (program) memory

DRV	Description
0	Port A pins are tri-stated
1	Port A pins remain driven

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Bus Control Register (BCR) X:$FFF9 Reset = $00FF	*	*	*	*	*	*	DRV	*	WSX3	WSX2	WSX1	WSX0	WSP3	WSP2	WSP1	WSP0
	0	**0**	**0**	**0**	**0**	**0**		**0**								
	$0															

* = Reserved, program as zero

Application: _____ Date: _____

_____ Programmer: _____

Sheet 1 of 1

Port B GPIO

INVn	Description
0	Detects rising edge
1	Detects falling edge

MSKn	Description
0	Pin masked from generating interrupt
1	Pin enabled for generating interrupt

Port B Interrupt Register (PBINT)
X:$FFEA
Reset = $0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSK7	MSK6	MSK5	MSK4	MSK3	MSK2	MSK1	MSK0	INV7	INV6	INV5	INV4	INV3	INV2	INV1	INV0

Port B Data Register (PBD)
X:$FFEC
Reset = Uninitialized

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BD15	BD14	BD13	BD12	BD11	BD10	BD9	BD8	BD7	BD6	BD5	BD4	BD3	BD2	BD1	BD0

Port B Data Direction Register (PBDDR)
X:$FFEB
Reset = $0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BDD15	BDD14	BDD13	BDD12	BDD11	BDD10	BDD9	BDD8	BDD7	BDD6	BDD5	BDD4	BDD3	BDD2	BDD1	BDD0

Application: _____ Date: _____

_____ Programmer: _____

Sheet 1 of 1

Port C GPIO

PCC register must be configured for Port C GPIO, Timers 0–2, SPI 0–1, and SSI. Use this programming sheet for all these peripherals.

Port C Control Register (PCC)
X:$FFED
Reset = $0000

Timer	Synchronous Serial Interface (SSI)			SPI1				SPI0							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CC15	CC14	CC13	CC12	CC11	CC10	CC9	CC8	CC7	CC6	CC5	CC4	CC3	CC2	CC1	CC0

CCn	Description
0	Pin is GPIO pin
1	Pin used for dedicated peripheral

Port C Data Register (PCD)
X:$FFEF
Reset = Uninitialized

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CD15	CD14	CD13	CD12	CD11	CD10	CD9	CD8	CD7	CD6	CD5	CD4	CD3	CD2	CD1	CD0

Port C Data Direction Register (PCDDR)
X:$FFEE
Reset = $0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CDD15	CDD14	CDD13	CDD12	CDD11	CDD10	CDD9	CDD8	CDD7	CDD6	CDD5	CDD4	CDD3	CDD2	CDD1	CDD0

Application: _____ Date: _____

_____ Programmer: _____

SPI0

CPH
Controls clock phase—see Section 72.1.8.

CPL	Description
0	SCK pin idles at logic low
1	SCK pin idles at logic high

MST	Description
0	SPI in Slave mode
1	SPI in Master mode

WOM	Description
0	SPI pins configured as push-pull drivers
1	SPI pins configured as open-drain drivers

SPE	Description
0	SPI disabled
1	SPI enabled

SPTE	Description
0	SPI interrupt disabled
1	SPI interrupt enabled

SPR2	SPR1	SPR0	Divider
0	0	0	1
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPI 0 Control Register (SPCR0) X:$FFE2 Reset = $0000	*	*	*	*	*	*	*	SPR2	SPIE	SPE	WOM	MST	CPL	CPH	SPR1	SPR0
	0	0	0	0	0	0	0									
	$0															

* = Reserved, program as zero

Application: _____ Date: _____

_____ Programmer: _____

Sheet 2 of 2

SPI0

WCOL	Description
0	(Bit is cleared)
1	Write collision detected

MDF	Description
0	(Bit is cleared)
1	Mode fault detected

SPIF	Description
0	(Bit is cleared)
1	Data transfer is completed

SPI 0 Status Register (SPSR0) X:$FFE1 Reset = $0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	SPIF	WCOL	*	MDF	*	*	*	*
0	0	0	0	0	0	0	0			0		0	0	0	0
$0				$0											

* = Reserved, Program as 0

SPI 0 Data Register (SPDR0) X:$FFE0 Reset = Uninitialized

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	data	data	data	data	data	data	data	data
0	0	0	0	0	0	0	0								
$0				$0											

* = Reserved, program as zero

Application: _____ Date: _____

Programmer: _____

SPI1

CPH
Controls clock phase—see Section 7.2.1.8

CPL	Description
0	SCK pin idles at logic low
1	SCK pin idles at logic high

MST	Description
0	SPI in Slave mode
1	SPI in Master mode

WOM	Description
0	SPI pins configured as push-pull drivers
1	SPI pins configured as open-drain drivers

SPE	Description
0	SPI disabled
1	SPI enabled

SPTE	Description
0	SPI interrupt disabled
1	SPI interrupt enabled

SPR2	SPR1	SPR0	Divider
0	0	0	1
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

SPI 1 Control Register (SPCR1)
X:$FFE6
Reset = $0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	SPR2	SPIE	SPE	WOM	MST	CPL	CPH	SPR1	SPR0
0	0	0	0	0	0	0									
$0															

* = Reserved, program as zero

Application: _____ Date: _____

_____ Programmer: _____

SPI1

WCOL	Description
0	(Bit is cleared)
1	Write collision detected

MDF	Description
0	(Bit is cleared)
1	Mode fault detected

SPIF	Description
0	(Bit is cleared)
1	Data transfer is completed

SPI 1 Status Register (SPSR1)
X:$FFE5
Reset = $0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	SPIF	WCOL	*	MDF	*	*	*	*
0	0	0	0	0	0	0	0			0		0	0	0	0
$0				$0											

* = Reserved, program as zero

SPI 1 Data Register (SPDR1)
X:$FFE4
Reset = Uninitialized

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	data	data	data	data	data	data	data	data
0	0	0	0	0	0	0	0								
$0				$0											

* = Reserved, program as zero

Application: _____ Date: _____

_____ Programmer: _____

SSI

WL1	WL0	Description
0	0	8 bits per word
0	1	10 bits per word
1	0	12 bits per word
1	1	16 bits per word

PSR	Description
0	Prescaler disabled
1	Prescaler ÷8 enabled

Frame Rate Divider Bits

Prescale Modulus Bits

SSI Receive Control Register (SCRRX)
X:$FFD4
Reset = $0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PSR	WL1	WL0	DC4	DC3	DC2	DC1	DC0	PM7	PM6	PM5	PM4	PM3	PM2	PM1	PM0

SSI Transmit Control Register (SCRTX)
X:$FFD3
Reset = $0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PSR	WL1	WL0	DC4	DC3	DC2	DC1	DC0	PM7	PM6	PM5	PM4	PM3	PM2	PM1	PM0

Application: _____ Date: _____

_____ Programmer: _____

SSI

SHFD	Description
0	Data sent MSB first
1	Data sent LSB first

RXD, TXD, SYN
See Table 8-5, "Frame Sync and Clock Pin Configuration."

TBF	Description
0	Transmit buffer disabled
1	Transmit buffer enabled

RBF	Description
0	Receive buffer disabled
1	Receive buffer enabled

TE	Description
0	Transmit disabled
1	Transmit enabled

RE	Description
0	Receive disabled
1	Receive enabled

TIE	Description
0	Transmit interrupt disabled
1	Transmit interrupt enabled

RIE	Description
0	Receive interrupt disabled
1	Receive interrupt enabled

SCKP	Description
0	Data clocked on rising edge of bit clock
1	Data clocked on falling edge of bit clock

SSIEN	Description
0	SSI disabled
1	SSI enabled

NET	Description
0	Normal mode selected
1	Network mode selected

FSI	Description
0	Frame sync active high
1	Frame sync active low

FSL	ÿDescription
0	Frame sync is one word long
1	Frame sync is one clock bit long

EFS	Description
0	Frame sync initiated with first data bit
1	Frame sync initiated before first data bit

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SSI Control Register 2 (SCR2) X:$FFD2 Reset = $0000	RIE	TIE	RE	TE	RBF	TBF	RXD	TXD	SYN	SHFD	SCKP	SSI	NET	FSI	FSL	EFS

Application: _____ Date: _____

_____ Programmer: _____

SSI

ROE	Description
0	(Bit is cleared)
1	RX overrun occurred

TUE	Description
0	(Bit is cleared)
1	TX underrun occurred

TDE	Description
0	(Bit is cleared)
1	TX data register empty

RFS	Description
0	No frame sync
1	Frame sync occurred

RDF	Description
0	(Bit is cleared)
1	RX data register has data

TFS	Description
0	No frame sync
1	Frame sync occurred

REFS	Description
0	Frame sync on first data bit
1	Frame sync before first data bit

RDBF	Description
0	Receive buffer empty
1	Receive buffer full

RFSL	Description
0	One-word-long frame sync
1	One-clock-bit long frame sync

TDBE	Description
0	Transmit buffer empty
1	Transmit buffer full

RFSI	Description
0	Frame sync active high
1	Frame sync active low

RSCKP	Description
0	Data latched on falling clock edge
1	Data latched on rising clock edge

RSHFD	Description
0	Data received MSB first
1	Data received LSB first

SSI Status/Control Register (SCSR)
X:$FFD1
Reset = $0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	RSH FD	RSC KP	*	*	RFSI	RFSL	REFS	RDF	TDE	ROE	TUE	RFS	TFS	RDBF	TDBE
0			0	0											

* = Reserved, program as zero

Application: _____ Date: _____

_____ Programmer: _____

SSI

SSI Transmit Register (STX) X:$FFD0 Write-Only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
High Byte								Low Byte							

SSI Receive Register (SRX) X:$FFD0 Read-Only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
High Byte								Low Byte							

SSI Time Slot Register (STSR) X:$FFD5 Write-Only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Dummy Register, Written During Inactive Time Slots															

Application: _____ Date: _____

_____ Programmer: _____

Sheet 1 of 1

Timer 0

TO10	TO00	Description
0	0	TIO pin configured as input
0	1	(Reserved)
1	0	Overflow pulse
1	1	Overflow toggle

ES10	ES00	Description
0	0	Internal Phi clock ÷ 4 selected
0	1	Internal prescaler clock selected
1	0	Previous timer overflow selected
1	1	External event from TIO pin

OIE0	Description
0	Overflow interrupt disabled
1	Overflow interrupt enabled

INV	Description
0	TIO pin detects rising edge
1	TIO pin detects falling edge

TE0	Description
0	Timer 0 disabled
1	Timer 0 enabled

Timer 0/1 Control Register (TCR01)
X:$FFDF
Reset = $0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TE1	*	*	OIE1	TO11	TO01	ES11	ES01	TE0	INV	*	OIE0	TO10	TO00	ES10	ES00
X	0	0	X	X	X	X	X			0					

***** = Reserved, program as zero

X = Used for timer 1, program accordingly

Timer 0 Preload Register (TPR0)
X:$FFDE
Reset = $0000
Write-Only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Timer 0 Count Register (TCT0)
X:$FFDD
Reset = $FFFF

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Application: _____ Date: _____

_____ Programmer: _____

Sheet 1 of 1

Timer 1

ES11	ES01	Description
0	0	Internal Phi clock ÷ 4 selected
0	1	Internal prescaler clock selected
1	0	Previous timer overflow selected
1	1	External event from TIO pin

TO11	TO01	Description
0	0	TIO pin configured as input
0	1	(Reserved)
1	0	Overflow pulse
1	1	Overflow toggle

INV	Description
0	TIO pin detects rising edge
1	TIO pin detects falling edge

OIE1	Description
0	Overflow interrupt disabled
1	Overflow interrupt enabled

TE1	Description
0	Timer 1 disabled
1	Timer 1 enabled

Timer 0/1 Control Register (TCR01)
X:$FFDF
Reset = $0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TE1	*	*	OIE1	TO11	TO01	ES11	ES01	TE0	INV	*	OIE0	TO10	TO00	ES10	ES00
	0	0						X		0	X	X	X	X	X

***** = Reserved, program as zero
X = Used for timer 0, program accordingly

Timer 1 Preload Register (TPR1)
X:$FFDC
Reset = $0000
Write-Only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Timer 1 Count Register (TCT1)
X:$FFDB
Reset = $FFFF

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Application: _____ Date: _____

_____ Programmer: _____

Sheet 1 of 1

Timer 2

TO12	TO02	Description
0	0	TIO pin configured as input
0	1	(Reserved)
1	0	Overflow pulse
1	1	Overflow toggle

ES12	ES02	Description
0	0	Internal Phi clock ÷ 4 selected
0	1	Internal prescaler clock selected
1	0	Previous timer overflow selected
1	1	External event from TIO pin

OIE2	Description
0	Overflow interrupt disabled
1	Overflow interrupt enabled

INV	Description
0	TIO pin detects rising edge
1	TIO pin detects falling edge

TE2	Description
0	Timer 2 disabled
1	Timer 2 enabled

Timer 2 Control Register (TCR2)
X:$FFDA
Reset = $0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	TE2	INV	*	OIE2	TO10	TO02	ES12	ES02
0	0	0	0	0	0	0	0			0					
$0				$0											

* = Reserved, Program as 0

Timer 2 Preload Register (TPR2)
X:$FFD9
Reset = $0000
Write-Only

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Timer 2 Count Register (TCT2)
X:$FFD8
Reset = $FFFF

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Application: _____ Date: _____

_____ Programmer: _____

Sheet 1 of 1

PLL

LPST	Description
0	PLL running
1	Low-power stop mode

PLLD	Description
0	PLL active
1	PLL powered down

PLLE	Description
0	PLL disabled
1	PLL enabled

PS2	PS1	PS0	Description
0	0	0	÷1, use for 32.0 and 38.4 kHz crystals
0	0	1	Prescaler disabled
0	1	0	÷ 16
0	1	1	(Reserved)
1	0	0	÷ 64
1	0	1	(Reserved)
1	1	0	÷ 256
1	1	1	(Reserved)

TSTEN	CS1	CS0	Description
0	0	0	Phi clock
0	0	1	(Reserved)
0	1	0	Oscillator clock
0	1	1	Disabled
1	0	0	(Reserved)
1	0	1	(Reserved)
1	1	0	(Reserved)
1	1	1	Prescaler output

VCS0	Description
0	45 MHz to 70 MHz
1	10 MHz to 45 MHz

PLL Control Register 1 (PCR1)
X:$FFF3
Reset = $0200

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	PLLE	PLLD	LPST	TST EN	PS2	PS1	PS0	CS1	CS0	*	*	VCS0	*	*	*
0										0	0		0	0	0
													$0		

PLL Feedback Divider

PLL Control Register 0 (PCR0)
X:$FFF2
Reset = $0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	YD9	YD8	YD7	YD6	YD5	YD4	YD3	YD2	YD1	YD0	*	*	*	*	*
0											0	0	0	0	0

* = Reserved, program as zero

Application: _____ Date: _____

_____ Programmer: _____

Sheet 1 of 1

COP/RTI

RTE	Description
0	RTE and COP disabled
1	RTE and COP enabled

CT	Description
0	COP timer ÷ 8
1	COP timer ÷ 64

CPE	Description
0	COP disabled
1	COP enabled

RTIE	Description
0	Real-time interrupt disabled
1	Real-time interrupt disabled

RTIF	Description
0	RTE and COP disabled
1	RTE and COP enabled

RP	Description
0	Real-time prescaler disabled
1	Real-time prescaler enabled, ÷ 4

COP/RTI Divider

COP/RTI Control Register (COPCTL)
X:$FFF1
Reset = $0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CPE	CT	*	*	RTE	RTIE	RTIF	RP	DV7	DV6	DV5	DV4	DV3	DV2	DV1	DV0
		0	0												

* = Reserved, program as zero

Initialized as 111

Value Set by RP Bit

COP/RTI Divider

COP/RTI Count Register (COPCNT)
X:$FFF0
Reset = $5555

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	DV7	DV6	DV5	DV4	DV3	DV2	DV1	DV0	RP1	RP0	SC2	SC1	SC0
0	0	0													

* = Reserved, program as zero

Freescale Semiconductor, Inc.
ARCHIVED BY FREESCALE SEMICONDUCTOR, INC. 2005

Index

E

F

G

H

I

J

P

R

S

T

V

W

X

Y