Order this document by DSP56652UM/D
Rev. 0, 04/1999

DSP56652

Baseband Digital Signal Processor User's Manual

Motorola, Incorporated
Semiconductor Products Sector
6501 William Cannon Drive West
Austin TX 78735-8598

MOTOROLA

Freescale Semiconductor, Inc.

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

This manual is one of a set of three documents. Three manuals are required for complete product information: the family manual, the user's manual, and the technical data sheet.

© Copyright Motorola, Inc., 1999. All rights reserved.

Motorola reserves the right to make changes without further notice to any products herein. Motorola makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Motorola assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters which may be provided in Motorola data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Motorola does not convey any license under its patent rights nor the rights of others. Motorola products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support life, or for any other application in which the failure of the Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the part.

Motorola and are registered trademarks of Motorola, Inc. Motorola, Inc. is an Equal Opportunity/Affirmative Action Employer.

All other tradenames, trademarks, and registered trademarks are the property of their respective owners.

Table of Contents

Preface

Chapter 1
Introduction

Chapter 2
Signal/Connection Description

Chapter 3
Memory Maps

Chapter 4
Core Operation and Configuration

Chapter 5
MCU–DSP Interface

Chapter 6
External Interface Module

Chapter 7
Interrupts

Chapter 8
Queued Serial Peripheral Interface

Chapter 9
Timers

Chapter 10
Protocol Timer

Chapter 11
UART

Chapter 12
Smart Card Port

Chapter 13
Keypad Port

Chapter 14
Serial Audio and Baseband Ports

Chapter 15
JTAG Port

Appendix A
DSP56652 DSP Bootloader

Appendix B
Equates and Header Files

Appendix C
Boundary Scan Register

Appendix D
Programmer’s Reference

Appendix E
Programmer’s Data Sheets

List of Figures

List of Tables

List of Examples

Preface

This section provides information on the data conventions used in this manual, as well as a list of complete product documentation.

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).
- 1 byte = 8 bits
  1 halfword = 16 bits = 2 bytes
  1 word = 32 bits = 4 bytes
- Bits within a register are indicated AA[n:0] when more than one bit is involved in a description. For purposes of description, the bits are presented as if they were contiguous within a register, regardless of their actual physical locations in a register.
- All bits in a register are read/write unless otherwise noted.
- When a bit is described as "set," its value is 1. When a bit is described as "cleared," its value is 0.
- Register bits that are unused or reserved for future use are read as 0 and should be written with 0 to ensure future compatibility. In the register descriptions, each of these bits is indicated with a shaded box (▨).
- The word "reset" is used in three different contexts in this manual:
  - There is a reset instruction that is always written as "RESET".
  - In lower case, "reset" refers to the reset function. A leading capital letter is used as grammar dictates.
  - "Reset" refers to the Reset state.
- The word "pin" is a generic term for any pin on the chip. Because of on-chip pin multiplexing, more than one signal may be present on any given pin.
- Pins or signals that are asserted low (made active when pulled to ground) have an overbar over their name; for example, the $\overline{\text{SS0}}$ pin is asserted low.

- Hex values are indicated with a dollar sign ($) preceding the hex value as follows: X:$FFFF is the X memory address for the Interrupt Priority Register—Core (IPR-C).

Code examples are displayed in a monospaced font, as shown in Example 1.

Example 1. Code Example

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

- In code examples, the names of pins or signals that are asserted low are preceded by a tilde. In the previous example, line 3 refers to the $\overline{\text{SS0}}$ pin (shown as `~SS0`).
- 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}. These conventions are summarized in Table 1.

Table 1. Signal States

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

Documentation

This manual (DSP56652UM/D) is one of a set of five documents that provides complete product information for the DSP56652. The other four documents include the following:

- *M•CORE Reference Manual* (MCORERM/AD)
- *MMC2001 Reference Manual* (MMC2001M/AD
- *DSP56600 Family Manual* (DSP56600FM/AD)
- *DSP56652 Technical Data Sheet* (DSP56652/D)

Chapter 1
Introduction

Motorola designed the ROM-based DSP56652 to support the rigorous demands of the cellular subscriber market. The high level of on-chip integration in the DSP56652 minimizes application system design complexity and component count, resulting in very compact implementations. This integration also yields very low power consumption and cost-effective system performance. The DSP56652 chip combines Motorola’s 32-bit M•CORE™ MicroRISC Engine and the DSP56600 Digital Signal Processor (DSP) core with on-chip memory, a protocol timer, and custom peripherals to provide a single-chip cellular base-band processor. A block diagram of the 56652 is shown in Figure 1-1.

1.1 DSP56652 Key Features

The following list summarizes the key features of the DSP56652.

- M•CORE (MCU) core
  - 32-bit load/store M•CORE RISC architecture
  - Fixed 16-bit instruction length
  - 16-entry 32-bit general-purpose register file
  - 32-bit internal address and data buses
  - Efficient four-stage, fully interlocked execution pipeline
  - Single-cycle execution for most instructions, two cycles for branches and memory accesses
  - Special branch, byte, and bit manipulation instructions
  - Support for byte, halfword, and word memory accesses
  - Fast interrupt support via vectoring/auto-vectoring and a 16-entry dedicated alternate register file

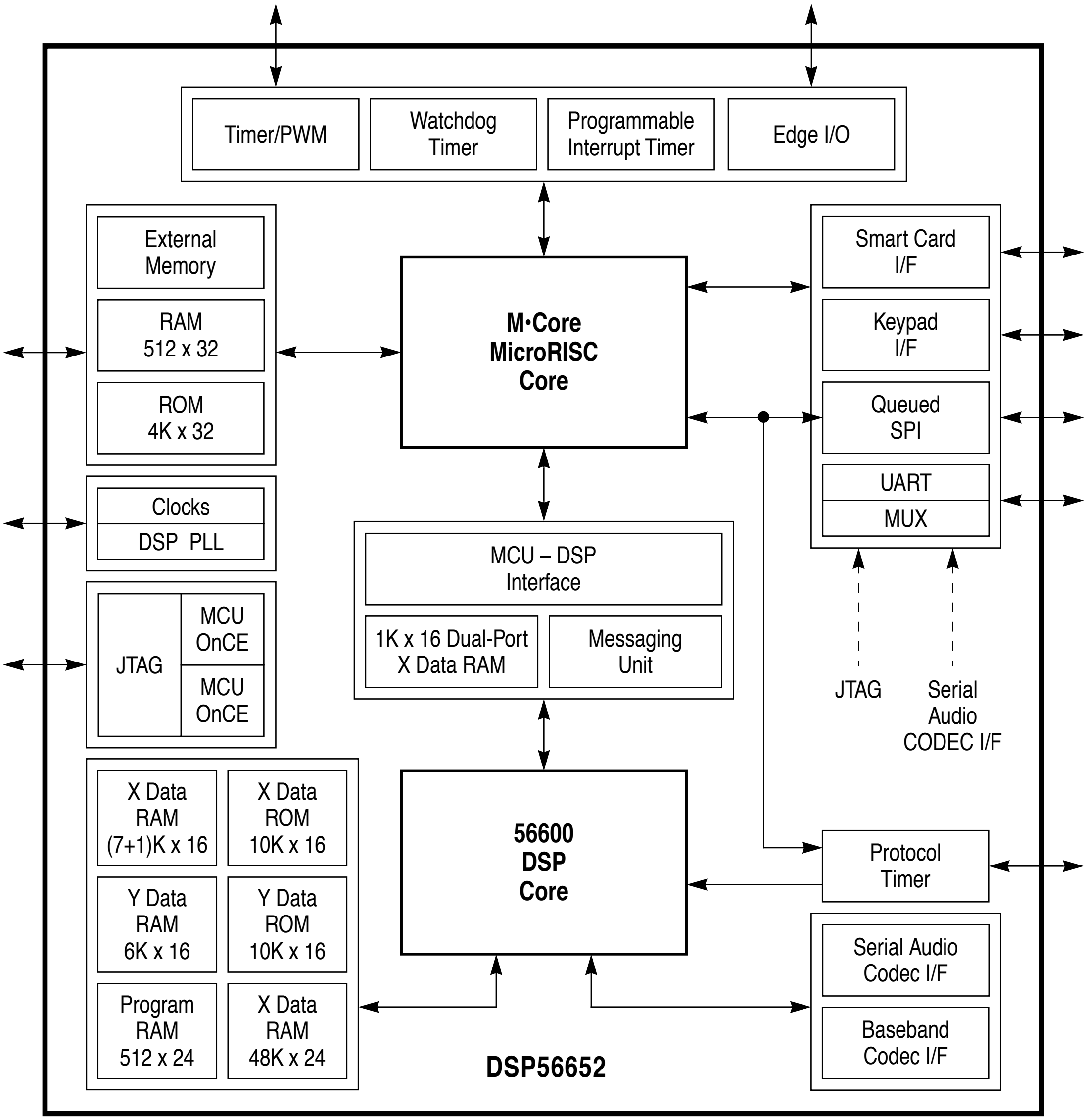


Figure 1-1. DSP56652 Block Diagram

- DSP core
  - — DSP56600 architecture
  - — Single-cycle arithmetic instructions
  - — Fully pipelined 16 × 16-bit parallel multiply accumulator (MAC)
  - — Two 40-bit accumulators including extension bits
  - — 40-bit parallel barrel shifter
  - — Highly parallel instruction set with unique DSP addressing modes
  - — Position-independent code support
  - — Nested hardware DO loops

  - Fast auto-return interrupts
  - On-chip support for software patching and enhancements
  - Real-time trace capability via external address bus
- On-chip memory
  - 4K × 32-bit MCU ROM
  - 512 × 32-bit MCU RAM
  - 48K × 24-bit DSP program ROM
  - 512 × 24-bit DSP program RAM
  - 10K × 16-bit DSP X data ROM
  - 10K × 16-bit DSP Y data ROM
  - (7+1)K × 16-bit X data RAM
  - 6K × 16-bit Y data RAM
- On-chip peripherals
  - Fully programmable phase-locked loop (PLL) for DSP clock generation
  - External interface module (EIM) for glueless system integration
  - External 22-bit address and 16-bit data MCU buses
  - 32-source MCU interrupt controller
  - Intelligent MCU/DSP interface (MDI) with 1K × 16-bit dual-port RAM as well as messaging status and control unit
  - Serial audio codec port (SAP)
  - Serial baseband codec port (BBP)
  - Protocol timer frees the MCU from radio channel timing events
  - Queued serial peripheral interface (QSPI)
  - Keypad port capable of scanning up to an 8 × 8 matrix keypad
  - General-purpose MCU and DSP timers
  - Pulse width modulation (PWM) output
  - Universal asynchronous receiver/transmitter (UART) with FIFO
  - IEEE 1149.1-compliant boundary scan JTAG test access port (TAP)
  - Integrated DSP/MCU On-Chip Emulation (OnCE™) module
  - DSP program address bus visibility mode for system development
  - ISO 7816-compatible smart card port

- Operating features
  - — Comprehensive static and dynamic power management
  - — MCU operating frequency: DC to 16.8 MHz at 1.8 V
  - — DSP operating frequency: DC to 58.8 MHz at 1.8 V
  - — Internal operating voltage range: 1.8–2.5 V with 3.1 V-tolerant I/O
  - — Operating temperature: –40° to 85°C ambient
  - — Package option: 15 × 15 mm, 196-lead PBGA

1.2 Architecture Overview

The DSP56652 combines the control and I/O capability of the M•CORE MCU with the data processing power of the DSP56600 core to provide a complete system solution for a cellular baseband system. The DSP subsystem has a closed architecture, meaning that all DSP memory is contained on the device and the DSP address and data buses do not appear external to the device. The MCU subsystem provides both on-chip memory and an external bus interface. Both processors provide external interrupt pins.

The two cores communicate through the MDI, which includes a block of dual-access RAM.

Each core generates its own independent clock, and the DSP core contains a PLL as part of its clock generation subsystem. Each processor and its associated peripherals have several low-power standby modes.

A single JTAG port is shared by the two cores for debug and test purposes. The JTAG port is integrated with on-chip emulation modules for both the MCU and the DSP, providing a non-intrusive way to interact with the processors and their peripherals and memory. The MCU has additional external debug pins for in-circuit emulation. The DSP program address bus is multiplexed on other DSP56652 pins.

The pins associated with most peripherals can be programmed individually to function as general-purpose input/output signals (GPIO) if their primary functions are not required. (The exceptions are the MCU pulse width modulator and general-purpose timer, which have no GPIO capability, and the SmartCard Port (SCP), whose five pins must all function either as SCP pins or GPIO (i.e., cannot be individually programmed).

1.2.1 MCU

This section describes the MCU core, peripherals, and memory.

1.2.1.1 Core Description

The M•CORE MCU utilizes a four-stage pipeline for instruction execution. The instruction fetch, instruction decode/register file read, execute, and register write-back stages operate in an overlapped fashion, allowing most instructions to execute in a single clock cycle. Sixteen general-purpose registers are provided for source operands and instruction results.

The execution unit consists of a 32-bit arithmetic/logic unit (ALU), a 32-bit barrel shifter, a find-first-one unit (FFO), result feed-forward hardware, and miscellaneous support hardware for multiplication and multiple register loads and stores. Arithmetic and logical operations are executed in a single cycle with the exception of the multiply and divide instructions. The FFO unit operates in a single clock cycle.

The program counter unit contains a PC incrementer and a dedicated branch address adder to minimize delays during change-of-flow operations. Memory load and store operations are provided for byte, halfword, and word (32-bit) data with automatic zero extension of byte and halfword load data. These instructions can execute in two clock cycles. Load and store multiple register instructions allow low overhead context save and restore operations.

A single condition code/carry (C) bit is provided for condition testing and to implement arithmetic and logical operations greater than 32 bits. A 16-entry alternate register file is provided to minimize exception processing overhead, and the CPU supports both vectored and auto-vectored interrupts.

The user programming model contains the program counter, sixteen 32-bit general-purpose registers, and the carry bit. A separate supervisor mode is provided for exception processing. The supervisor programming model includes all of the user registers plus an additional sixteen 32-bit general-purpose registers, 12 control registers, and 5 scratch registers.

For a complete description of M•CORE architecture, refer to the *M•CORE Reference Manual*.

1.2.1.2 MCU-Side Peripherals

The MCU-side peripherals for the DSP56652 support a variety of I/O functions, including radio channel timing, signal generation, periodic interrupts, smart card interface, LCD displays, and key pads.

- A **keypad port** supports up to 8 rows and 8 columns.
- The **QSPI** enables serial communication to multiple peripheral devices through a single port.

- The **SCP** provides user information to an external device through a smart card port.
- A **UART** connects to a modem or another computer.
- An **edge I/O port** enables up to eight external interrupts.
- An **interrupt controller** prioritizes up to 32 peripheral interrupts.
- Four timers are provided, including
  - a **periodic interval timer** to generate periodic interrupts
  - a **watchdog timer** to protect against system failure
  - a **pwm** and **general-purpose timer** to generate custom signals
  - a **protocol timer** with TDMA counters for radio channel control, event scheduling, QSPI triggers or generating interrupts to either core.
- MCU **OnCE** facilitates test and debug.

1.2.1.3 MCU-Side Memory

All MCU memory is 32 bits (1 word) wide. On-chip MCU memory includes 512 words of RAM and 4K words of ROM. In addition, the EIM provides a 22-bit address/16-bit data bus with control signals to access external memory. Programmable timing on this bus allows the use of a wide range of memory devices. As many as six external memory banks can be connected.

1.2.2 DSP

This section describes the DSP core, peripherals, and memory.

1.2.2.1 Core Description

The DSP56600 core contains a data arithmetic logic unit, an address generation unit, a program control unit, and program patch logic.

1.2.2.1.1 Data Arithmetic Logic Unit

The data arithmetic logic unit (ALU) performs all data arithmetic and logical operations in the DSP core. The components of the data ALU include the following:

- Four 16-bit input general purpose registers: X1, X0, Y1, and Y0
- A parallel, fully pipelined MAC
- Six data ALU registers (A2, A1, A0, B2, B1, and B0) that are concatenated into two general-purpose, 40-bit accumulators, A and B
- An accumulator shifter that is an asynchronous parallel shifter with a 40-bit input and a 40-bit output

- A bit field unit (BFU) with a 40-bit barrel shifter
- Two data bus shifter/limiter circuits

The data ALU registers can be read or written over the X data bus (XDB) and the Y data bus (YDB) as 16- or 32-bit operands. The source operands for the data ALU, which can be 16, 32, or 40 bits, always originate from data ALU registers. The results of all data ALU operations are stored in an accumulator.

A seven-stage pipeline executes one instruction per clock cycle. The destination of every arithmetic operation can be used as a source operand for the immediate following operation without penalty.

The MAC unit comprises the main arithmetic processing unit of the DSP core and performs all of the calculations on data operands. For arithmetic instructions, the unit accepts as many as three input operands and outputs one 40-bit result, formatted as Extension:Most Significant Product:Least Significant Product (EXT:MSP:LSP).

The multiplier executes 16-bit × 16-bit, parallel, fractional multiplies, between two's-complement signed, unsigned, or mixed operands. The 32-bit product is right-justified and added to the 40-bit contents of either the A or B accumulator. A 40-bit result can be stored as a 16-bit operand. The LSP can either be truncated or rounded into the MSP. Rounding is performed if specified.

1.2.2.1.2 Address Generation Unit

The address generation unit (AGU) performs the effective address calculations using integer arithmetic necessary to address data operands in memory and contains the registers used to generate the addresses. It implements four types of arithmetic: linear, modulo, multiple wrap-around modulo, and reverse-carry. The AGU operates in parallel with other chip resources to minimize address-generation overhead.

The AGU is divided into two halves, each with its own address ALU. Each address ALU has four sets of register triplets, and each register triplet is composed of an address register, an offset register, and a modifier register. The two address ALUs are identical. Each contains a 16-bit full adder (referred to as an offset adder).

A second full adder (referred to as a modulo adder) adds the summed result of the first full adder to a modulo value that is stored in its respective modifier register. A third full adder (called a reverse-carry adder) is also provided.

The offset adder and the reverse-carry adder are in parallel and share common inputs. The only difference between them is that they carry propagates in opposite directions. Test logic determines which of the three summed results of the full adders is output.

Each address ALU can update one address register from its respective address register file during one instruction cycle. The contents of the associated modifier register specifies the type of arithmetic to be used in the address register update calculation. The modifier value is decoded in the address ALU.

1.2.2.1.3 Program Control Unit

The program control unit (PCU) performs instruction prefetch, instruction decoding, hardware DO loop control and exception processing. The PCU implements a seven-stage pipeline and controls the different processing states of the DSP core. The PCU consists of three hardware blocks:

- program decode controller (PDC)
- program address generator (PAG)
- program interrupt controller (PIC)

The PDC decodes the 24-bit instruction loaded into the instruction latch and generates all signals necessary for pipeline control. The PAG contains all the hardware needed for program address generation, system stack and loop control. The PIC arbitrates among all interrupt requests and generates the appropriate interrupt vector address.

The PCU implements its functions using the following registers:

- PC—Program Counter register
- SR—Status Register
- LA—Loop Address register
- LC—Loop Counter register
- VBA—Vector Base Address register
- SZ—Size register
- SP—Stack Pointer
- OMR—Operating Mode Register
- SC—Stack Counter register

The PCU also includes a hardware System Stack (SS).

1.2.2.1.4 Program Patch Logic

The program patch logic (PPL) block provides a way to adjust program code in the on-chip ROM without generating a new mask. Implementing the code correction is done by replacing a piece of ROM-based code with a patch program stored in RAM. The PPL consists of four patch address registers (PAR0–PAR3) and four patch address comparators. Each PAR points to a starting location in the ROM code where the program flow is to be changed. The PC register in the PCU is compared to each PAR. When an address of a fetched instruction is identical to an address stored in one of the PARs, the program data bus is forced to a corresponding JMP instruction, replacing the instruction that otherwise would have been fetched from the ROM.

1.2.2.2 DSP-Side Peripherals

The DSP-side peripherals for the DSP56652 are primarily targeted at handling baseband and audio processing.

- Two improved synchronous serial ports connect to external codecs to process received baseband information.
  - The **SAP** connects to a standard audio codec. This port also provides a general-purpose timer.
  - The **BBP** connects to a standard RF/IF codec.
- DSP **OnCE** facilitates test and debug.

1.2.2.3 DSP-Side Memory

All DSP memory is contained on-chip. DSP program memory is 24 bits wide, while data memory is 16 bits (1 halfword) wide. Program ROM is 48K by 24-bits, and program RAM is 512 by 24-bits. Data memory is organized into two separate areas, X and Y, each accessed by its own address and data buses. X and Y data ROM are 10K by 16 bits each. X data RAM is 7K by 16 bits, and Y data RAM is 6K by 16 bits. In addition, 1K of X data memory space serves as dual-port RAM for the MDI.

1.2.3 MCU–DSP Interface

The MDI provides a way for the MCU and DSP cores to communicate with each other. It contains a message and control unit as well as 1K × 16-bit dual-ported RAM.

Chapter 2
Signal/Connection Description

The DSP56652 input and output signals are organized into functional groups in Table 2-1 below and in Figure 2-1 on page 2-2. Many of the pins in the DSP56652 have multiple functions. In Table 2-1, pin function is described to reflect primary pin function. Subsequent tables in this section are named for these primary functions and provide full descriptions of all signals on the pins.

Table 2-1. DSP56652 Signal Functional Group Allocations

Functional Group		Number of Signals	Detailed Description
Power (V_{CCX})		20	Table 2-2
Function-specific ground (GND_X)		17	Table 2-3
General ground (GND)		20	
PLL and clocks		5	Table 2-4
External Interface Module (EIM)	Address bus	22	Table 2-5
	Data bus	16	
	Bus control	4	Table 2-6
	Chip selects	6	Table 2-7
Reset, mode, and multiplexer control		5	Table 2-8
External interrupts		9	Table 2-9
Protocol Timer		8	Table 2-10
Keypad port		16	Table 2-11
UART		4	Table 2-12
Queued Serial Peripheral Interface (QSPI)		8	Table 2-13
Smart Card Port (SCP)		5	Table 2-14
Serial Audio Codec Port (SAP)		6	Table 2-15
Baseband Codec Port (BBP)		6	Table 2-16
Development & Test	Emulation port	6	Table 2-17
	Debug control port	2	Table 2-18
	JTAG test access port (TAP)	6	Table 2-19

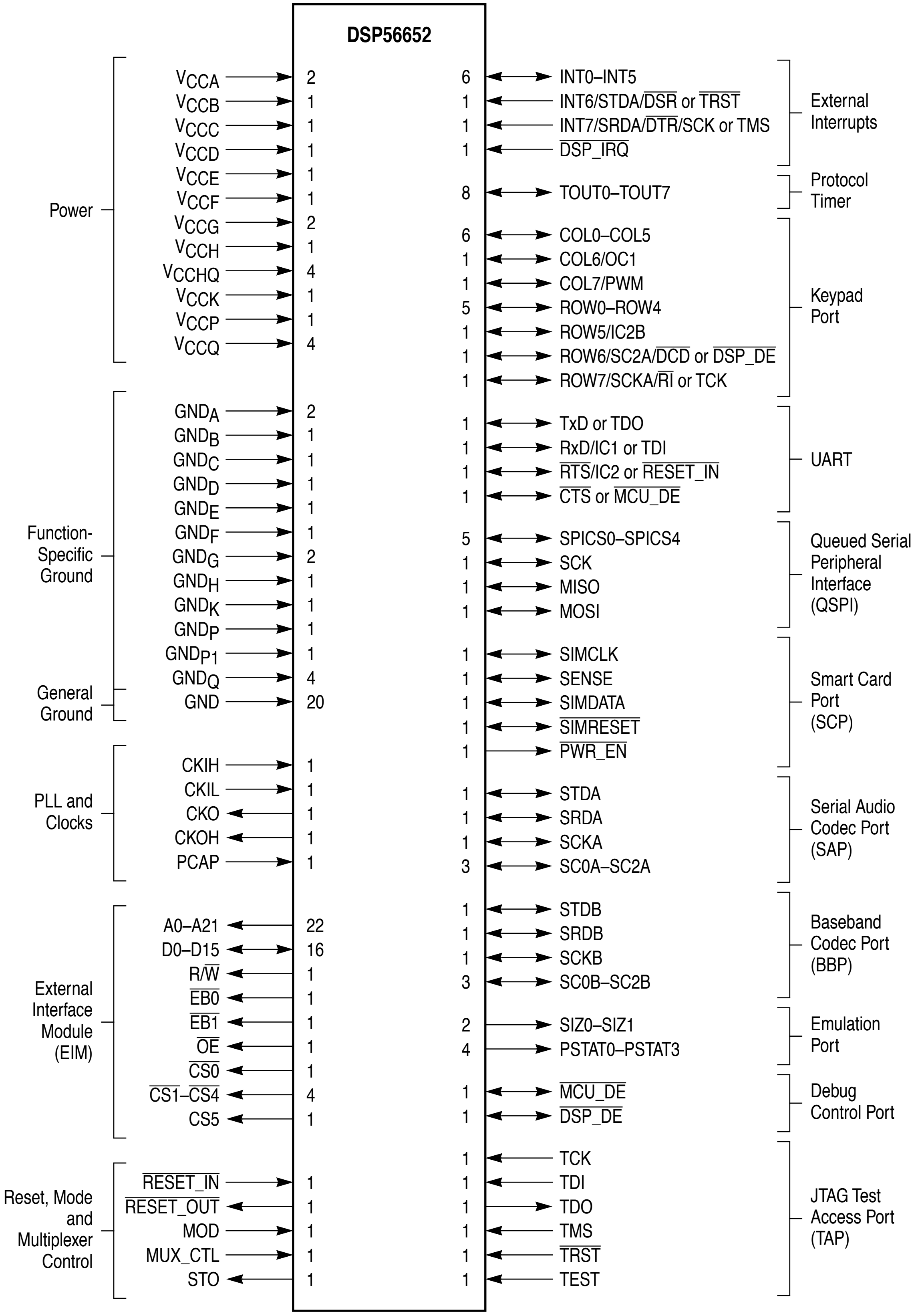


Figure 2-1. Signal Group Organization

2.1 Power

The DSP56652 power pins are listed in Table 2-2.

Table 2-2. Power

Power Signals	Description
V_{CCA}	**Address bus power**—Lines C1 and F1 supply isolated power to the address bus drivers.
V_{CCB}	**SIM power**—Line L8 supplies isolated power for the smart card I/O drivers.
V_{CCC}	**Bus control power**—Line L3 supplies power to the bus control logic.
V_{CCD}	**Data bus power**—These lines supply power to the data bus.
V_{CCE}	**Audio codec port power**—This line supplies power to audio codec I/O drivers.
V_{CCF}	**Clock output power**—This line supplies a quiet power source for the CKOUT output. Ensure that the input voltage to this line is well-regulated and uses an extremely low impedance path to tie to the V_{CC} power rail. Use a 0.1 μF bypass capacitor located as close as possible to the chip package to connect between the V_{CCF} line and the GND_F line.
V_{CCG}	**GPIO power**—This line supplies power to the GPIO, keypad, data port, interrupts, STO, and JTAG I/O drivers.
V_{CCH}	**Baseband codec and timer power**—This line supplies power to the baseband codec and Timer I/O drivers.
V_{CCHQ}	**Quiet power high**—These lines supply a quiet power source to the pre-driver voltage converters. This value should be equal to the maximum value of the power supplies of the chip I/O drivers (i.e., the maximum of V_{CCA}, V_{CCB}, V_{CCC}, V_{CCD}, V_{CCE}, V_{CCF}, V_{CCG}, V_{CCH}, and V_{CCK}).
V_{CCK}	**Emulation port power**—This line supplies power to the emulation port I/O drivers.
V_{CCP}	**Analog PLL circuit power**—This line is dedicated to the analog PLL circuits and must remain noise-free to ensure stable PLL frequency and performance. Ensure that the input voltage to this line is well-regulated and uses an extremely low impedance path to tie to the V_{CC} power rail. Use a 0.1 μF capacitor and a 0.01 μF capacitor located as close as possible to the chip package to connect between the V_{CCP} line and the GND_P and GND_{P1} lines.
V_{CCQ}	**Quiet power**—These lines supply a quiet power source to the internal logic circuits. Ensure that the input voltage to this line is well-regulated and uses an extremely low impedance path to tie to the V_{CC} power rail. Use a 0.1 μF bypass capacitor located as close as possible to the chip package to connect between the V_{CCQ} lines and the GND_Q lines.

2.2 Ground

The DSP56652 ground pins are listed in Table 2-3.

Table 2-3. Ground

Ground Signals	Description
GND_A	**Address bus ground**—These lines connect system ground to the address bus.
GND_B	**SIM ground**—These lines connect system ground to the smart card bus.
GND_C	**Bus control ground**—This line connects ground to the bus control logic.
GND_D	**Data bus ground**—These lines connect system ground to the data bus.
GND_E	**Audio codec port ground**—These lines connect system ground to the audio codec port.
GND_F	**Clock output ground**—This line supplies a quiet ground connection for the clock output drivers.
GND_G	**GPIO ground**—These lines connect system ground to GPIO, keypad, data port, interrupts, STO, and JTAG I/O drivers.
GND_H	**Baseband codec and timer ground**—These lines connect system ground to the baseband codec and timer I/O drivers.
GND_K	**Emulation port ground**—These lines connect system ground to the emulation port I/O drivers.
GND_P	**Analog PLL circuit ground**—This line supplies a dedicated quiet ground connection for the analog PLL circuits.
GND_{P1}	**Analog PLL circuit ground**—This line supplies a dedicated quiet ground connection for the analog PLL circuits.
GND_Q	**Quiet ground**—These lines supply a quiet ground connection for the internal logic circuits.
GND	**Substrate ground**—These lines must be tied to ground.

2.3 Clock and Phase-Locked Loop

The pins controlling DSP56652 clocks and PLL are listed in Table 2-4.

Table 2-4. PLL and Clock Signals

Signal Name	Type	Reset State	Signal Description
CKIH	Input	Input	**High frequency clock input**—This input can be connected to either a CMOS square wave or sinusoid clock source.
CKIL	Input	Input	**Low frequency clock input**—This input should be connected to a square wave with a frequency less than or equal to CKIH. This is the default input clock after reset.
CKO	Output	Driven low	**DSP/MCU output clock**—This signal provides an output clock synchronized to the DSP or MCU core internal clock phases, according the selected programming option. The choices of clock source and enabling/disabling the output signal are software selectable.
CKOH	Output	Driven low	**High frequency clock output**—This signal provides an output clock derived from the CKIH input. This signal can be enabled or disabled by software.
PCAP	Input/ Output	Indeterminate	**PLL capacitor**—This signal is used to connect the required external filter capacitor to the PLL filter.

2.4 External Interface Module

The bus, bus control, and chip select signals of the EIM are listed in Table 2-5, Table 2-6, andTable 2-7 respectively.

Table 2-5. Address and Data Buses

Signal Name	Type	Reset State	Signal Description
A0–A21	Output	Driven low	**Address bus—**These signals specify the address for external memory accesses. If there is no external bus activity, A0–A21 remain at their previous values to reduce power consumption.
D0–D15	Input/ Output	Input	**Data bus**—These signals provide the bidirectional data bus for external memory accesses. They remain in their previous logic state when there is no external bus activity to reduce power consumption.

Table 2-6. Bus Control

Signal Name	Type	Reset State	Signal Description
R/$\overline{\text{W}}$	Output	Driven high	**Read/Write**—This signal indicates the bus access type. A high signal indicates a bus read. A low signal indicates a write to the bus. This signal can also be used as a memory write enable ($\overline{\text{WE}}$) signal. When accessing a peripheral chip, the signal acts as a read/write.
$\overline{\text{EB0}}$	Output	Driven high	**Enable Byte 0**—When driven low, this signal indicates access to data byte 0 (D8–D15) during a read or write cycle. This pin may also act as a write byte enable, if so programmed. This output is used when accessing 8-bit wide SRAM.
$\overline{\text{EB1}}$	Output	Driven high	**Enable Byte 1**—When driven low, this signal indicates access to data byte 1 (D0–D7) during a read or write cycle. This pin may also act as a write byte enable, if so programmed. This output is used when accessing 8-bit wide SRAM.
$\overline{\text{OE}}$	Output	Driven high	**Bus select—**When driven low, this signal indicates that the current bus access is a read cycle and enables slave devices to drive the data bus with a read.

Table 2-7. Chip Select Signals

Signal Name	Type	Reset State	Signal Description
$\overline{\text{CS0}}$	Output	Chip-driven	**Chip Select 0—**This signal is asserted low based on the decode of the internal address bus bits A[31:24] and is typically used as the external flash memory chip select. After reset, accesses using CS0 have a default of 15 wait states.
$\overline{\text{CS1}}$–$\overline{\text{CS4}}$	Output	Driven high	**Chip Selects 1–4—**These signals are asserted low based on the decode of the internal address bus bits A[31:24] of the access address. When not configured as chip selects, these signals become general purpose outputs (GPOs). After reset, these signals are GPOs that are driven high.
CS5	Output	Driven low	**Chip Select 5—**This signal is asserted high based on the decode of the internal address bus bits A[31:24] of the access address. When not configured as a chip select, this signal functions as a GPO. After reset, this signal is a GPO that is driven low.

2.5 Reset, Mode, and Multiplexer Control

The reset, mode select, and multiplexer control pins are listed in Table 2-8.

Table 2-8. Reset, Mode, and Multiplexer Control Signals

<table>
<tr><th>Signal Name</th><th>Type</th><th>Reset State</th><th>Signal Description</th></tr>
<tr><td>RESET_IN</td><td>Input</td><td>Input</td><td>Reset Input—This signal is an active low Schmitt trigger input that provides a reset signal to the internal circuitry. The input is valid if it is asserted for at least three CKIL clock cycles.
Note: If MUX_CTL is held high, the RTS signal of the serial data port (UART) becomes the RESET_IN input line. (See Table 2-12 on page 2-13.)</td></tr>
<tr><td>RESET_OUT</td><td>Output</td><td>Pulled low</td><td>Reset Output—This signal is asserted low for at least seven CKIL clock cycles under any one of the following three conditions:
RESET_IN is pulled low for at least three CKIL clock cycles
The alternate RESET_IN signal is enabled by MUX_CTL and is pulled low for at least three CKIL clock cycles
The watchdog count expires.
This signal is asserted immediately after the qualifier detects a valid RESET_IN signal, remains asserted during RESET_IN assertion, and is stretched for at least seven more CKIL clock cycles after RESET_IN is deasserted. Three CKIL clock cycles before RESET_OUT is deasserted, the MCU boot mode is latched from the MOD signal.</td></tr>
<tr><td>MOD</td><td>Input</td><td>Input</td><td>Mode Select—This signal selects the MCU boot mode during hardware reset. It should be driven at least four CKIL clock cycles before RESET_OUT is deasserted.
MOD driven high—MCU fetches the first word from internal MCU ROM.
MOD driven low—MCU fetches the first word from external flash memory.</td></tr>
<tr><td>MUX_CTL</td><td>Input</td><td>Input</td><td>Multiplexer Control—This input allows the designer to select an alternate set of pins to be used for RESET_IN, the debug control port signals, and the JTAG signals as follows:
<table>
<tr><th></th><th>Normal
(MUX_CTL low)</th><th>Alternate
(MUX_CTL high)</th></tr>
<tr><td rowspan="2">Interrupt signals (See Table 2-9)</td><td>INT6/STDA/DSR</td><td>TRST</td></tr>
<tr><td>INT7/SRDA/DTR/SCLK</td><td>TMS</td></tr>
<tr><td rowspan="2">Keypad signals (See Table 2-11)</td><td>ROW6/SC2A/DCD</td><td>DSP_DE</td></tr>
<tr><td>ROW7/SCKA/RI</td><td>TCK</td></tr>
<tr><td rowspan="4">Serial Data Port (UART) signals (See Table 2-12)</td><td>TxD</td><td>TDO</td></tr>
<tr><td>RxD/IC1</td><td>TDI</td></tr>
<tr><td>RTS/IC2A</td><td>RESET_IN</td></tr>
<tr><td>CTS</td><td>MCU_DE</td></tr>
</table></td></tr>
<tr><td>STO</td><td>Output</td><td>Chip driven</td><td>Soft Turn Off—This is a GPO pin. Its logic state is not affected by reset.</td></tr>
</table>

2.6 Internal Interrupts

With the exception of alternate signal functions $\overline{\text{TRST}}$, TMS, and $\overline{\text{DSP\_IRQ}}$, the signals described in Table 2-9 are GPIO when not programmed otherwise, and default as general-purpose inputs (GPI) after reset.

Table 2-9. Interrupt Signals

Signal Name	Type	Reset State	Signal Description
INT0–INT5	Input or Output	Input	**Interrupts 0–5**[1]—These signals can be programmed as interrupt inputs or GPIO signals. As interrupt inputs, they can be programmed to be level sensitive, positive edge-triggered, or negative edge-triggered.
Normal—MUX_CTL driven low			
INT6	Input or Output	Input	**Interrupt 6**[1]—When selected, this signal can be programmed as an interrupt input or a GPIO signal. As an interrupt input, it can be programmed to be level sensitive, positive edge-triggered, or negative edge-triggered.
STDA	Output		**Audio Codec Serial Transmit Data** (alternate)—When programmed as STDA, this signal transmits data from the serial transmit shift register in the serial audio codec port. Note: When this signal functions as STDA, the primary STDA signal is disabled. (See Table 2-15 on page 2-16.)
$\overline{\text{DSR}}$	Output		**Data Set Ready**—When programmed as GPIO output, this signal can be used as the $\overline{\text{DSR}}$ output for the serial data port. (See Table 2-12.)
Alternate—MUX_CTL driven high			
$\overline{\text{TRST}}$	Input	Input	**Test Reset (alternate)**—When selected, this signal acts as the $\overline{\text{TRST}}$ input for the JTAG test access port (TAP) controller. The signal is a Schmitt trigger input that asynchronously initializes the JTAG test controller when asserted. Note: When this signal is enabled, the primary $\overline{\text{TRST}}$ signal is disconnected from the TAP controller. (See Table 2-19 on page 2-19.)

Table 2-9. Interrupt Signals (Continued)

Signal Name	Type	Reset State	Signal Description
Normal—MUX_CTL driven low			
INT7¯	Input or Output	Input	**Interrupt 7**[1]—When selected, this signal can be programmed as an interrupt input or a GPIO signal. As an interrupt input, it can be programmed to be level sensitive, positive edge-triggered, or negative edge-triggered.
SRDA	Input		**Audio Codec Serial Receive Data** (alternate)—When programmed as SRDA, this signal receives data into the serial receive shift register in the serial audio codec port. Note: When this signal is used as SRDA, the primary SRDA signal is disabled. (See Table 2-15 on page 2-16.)
$\overline{\text{DTR}}$	Input		**Data Terminal Ready**—When programmed as GPIO, this signal is used as the $\overline{\text{DTR}}$ positive and negative edge-triggered interrupt input for the serial data port. (See Table 2-12 on page 2-13.)
SCLK	Input		**Serial Clock**–This signal provides the input clock for the serial data port (UART). (See Table 2-12 on page 2-13.)
Alternate—MUX_CTL driven high			
TMS	Input	Input	**Test Mode Select (alternate)**—This signal is the TMS input for the JTAG test access port (TAP) controller. TMS is used to sequence the TAP controller state machine. It is sampled on the rising edge of TCK. Note: When this signal is enabled, the primary TMS signal is disconnected from the TAP controller. See Table 2-19 on page 2-19
$\overline{\text{DSP\_IRQ}}$	Input	Input	**DSP External Interrupt Request**—This active low Schmitt trigger input can be programmed as a level-sensitive or negative edge-triggered maskable interrupt request input during normal instruction processing. If the DSP is in the STOP state and DSP_IRQ is asserted, the DSP exits the STOP state.

1. As Schmitt trigger interrupt inputs, these signals can be programmed to be level sensitive, positive edge-triggered, or negative edge-triggered. An edge-triggered interrupt is initiated when the input signal reaches a particular voltage level, regardless of the rise or fall time. However, as signal transition time increases, the probability of noise generating extraneous interrupts also increases.

2.7 Protocol Timer

Table 2-10 describes the eight Protocol Timer signals.

Table 2-10. Protocol Timer Output Signals

Name	Type	Reset State	Signal Description
TOUT0–TOUT7	Input or Output	Input	**Timer Outputs 0–7**—These timer output signals can also be configured as GPIO. The default function after reset is GPI.

2.8 Keypad Port

With the exception of alternate signal functions $\overline{\text{DSP\_DE}}$ and TCK, the signals described in Table 2-11 are GPIO when not programmed otherwise and default as GPI after reset.

Table 2-11. Keypad Port Signals

Signal Name	Type	Reset State	Signal Description
COL0–COL5	Input or Output	Input	**Column Strobe 0–5—**As keypad column strobes, these signals can be programmed as regular or open drain outputs.
COL6 OC1	Input or Output Output	Input	**Column Strobe 6—**As a keypad column strobe, this signal can be programmed as regular or open drain output. **MCU Timer Output Compare 1**—This signal is the MCU timer output compare 1 signal. Programming of this signal function is performed using the general port control register and the keypad control register.
COL7 PWM	Input or Output Output	Input	**Column Strobe 7—**As a keypad column strobe, this signal can be programmed as regular or open drain output. **PWM Output** Note: Programming of this signal function is performed using the general port control register and the keypad control register.
ROW0–ROW4	Input or Output	Input	**Row Sense 0–4—**These signals function as keypad row senses.
ROW5 IC2	Input or Output Input	Input	**Row Sense 5—**This signal functions as a keypad row sense. **MCU Timer Input Capture 2**—This signal is the input capture for the MCU input capture 2 timer. Note: Programming of this signal function is performed using the general port control register.

Table 2-11. Keypad Port Signals (Continued)

Signal Name	Type	Reset State	Signal Description
Normal—MUX_CTL driven low			
ROW6	Input or Output	Input	**Row Sense 6—**This signal functions as a keypad row sense.
SC2A	Input or Output		**Audio Codec Serial Control 2 (alternate)**—This signal provides I/O frame synchronization for the serial audio codec port. In synchronous mode, the signal provides the frame sync for both the transmitter and receiver. In asynchronous mode, the signal provides the frame sync for the transmitter only. Note: When this signal is used as SC2A, the primary SC2A signal is disabled. (See Table 2-15 on page 2-16.)
$\overline{\text{DCD}}$	Output		**Data Carrier Detect**—This signal can be used as the $\overline{\text{DCD}}$ output for the serial data port. (See Table 2-12 on page 2-13.) Note: Programming of these functions is done through the general port control register and the SAP control register.
Alternate—MUX_CTL driven high			
$\overline{\text{DSP\_DE}}$	Input or Output	Input	**Digital Signal Processor Debug Event**—This signal functions as $\overline{\text{DSP\_DE}}$. In normal operation, $\overline{\text{DSP\_DE}}$ is an input that provides a means to enter the debug mode of operation from an external command converter. When the DSP enters the debug mode due to a debug request or as the result of meeting a breakpoint condition, it asserts $\overline{\text{DSP\_DE}}$ as an output signal for three clock cycles to acknowledge that it has entered debug mode. Note: When this signal is enabled, the primary $\overline{\text{DSP\_DE}}$ signal is disabled.

Table 2-11. Keypad Port Signals (Continued)

Signal Name	Type	Reset State	Signal Description
Normal—MUX_CTL driven low			
ROW7	Input or Output	Input	**Row Sense 7—**This signal functions as a keypad row sense.
SCKA	Input		**Audio Codec Serial Clock (alternate)**—This signal provides the serial bit rate clock for the serial audio codec port. In synchronous mode, the signal provides the clock input or output for both the transmitter and receiver. In asynchronous mode, the signal provides the clock for the transmitter only. Note: When this signal is used as SCKA, the primary SCKA signal is disabled. (See Table 2-15 on page 2-16.)
$\overline{\text{RI}}$	Output		**Ring Indicator**—This signal can be used as the $\overline{\text{RI}}$ output for the serial data port. (See Table 2-12 on page 2-13.) Note: Programming of these functions is done through the general port control register and the SAP control register.
Alternate—MUX_CTL driven high			
TCK	Input	Input	**Test Clock (alternate)**—This signal provides the TCK input for the JTAG test access port (TAP) controller. TCK is used to synchronize the JTAG test logic. Note: When this signal is enabled, the primary TCK signal is disconnected from the TAP controller. (See Table 2-19 on page 2-19.)

2.9 UART

With the exception of alternate signal functions TDO, TDI, $\overline{\text{RESET\_IN}}$, and $\overline{\text{MCU\_DE}}$, the signals described in Table 2-12 are GPIO when not programmed otherwise and default as GPI after reset.

The remaining UART signals can be implemented with GPIO pins. Suggested allocations include the following:

- $\overline{\text{DSR}}$—alternate function for INT6. (See Table 2-9 on page 2-8.)
- $\overline{\text{DTR}}$—alternate function for INT7. (See Table 2-9 on page 2-8.)
- $\overline{\text{DCD}}$—alternate function for ROW6. (See Table 2-11 on page 2-10.)
- $\overline{\text{RI}}$—alternate function for ROW7. (See Table 2-11 on page 2-10.)

Table 2-12. UART Signals

Signal Name	Type	Reset State	Signal Description
Normal—MUX_CTL driven low			
TxD	Input or Output	Input	**UART Transmit—**This signal transmits data from the UART.
Alternate—MUX_CTL driven high			
TDO	Output		**Test Data Output (alternate)—**This signal provides the TDO serial output for test instructions and data from the JTAG TAP controller. TDO is a tri-state signal that is actively driven in the shift-IR and shift-DR controller states. Note: When this signal is enabled, the primary TDO signal is disconnected from the TAP controller. (See Table 2-19 on page 2-19.)
Normal—MUX_CTL driven low			
RxD	Input or Output	Input	**UART Receive—**This signal receives data into the UART.
IC1	Input		**MCU Timer Input Capture 1**—The signal connects to an input capture/output compare timer used for autobaud mode support.
Alternate—MUX_CTL driven high			
TDI	Input	Input	**Test Data In (alternate)**—This signal provides the TDI serial input for test instructions and data for the JTAG TAP controller. TDI is sampled on the rising edge of TCK. Note: When this signal is enabled, the primary TDI signal is disconnected from the TAP controller. (See Table 2-19 on page 2-19.)

Table 2-12. UART Signals (Continued)

Signal Name	Type	Reset State	Signal Description
Normal—MUX_CTL driven low			
$\overline{\text{RTS}}$	Input or Output	Input	**Request To Send—**This signal functions as the UART $\overline{\text{RTS}}$ signal.
IC2	Input		**MCU Timer Input Capture 2**—This signal connects to an input capture timer channel.
Alternate—MUX_CTL driven high			
$\overline{\text{RESET\_IN}}$	Input	Input	**Reset Input**—This signal is an active low Schmitt trigger input that provides a reset signal to the internal circuitry. The input is valid if it is asserted for at least three CKIL clock cycles. Note: When this signal is enabled, the primary $\overline{\text{RESET\_IN}}$ signal is disabled. (See Table 2-8 on page 2-7.)
Normal—MUX_CTL driven low			
$\overline{\text{CTS}}$	Input or Output	Input	**Clear To Send—**This signal functions as the UART $\overline{\text{CTS}}$ signal.
Alternate—MUX_CTL driven high			
$\overline{\text{MCU\_DE}}$	Input or Output		**Microcontroller Debug Event**—As an input, this signal provides a means to enter the debug mode of operation from an external command converter. As an output signal, it acknowledges that the MCU has entered the debug mode. When the MCU enters the debug mode due to a debug request or as the result of meeting a breakpoint condition, it asserts $\overline{\text{MCU\_DE}}$ as an output signal for several clock cycles. Note: When this signal is enabled, the primary $\overline{\text{MCU\_DE}}$ signal is disabled.

2.10 QSPI

The signals described in Table 2-13 are GPIO when not programmed otherwise and default as GPI after reset.

Table 2-13. QSPI Signals

Signal Name	Type	Reset State	Signal Description
SPICS0– SPICS4	Output	Input	**Serial Peripheral Interface Chip Select 0–4**—These output signals provide chip select signals for the QSPI. The signals are programmable as active high or active low.
SCK	Output	Input	**Serial Clock**—This output signal provides the serial clock from the QSPI for the accessed peripherals. The delay (number of clock cycles) between the assertion of the chip select signals and the first transmission of the serial clock is programmable. The polarity and phase of SCK are also programmable.
MISO	Input	Input	**Synchronous Master In Slave Out**—This input signal provides serial data input to the QSPI. Input data can be sampled on the rising or falling edge of SCK and received in QSPI RAM most significant bit or least significant bit first.
MOSI	Output	Input	**Synchronous Master Out Slave In**—This output signal provides serial data output from the QSPI. Output data can be sampled on the rising or falling edge of SCK and transmitted most significant bit or least significant bit first.

2.11 SCP

The signals described in Table 2-14 are GPIO when not programmed otherwise, and default as GPI after reset.

Table 2-14. SCP Signals

Signal Name	Type	Reset State	Signal Description
SIMCLK	Output	Input	**SIM Clock—**This signal is an output clock from the SCP to the smart card.
SENSE	Input	Input	**SIM Sense—**This signal is a Schmitt trigger input that signals when a smart card is inserted or removed.
SIMDATA	Input or Output	Input	**SIM Data—**This bidirectional signal is used to transmit data to and receive data from the smart card.
$\overline{\text{SIMRESET}}$	Output	Input	**SIM Reset—**The SCP can activate the reset of an inserted smart card by driving $\overline{\text{SIMRESET}}$ low.
$\overline{\text{PWR\_EN}}$	Output	Input	**SIM Power Enable—**This active high signal enables an external device that supplies V_{CC} to the smart card, providing effective power management and power sequencing for the SIM. If the port drives this signal high, the external device supplies power to the smart card. Driving the signal low disables power to the card.

2.12 SAP

The signals described in Table 2-15 are GPIO when not programmed otherwise and default as GPI after reset.

Note: SAP signals STDA, SRDA, SCKA, and SC2A have alternate functions (as described in Table 2-9 on page 2-8 and Table 2-11 on page 2-10). When those alternate functions are selected, the SAP signals are disabled.

Table 2-15. SAP Signals

Signal Name	Type	Reset State	Signal Description
STDA	Output	Input	**Audio Codec Transmit Data**—This output signal transmits serial data from the audio codec serial transmitter shift register.
SRDA	Input	Input	**Audio Codec Receive Data**—This input signal receives serial data and transfers the data to the audio codec receive shift register.
SCKA	Input or Output	Input	**Audio Codec Serial Clock**—This bidirectional signal provides the serial bit rate clock. It is used by both transmitter and receiver in synchronous mode or by the transmitter only in asynchronous mode.
SC0A	Input or Output	Input	**Audio Codec Serial Clock 0—**This signal's function is determined by the transmission mode. • Synchronous mode—serial I/O flag 0 • Asynchronous mode— receive clock I/O

Table 2-15. SAP Signals (Continued)

Signal Name	Type	Reset State	Signal Description
SC1A	Input or Output	Input	**Audio Codec Serial Clock 1—**This signal's function is determined by the transmission mode. • Synchronous mode—serial I/O flag 1 • Asynchronous mode—receiver frame sync I/O
SC2A	Input or Output	Input	**Audio Codec Serial Clock 2—**This signal's function is determined by the transmission mode. • Synchronous mode—transmitter and receiver frame sync I/O • Asynchronous mode—transmitter frame sync I/O

2.13 BBP

The signals described in Table 2-16 are GPIO when not programmed otherwise and default as GPI after reset.

Table 2-16. BBP Signals

Signal Name	Type	Reset State	Signal Description
STDB	Output	Input	**Baseband Codec Transmit Data**—This output signal transmits serial data from the baseband codec serial transmitter shift register.
SRDB	Input	Input	**Baseband Codec Receive Data**—This input signal receives serial data and transfers the data to the baseband codec receive shift register.
SCKB	Input or Output	Input	**Baseband Codec Serial Clock**—This bidirectional signal provides the serial bit rate clock. It is used by both transmitter and receiver in synchronous mode or by the transmitter only in asynchronous mode.
SC0B	Input or Output	Input	**Baseband Codec Serial Clock 0—**This signal's function is determined by the SCLK mode. • Synchronous mode—serial I/O flag 0 • Asynchronous mode—receive clock I/O
SC1B	Input or Output	Input	**Baseband Codec Serial Clock 1—**This signal's function is determined by the SCLK mode. • Synchronous mode—serial I/O flag 0 • Asynchronous mode—receiver frame sync I/O
SC2B	Input or Output	Input	**Baseband Codec Serial Clock 2—**This signal's function is determined by the SCLK mode. • Synchronous mode—transmitter and receiver frame sync I/O • Asynchronous mode—transmitter frame sync I/O

2.14 MCU Emulation Port

The signals described in Table 2-17 are GPIO when not programmed otherwise and default as GPI after reset.

Table 2-17. Emulation Port Signals

Signal Name	Type	Reset State	Signal Description
SIZ0–SIZ1	Output	Input	**Data Size**—These output signals encode the data size for the current MCU access.
PSTAT0–PSTAT3	Output	Input	**Pipeline State**—These output signals encode the internal MCU execution status.

2.15 Debug Port Control

The signals described in Table 2-18 are GPIO when not programmed otherwise and default as GPI after reset.

Table 2-18. Debug Control Signals

Signal Name	Type	Reset State	Signal Description
$\overline{\text{MCU\_DE}}$	Input or Output	Input	**Microcontroller Debug Event**—As an input, this signal provides a means to enter the debug mode of operation from an external command converter. As an output signal, it acknowledges that the MCU has entered the debug mode. When the MCU enters the debug mode due to a debug request or as the result of meeting a breakpoint condition, it asserts $\overline{\text{MCU\_DE}}$ as an output signal for several clock cycles.
$\overline{\text{DSP\_DE}}$	Input or Output	Input	**Digital Signal Processor Debug Event**—This signal functions as $\overline{\text{DSP\_DE}}$. In normal operation, $\overline{\text{DSP\_DE}}$ is an input that provides a means to enter the debug mode of operation from an external command converter. When the DSP enters the debug mode due to a debug request or as the result of meeting a breakpoint condition, it asserts $\overline{\text{DSP\_DE}}$ as an output signal for three clock cycles to acknowledge that it has entered debug mode.

2.16 JTAG Test Access Port

When the bottom connector pins are selected by holding the MUX_CTL pin at a logic high, all JTAG pins become inactive, i.e., disconnected from the JTAG TAP controller.

Table 2-19. JTAG Port Signals

Signal Name	Type	Reset State	Signal Description
TMS	Input	Input	**Test Mode Select**—TMS is an input signal used to sequence the test controller's state machine. TMS is sampled on the rising edge of TCK.
TDI	Input	Input	**Test Data Input**—TDI is an input signal used for test instructions and data. TDI is sampled on the rising edge of TCK.
TDO	Output	Tri-stated	**Test Data Output**—TDO is an output signal used for test instructions and data. TDO can be tri-stated and is actively driven in the shift-IR and shift-DR controller states. TDO changes on the falling edge of TCK.
$\overline{\text{TCK}}$	Input	Input	**Test Clock**—TCK is an input signal used to synchronize the JTAG test logic.
$\overline{\text{TRST}}$	Input	Input	**Test Reset**—$\overline{\text{TRST}}$ is an active-low Schmitt-trigger input signal used to asynchronously initialize the test controller.
TEST	Input	Input	**Factory Test Mode**—Selects factory test mode. Reserved.

Chapter 3
Memory Maps

This section describes the internal memory map of the DSP56652. The memory maps for MCU and DSP are described separately.

3.1 MCU Memory Map

The MCU side of the DSP56652 has a single, contiguous memory space with four separate partitions:

- Internal ROM
- Internal RAM
- Memory-mapped peripherals
- External memory space

These spaces are shown in Figure 3-1 on page 3-2.

3.1.1 ROM

The MCU memory map allocates 1 Mbyte for internal ROM. The actual ROM size is 16 kbytes, starting at address $0000_0000, and is modulo-mapped into the remainder of the 1 Mbyte space. Read access to internal ROM space returns the transfer acknowledge ($\overline{\text{TA}}$) signal except in user mode while supervisor protection is active, in which case a transfer error acknowledge signal ($\overline{\text{TEA}}$) is returned, resulting in termination and an access error exception. Any attempt to write to the MCU ROM space also returns $\overline{\text{TEA}}$. Software should not rely on modulo-mapping because future DSP5665*x* chip implementations may behave differently.

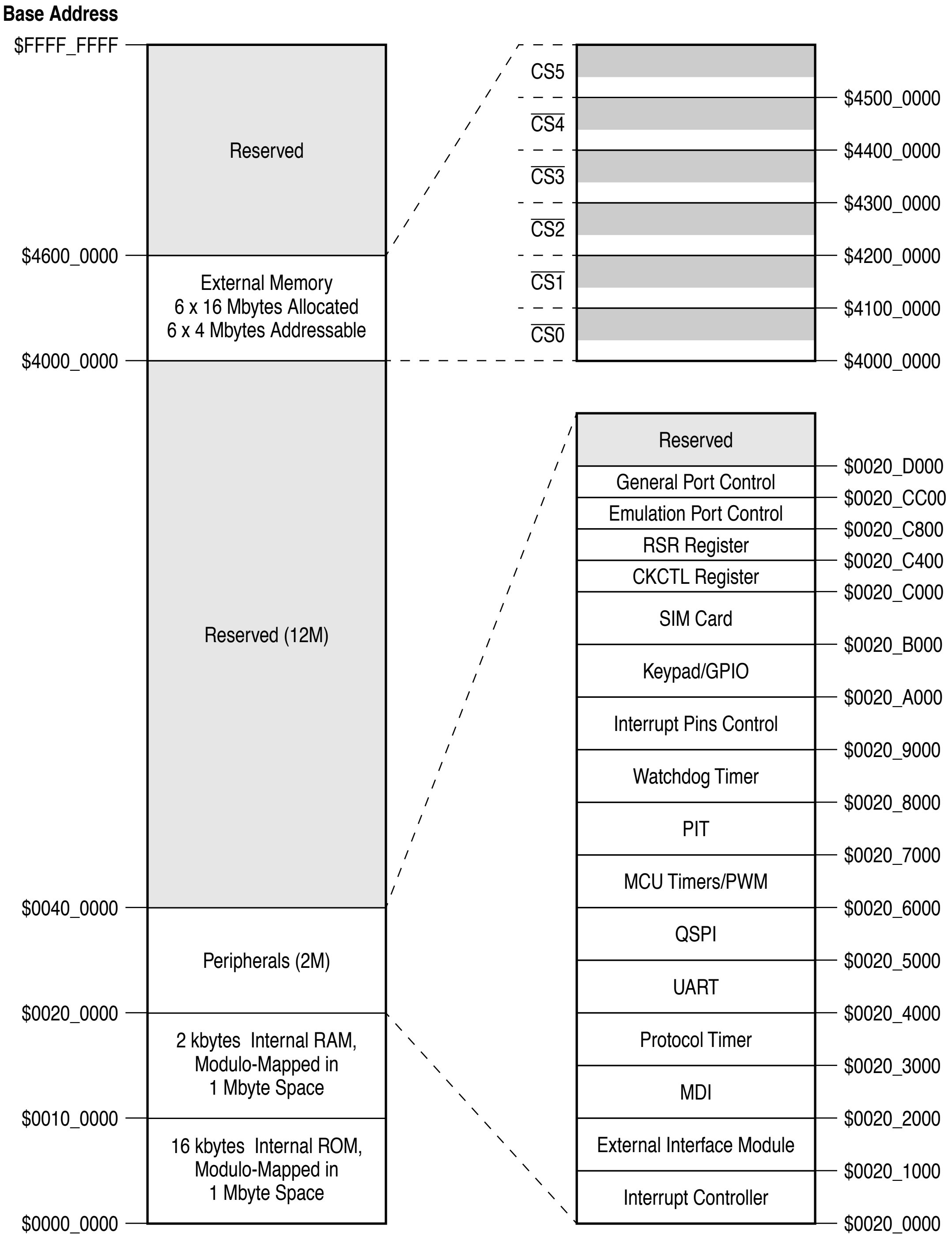


Figure 3-1. MCU Memory Map

3.1.2 RAM

The MCU memory map allocates 1 Mbyte for internal RAM. The actual size of the RAM is 2 KB, starting at address $0010_0000, and is modulo-mapped into the remainder of the 1 Mbyte space. Read and write access to internal RAM space returns $\overline{\text{TA}}$ except in user mode while supervisor protection is active, in which case $\overline{\text{TEA}}$ is returned, resulting in termination and an access error exception. Software should not rely on modulo-mapping because future DSP5665*x* chip implementations may behave differently.

3.1.3 Memory-Mapped Peripherals

Interface requirements for MCU peripherals are defined to simplify the hardware interface implementation while providing a reasonable and extendable software model. The following requirements are currently defined (others may be added in the future):

- A given peripheral device appears only in the 4-kbytes region(s) allocated to it.
- For on-chip devices, registers are defined to be 16 or 32 bits wide. For registers that do not implement all 32 bits, the unimplemented bits return zero when read, and writes to unimplemented bits have no effect. In general, unimplemented bits should be written to zero to ensure future compatibility.
- All peripherals define the exact results for 32-bit, 16-bit, and 8-bit accesses, according to individual peripheral definitions. Misaligned accesses are not supported, nor is bus sizing performed for accesses to registers smaller than the access size.

The MCU memory map allocates 2 Mbyte for internal MCU peripherals starting at address $0020_0000. Twelve of the sixteen DSP56652 peripherals are allocated 4 kbytes each, and four peripherals are allocated 1 kbyte each for a total of 52 kbytes. The remainder of the 2-Mbyte space is reserved for future peripheral expansion.

Each peripheral space may contain several registers. Details of these registers are located in the respective peripheral description sections. Software should explicitly address these registers, making no assumptions regarding modulo-mapping. A complete list of these registers and their addresses is given in Table D-8 on page D-14.

Read accesses to unmapped areas within the first 52 kbytes of peripheral address space returns the $\overline{\text{TA}}$ signal if supervisor permission allows. Uninitialized write accesses within the first 52 kbytes also return the $\overline{\text{TA}}$ signal and may alter the peripheral register contents. Any attempted access within the reserved portion of the peripheral memory space ($0020_D000 to $003F_FFFF) results in $\overline{\text{TEA}}$ termination and an access error exception from an EIM watchdog time-out after 128 MCU clock cycles.

3.1.4 External Memory Space

The MCU memory map allocates 96 Mbytes for external chip access, starting at address $4000_0000. Six external chip selects are allocated 16 Mbytes each. Only the first 4Mbytes in each 16-Mbyte space are addressable by the 22 address lines A0–A21. An access to an address more than 4 Mbytes above the chip select base address is modulo-mapped into the first 4 Mbytes. See Table 6-2 on page 6-4, for more information regarding this portion of the memory map.

3.1.5 Reserved Memory

Two portions of the MCU memory map are reserved: $0040_0000 to $3FFF_FFFF, and $4600_0000 to $FFFF_FFFF. Any attempted access within these reserved portions of the memory space results in $\overline{\text{TEA}}$ termination and an access error exception from an EIM watchdog time-out after MCU 128 clock cycles.

3.2 DSP Memory Map and Descriptions

The DSP56652 DSP core contains three distinct memory spaces:

- X data memory space
- Y data memory space
- program (P) memory space

Each of these spaces contains both RAM and ROM. In addition, the X data space has partitions for peripherals and the MCU-DSP interface (MDI). All memory on the DSP side is contained on-chip—there is no provision for connection to external memory.

The three memory spaces are shown in Figure 3-2.

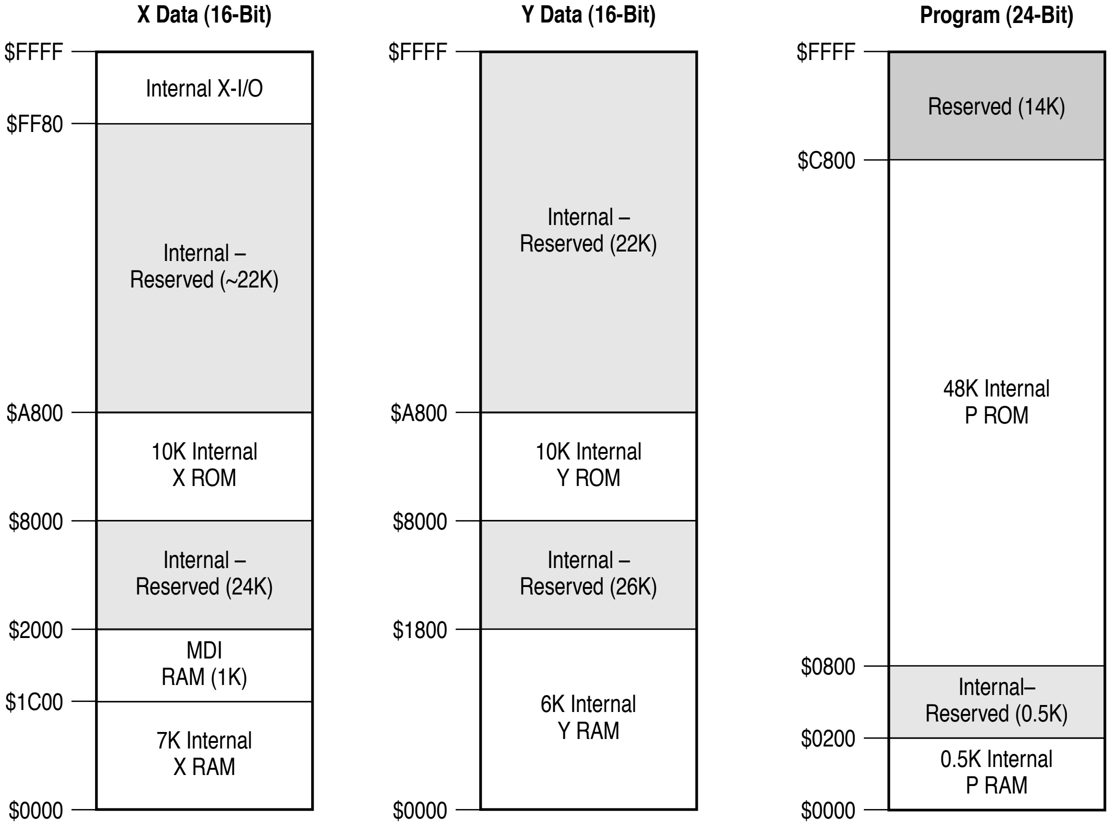


Figure 3-2. DSP Memory Map

3.2.1 X Data Memory

X data RAM is a 16-bit-wide, internal, static memory occupying the lowest 8K locations in X memory space. The upper 1K of this space (X:$1C00–1CFF) is dedicated to the MDI.

X data ROM is a 16-bit-wide, internal, static memory occupying 10K located at X:$8000–$A7FF.

The top 128 locations of the X data memory ($FF80–$FFFF) contain the DSP-side peripheral registers and addressable core registers. This area, referred to as X-I/O space, can be accessed by MOVE, MOVEP, and the bit-oriented instructions (BCHG, BCLR, BSET, BTST, BRCLR, BRSET, BSCLR, BSSET, JCLR, JSET, JSCLR and JSSET). The specific addresses for DSP registers are listed in Table D-9 on page D-19.

3.2.2 Y Data Memory

Y data RAM is a 16-bit-wide, internal, static memory occupying the lowest 6K locations in Y memory space, Y:$0000–$17FF.

Y data ROM is a 16-bit-wide, internal, static memory occupying 10K locations in Y memory space at Y:$8000–$A7FF.

3.2.3 Program Memory

Program RAM is a 24-bit-wide, high-speed, static memory occupying the lowest 512 locations in the P memory space, P:$0000–$01FF.

Program ROM is a 24-bit-wide, internal, static memory occupying 48K locations at P:$0800–$C7FF. The first 1K of this space (P:$0800–$0BFF) contains factory code that enables the user to download code to program RAM via the MDI. This code is described and listed in Appendix A, “DSP56652 DSP Bootloader”.

3.2.4 Reserved Memory

All memory locations not specified in the above description are reserved and should not be accessed. These areas include the following:

- X:$2000–$7FFF and X:$A800–$FF7F
- Y:$1800–$7FFF and Y:$A800–$FFFF
- P:$0200–$07FF and P:$C800–$FFFF.

Chapter 4
Core Operation and Configuration

This section describes features of the DSP56652 not covered by the sections describing individual peripherals. These features include the following:

- Clock configurations for both the MCU and DSP
- Low power operation
- Reset
- DSP features—operating mode, patch addresses, and device identification.
- I/O/ multiplexing

4.1 Clock Generation

Two internal processor clocks, MCU_CLK and DSP_CLK, drive the MCU and DSP cores respectively. Each of these clocks can be derived from either the CKIH or CKIL clock input pins. Both pins should be driven, even if one input is used for both internal clocks.

- CKIH is typically in the frequency range of 10–20 MHz. The DSP56652 converts CKIH to a buffered CMOS square wave which can be brought out externally on the CKOH pin by clearing the CKOHD bit in the Clock Control Register (CKCTL). The buffer can be disabled by setting the CKIHD bit in the CKCTL, but only if MCU_CLK is driven by CKIL.
- CKIL is usually a 32.768 kHz square wave input.

The frequency of each core clock can be adjusted by manipulating control register bits.

At reset, the MCU_CLK is output on the CKO pin. Software can change the output to DSP_CLK by setting the CKOS bit in the CKCTL. The CKO pin can be disabled by setting the CKOD bit in the CKCTL.

The DSP56652 clock scheme is shown in Figure 4-1.

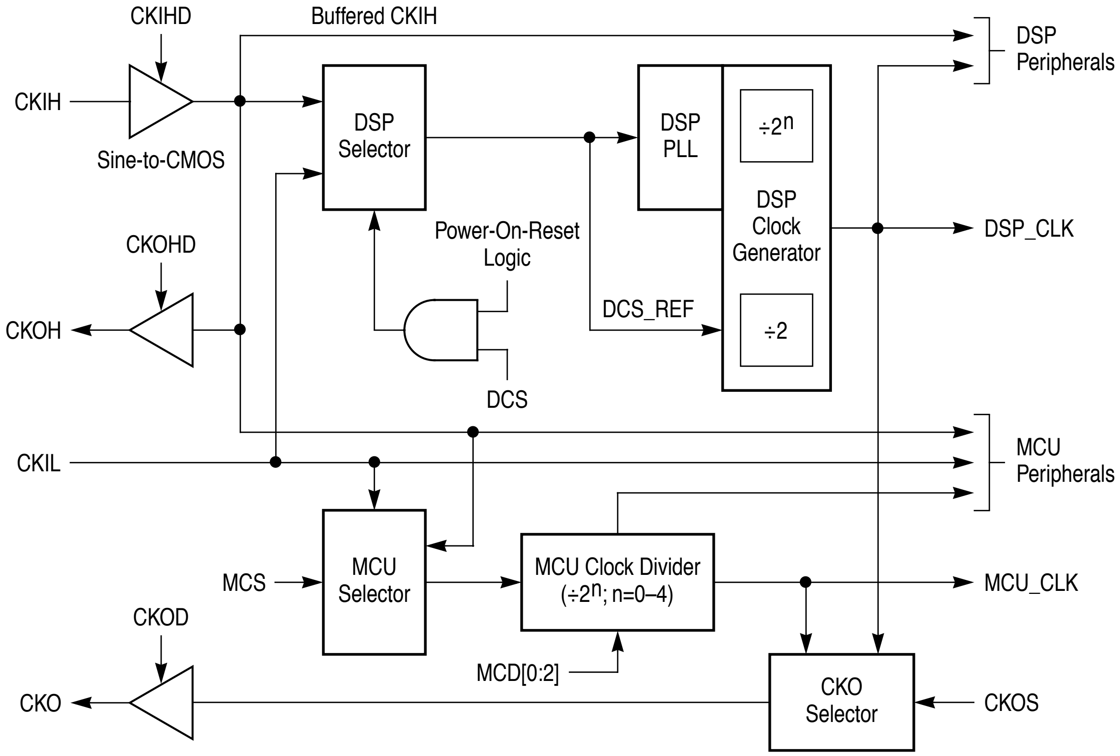


Figure 4-1. DSP56652 Clock Scheme

4.1.1 MCU_CLK

MCU_CLK is driven by either CKIL or (buffered) CKIH, according to the MCS bit in the CKCTL. The input is divided by a power of 2 (i.e., 1, 2, 4, 8 or 16) selected by the MCD bits in the CKCTL. The divider has two outputs, one for the core clock and one for peripherals, to support various low-power modes. MCU peripherals use a combination of CKIL, CKIH, and MCU_CLK, as shown in Table 4-1.

Table 4-1. MCU and MCU Peripherals Clock Source

Peripheral	Peripheral Clock Source
MCU	MCU_CLK
Protocol Timer	MCU_CLK
QSPI	MCU_CLK
UART	CKIH (MCU_CLK for interface to MCU) Serial clock should be slower than MCU_CLK by 1:4 rate.
Interrupt Controller	MCU_CLK
MCU Timers	MCU_CLK
Watchdog Timer	CKIL (MCU_CLK for interface to MCU)
O/S Interrupt (PIT)	CKIL (MCU_CLK for interface to MCU)
GPIO/Keypad	MCU_CLK (CKIL for interrupt debouncer)
SCP	CKIH (MCU_CLK for interface to MCU) Serial clock should be slower than or equal to MCU_CLK.

4.1.2 DSP_CLK

The DSP clock input, DSP_REF, is selected from either CKIL or (buffered) CKIH by the DCS bit in the CKCTL. DSP_REF drives the DSP clock generator either directly or through a PLL, according to the PEN bit in PLL Control Register 1 (PCTL1). The clock generator divides its input by two and puts out the core DSP_CLK signal and a two-phase clock to drive peripherals. DSP peripherals can also use CKIH as an input. Figure 4-2 is a block diagram of the DSP clock system.

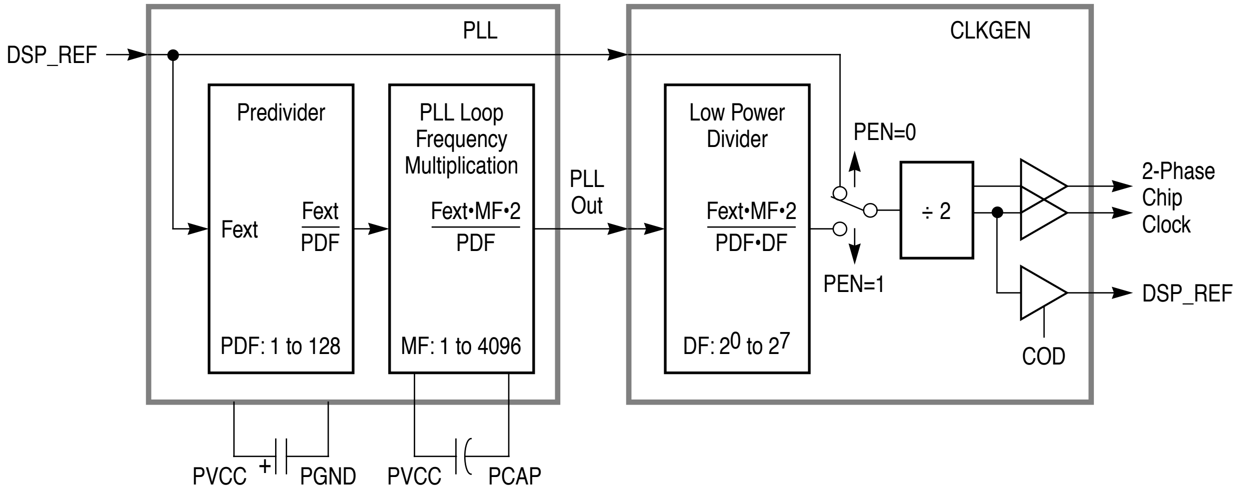


Figure 4-2. DSP PLL and Clock Generator

When the PLL is enabled, its input is divided by a predivide factor (PD bits in PCTL1 and PCTL0) and another divide factor (DF bits in PCTL1) which is intended to decrease the DSP_CLK frequency in low power modes. The DF bits can be adjusted without losing PLL lock. The PLL also multiplies the input by a factor determined by the MF bits in PCTL0. The PLL output frequency is

$$\text{PLLOUT} = \frac{\text{DSP\_REF} \times \text{MF} \times 2}{\text{PD} \times \text{DF}}$$

and the clock generator output frequency is

$$\text{DSP\_CLK} = \frac{\text{DSP\_REF} \times \text{MF}}{\text{PD} \times \text{DF}}$$

The PLL can be bypassed by clearing the PEN bit in PCTL1. It can also be disabled in low power modes by clearing the PSTP bit in PCTL1. In either case, the clock generator output is

$$\text{DSP\_CLK} = \frac{\text{DSP\_REF}}{2}$$

4.1.3 Clock and PLL Registers

CKCTL Clock Control Register **$0020_C000**

	BIT 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT 0
								DCS	CKOHD	CKOD	CKOS	MCD[2:0]			MCS	CKIHD
RESET	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

Table 4-2. CKCTL Description

<table>
<tr><th>Name</th><th>Description</th><th>Settings</th></tr>
<tr><td>DCS
Bit 8</td><td>DSP Clock Select—Selects the input to the DSP clock generator.</td><td>0 = CKIH (default).
1 = CKIL.</td></tr>
<tr><td>CKOHD
Bit 7</td><td>CKOH Disable—Controls the output at the CKOH pin.</td><td>0 = CKOH is a buffered CKIH (default).
1 = CKOH held low.</td></tr>
<tr><td>CKOD
Bit 6</td><td>CKO Disable—Controls the output of the CKO pin.</td><td>0 = CKO outputs either MCU_CLK or DSP_CLK according to CKOS bit (default).
1 = CKO held high.</td></tr>
<tr><td>CKOS
Bit 5</td><td>CKO Source Select—Selects the clock to be reflected on the CKO pin.</td><td>0 = MCU_CLK (default).
1 = DSP_CLK.</td></tr>
<tr><td>MCD[2:0]
Bits 4–2</td><td>MCU Clock Divide factor—Selects the divisor for the MCU clock.</td><td><table>
<tr><th>MCD[2:0]</th><th>Divisor</th></tr>
<tr><td>$0</td><td>1</td></tr>
<tr><td>$1</td><td>2</td></tr>
<tr><td>$2</td><td>4</td></tr>
<tr><td>$3</td><td>8</td></tr>
<tr><td>$4</td><td>16</td></tr>
<tr><td>$5...$7</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>MCS
Bit 1</td><td>MCU Clock Select—Determines MCU clock input</td><td>0 = CKIL (default).
1 = CKIH.</td></tr>
<tr><td>CKIHD
Bit 0</td><td>CKIH Disable—Controls the CKIH input buffer</td><td>0 = Buffer enabled (default).
1 = Buffer disabled if MCS is cleared.</td></tr>
</table>

PCTL0 PLL Control Register 0 **X:$FFFD**

	BIT 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT 0
	PD[3:0]				MF[11:0]											
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 4-3. PCTL0 Descriptions

<table>
<tr><th>Name</th><th>Description</th><th>Settings</th></tr>
<tr><td>PD[3:0]
Bits 15–12</td><td>Predivider Factor Bits—Concatenated with PD[6:4] (PCTL1 bits 11–9) to define the PLL input PDF.</td><td>See Table 4-4 on page 4-7.</td></tr>
<tr><td>MF[11:0]
Bits 11–0</td><td>Multiplication Factor Bits—Define the MF applied to the PLL input frequency.</td><td><table>
<tr><th>MF[11:0]</th><th>MF</th></tr>
<tr><td>$000</td><td>1 (default)</td></tr>
<tr><td>$001</td><td>2</td></tr>
<tr><td>$002</td><td>3</td></tr>
<tr><td>.
.
.</td><td>.
.
.</td></tr>
<tr><td>$FFE</td><td>4095</td></tr>
<tr><td>$FFF</td><td>4096</td></tr>
</table></td></tr>
</table>

PCTL1 PLL Control Register 1 **X:$FFFC**

	BIT 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT 0
					PD[6:4]				COD	PEN	PSTP	XTLD	XTLR	DF[2:0]		
RESET	0	0	0	0	0	0	0		0	0	0	1	0	0	0	0

Table 4-4. PCTL1 Description

<table>
<tr><th>Name</th><th>Description</th><th>Settings</th></tr>
<tr><td>PD[6:4]
Bits 11–9</td><td>Predivider Factor—Concatenated with PD[3:0] from PCTL0 to define the PLL input frequency divisor. The divisor is equal to one plus the value of PD[6:0].</td><td><table><tr><th>PD[6:0]</th><th>PLL Divisor</th></tr><tr><td>$00</td><td>1 (default)</td></tr><tr><td>$01</td><td>2</td></tr><tr><td colspan="2">- - -</td></tr><tr><td>$7F</td><td>128</td></tr></table></td></tr>
<tr><td>COD
Bit 7</td><td colspan="2">Clock Output Disable—This bit disconnects DSP_CLK from the CKO pin in some implementations. In the DSP56652 this bit has no effect.</td></tr>
<tr><td>PEN
Bit 6</td><td>PLL Enable—Enables PLL operation. Disabling the PLL shuts down the VCO and lowers power consumption. The PEN bit can be set or cleared by software any time during the chip operation.</td><td>0 = PLL is disabled (default). DSP_CLK is derived directly from DSP_REF.
1 = PLL is enabled. DSP_CLK is derived from the PLL VCO output.</td></tr>
<tr><td>PSTP
Bit 5</td><td>STOP Processing State—Controls the behavior of the PLL during the STOP processing state. Shutting down the PLL in STOP mode decreases power consumption but increases recovery time.</td><td>0 = Disable PLL in STOP mode (default).
1 = Enable PLL in STOP mode.</td></tr>
<tr><td>XTLD,
XTLR
Bits 4–3</td><td colspan="2">These bits affect the on-chip crystal oscillator in certain implementations. They are not used in the DSP56652.</td></tr>
<tr><td>DF[2:0]
Bits 2–0</td><td>Division Factor—Internal clock divisor that determines the frequency of the low-power clock. Changing the value of the DF bits does not cause a loss of lock condition. These bits should be changed rather than MF[11:0] to change the clock frequency (e.g., when entering a low-power mode to conserve power).</td><td><table><tr><th>DF[2:0]</th><th>DF</th></tr><tr><td>000</td><td>2^0 (default)</td></tr><tr><td>001</td><td>2^1</td></tr><tr><td colspan="2">- - -</td></tr><tr><td>111</td><td>2^7</td></tr></table></td></tr>
</table>

4.2 Low Power Modes

The DSP56652 features several modes of operation to conserve power under various conditions. Each core can run independently in either the normal, WAIT, or STOP mode. The MCU can also run in the DOZE mode, which operates at an activity level between WAIT and STOP. Each low-power mode is initiated by a software instruction, and terminated by an interrupt. The wake-up interrupt can come from any running peripheral. In STOP mode, certain stopped peripherals can also generate a wake-up interrupt. Peripheral operation in low power modes for the MCU and DSP is summarized in Table 4-5 and Table 4-6, respectively.

Table 4-5. MCU Peripherals in Low Power Mode

Peripheral	Normal	WAIT	DOZE	STOP
MCU	Running	Stopped	Stopped	Stopped
Protocol Timer	Running	Running	Programmable	Stopped
QSPI	Running	Running	Programmable	Stopped
UART	Running	Running	Programmable	Stopped; can trigger wake-up
Interrupt Controller	Running	Running	Running	Stopped; can trigger wake-up
MCU Timers	Running	Running	Programmable	Stopped
Watchdog Timer	Running	Running	Programmable	Stopped
PIT (O/S interrupt)	Running	Running	Running	Running
GPIO/Keypad	Running	Running	Running	Stopped; can trigger wake-up
MDI (MCU side)	Running	Running	Programmable	Stopped; can trigger wake-up
SCP	Running	Running	Programmable	Stopped
JTAG/OnCE	Running	Running	Programmable	Stopped; can trigger wake-up
External interrupt	Running	Running	Running	Stopped; can trigger wake-up

Table 4-6. DSP Peripherals in Low Power Modes

Peripheral	Normal	WAIT	STOP
MDI (DSP side)	Running	Running	Stopped; can trigger wake-up
BBP	Running	Running	Stopped
SAP	Running	Running	Stopped

For further power conservation, any running peripheral in a given mode, as well as the features summarized in Table 4-7, can be explicitly disabled by software.

Table 4-7. Programmable Power-Saving Features

Description	Register	Reference
Disable CKOH	CKCTL bit 7	Table 4-2
Disable CKO	bit 6	
Disable CKIH buffer	bit 0	
Disable DSP_CLK	PCTL1 bit 7	Table 4-4
Disable PLL	bit 6	
Disable PLL in STOP mode	bit 5	

4.3 Reset

Four events can cause a DSP56652 reset:

1. Power-on reset
2. $\overline{\text{RESET\_IN}}$ pin is asserted
3. Bottom connector $\overline{\text{RTS}}$ pin (acting as $\overline{\text{RESET\_IN}}$) is asserted
4. Watchdog timer times out

Reset from power-on or the watchdog timer time-out is immediately qualified. An input circuit qualifies the $\overline{\text{RESET\_IN}}$ signal from either pin, based on the duration of the signal in CKIL clock cycles:

- 2 cycles—not qualified
- 3 cycles—may or may not be qualified
- 4 cycles—qualified

A qualified reset signal asserts the $\overline{\text{RESET\_OUT}}$ signal, and the following reset conditions are established:

- All peripherals and both cores are initialized to their default values.
- Both MCU_CLK and DSP_ CLK are derived from CKIL.
- The CKO pin is enabled, driving MCU_CLK.
- The CKIH CMOS converter is enabled, and drives the CKOH pin.

An eight-cycle “stretch’ circuit guarantees that $\overline{\text{RESET\_OUT}}$ is asserted for at least eight CKIL clock cycles. This circuit also stretches the negation of $\overline{\text{RESET\_OUT}}$. (The precise time between the negation of RESET_IN and $\overline{\text{RESET\_OUT}}$ is between seven and eight CKIL cycles.) Four cycles before $\overline{\text{RESET\_OUT}}$ is negated, the MOD pin is latched. This externally-driven pin determines whether the first instruction is fetched from internal MCU ROM or external flash memory connected to $\overline{\text{CS0}}$, as described in Section 4.3.1 on page 4-11.

Reset timing is illustrated in Figure 4-3.

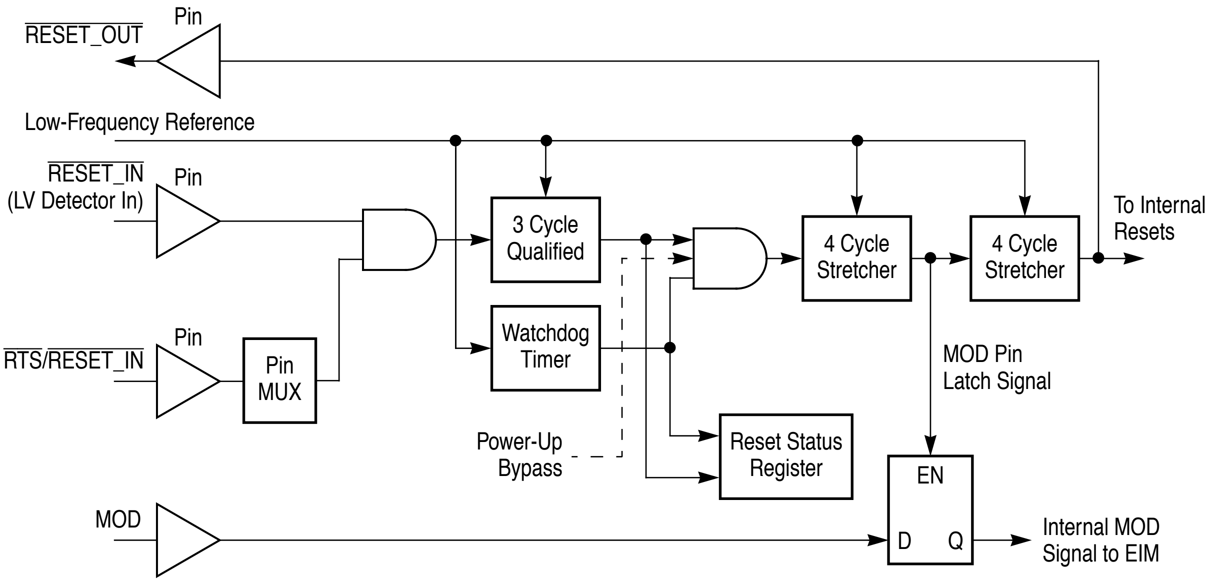


Figure 4-3. DSP56652 Reset Circuit

The DSP56652 provides a read-only Reset Source Register (RSR) to determine the cause of the last hardware reset.

RSR Reset Source Register **$0020_C400**

	BIT 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT 0
															WDR	EXR
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 4-8. RSR Description

Name	Description	Settings
WDR Bit 1	**Watchdog Reset**—Watchdog timer time-out	0 = Last reset not caused by watchdog timer. 1 = Last reset caused by watchdog timer.
EXR Bit 0	**External Reset**—$\overline{\text{RESET\_IN}}$ pin assertion	0 = Last reset not caused by $\overline{\text{RESET\_IN}}$. 1 = Last reset caused by $\overline{\text{RESET\_IN}}$.

If both external and watchdog reset conditions occur simultaneously, the external reset has precedence, and only the EXR bit is set. If a power-on reset occurs with no external reset or watchdog reset, both bits remain cleared.

4.3.1 MCU Reset

All MCU peripherals and the MCU core are configured with their default values when $\overline{\text{RESET\_OUT}}$ is asserted.

Note: The STO bit in the General-Purpose Configuration Register (GPCR), which is reflected on the STO pin, is not affected by reset. It is uninitialized by a power-on reset and retains its current value after $\overline{\text{RESET\_OUT}}$ is asserted.

The MOD input pin specifies the location of the reset boot ROM device. The pin must be driven at least four CKIL cycles before $\overline{\text{RESET\_OUT}}$ is deasserted. If MOD is driven low, the internal MCU ROM is disabled and $\overline{\text{CS0}}$ is asserted for the first MCU cycle. The MCU fetches the reset vector from address $0 of the $\overline{\text{CS0}}$ memory space, which is located at the absolute address $4000_0000 in the MCU address space. The internal MCU ROM is disabled for the first MCU cycle only and is available for subsequent accesses. Out of reset, $\overline{\text{CS0}}$ is configured for 15 wait states and a 16-bit port size. Refer to Table 6-6 on page 6-9 for a more detailed description of $\overline{\text{CS0}}$. If MOD is driven high, the internal ROM is enabled and the MCU fetches the reset vector from internal ROM at address $0000_0000.

4.3.2 DSP Reset

Any qualified MCU reset also resets the DSP core and its peripherals to their default values. In addition, the MCU can issue a hardware or software reset to the DSP through the MCU-DSP Interface (MDI). A hardware reset is generated by setting the DHR bit in the MCR. A software reset can be generated by setting the MC bit in the MCVR to issue a DSP interrupt. In this case, the interrupt service routine might include the following tasks:

- Issue a RESET instruction.
- Reset other core registers that are not affected by the RESET instruction such as the SR and the stack pointer.
- Jump to the initial address of the DSP reset routine, P:$0800.

Once the DSP exits the reset state, it executes the bootloader program described in Appendix A, "DSP56652 DSP Bootloader".

Out of reset, CKIL drives the DSP clock until $\overline{\text{RESET\_OUT}}$ is negated, when the clock source is switched to DSP_REF. To ensure a stable clock, the DSP is held in the reset state for 16 DSP_REF clocks after $\overline{\text{RESET\_OUT}}$ is negated. The PLL is disabled and the default source for DSP_REF is CKIH, so the DSP_CLK frequency is equal to CKIH ÷ 2.

It is recommended that clock sources be present on both the CKIH and CKIL pins. However, should CKIH be inactive at reset, the DSP remains in reset until the MCU sets the DCS bit in the CKCTL register, selecting CKIL as the DSP clock source. In this case, the following MCU sequence is recommended:

1. Set the DHR bit.
2. Set the DCS bit
3. After a minimum of 18 CKIL cycles, clear the DHR bit.

4.4 DSP Configuration

The DSP contains an Operating Mode Register (OMR) to configure many of its features. Four Patch Address Registers (PARs) allow the user to insert code corrections to ROM. A Device Identification Register (IDR) is also provided.

4.4.1 Operating Mode Register

The OMR is a 16-bit read/write DSP core register that controls the operating mode of the DSP56652 and provides status flags on its operation. The OMR is affected only by processor reset, by instructions that directly reference it (for example, ANDI and ORI), and by instructions that specify the OMR as a destination, such as the MOVEC instruction.

OMR Operating Mode Register

	BIT 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT 0
	ATE			SEN	WR	EOV	EUN	XYS		SD	PCD	EBD			MB	MA
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	—	1

Table 4-9. OMR Description

Name	Description	Settings
ATE Bit 15	**Address Trace Enable—**Used in debugging for internal activity that can be traced via a logic analyzer.	0 = Disabled (default)—normal operation. 1 = Enabled. External bus reflects DSP internal program address bus.
SEN Bit 12	**Stack Extension Enable**	0 = Disabled (default). 1 = Enabled.
WR Bit 11	**Extended Stack Wrap Flag—**The DSP sets this bit when it recognizes that the stack extension memory requires a copy of the on-chip hardware stack. This flag is useful in debugging to determine if the speed of software-implemented algorithms must be increased. Once this bit is set it can only be cleared by reset or a MOVE operation to the OMR.	0 = No copy required (default). 1 = Copy of on-chip hardware stack to stack extension memory is required.
EOV Bit 10	**Extended Stack Overflow Flag—**This flag is set when a stack overflow occurs in the Stack Extended mode. The Extended Stack Overflow is generated when SP equals SZ and an additional push operation is requested while the Extended mode is enabled by the SEN bit. The EOV bit is a “sticky bit” (i.e., can be cleared only by hardware reset or an explicit MOVE operation to the OMR). The transition of EOV from 0 to 1 causes an IPL 3 Stack Error interrupt.	0 = No overflow has occurred (default). 1 = Stack overflow in stack extended mode.
EUN Bit 9	**Extended Stack Underflow Flag—**Set when a stack underflow occurs in the Stack Extended mode. The Extended Stack Underflow is generated when the SP equals 0 and an additional pull operation is requested while the Extended mode is enabled by the SEN bit. The EUN bit is a “sticky bit” (i.e., can be cleared only by hardware reset or an explicit MOVE operation to the OMR). The transition of EUN from 0 to 1 causes an Interrupt Priority Level (IPL) Level 3 Stack Error interrupt.	0 = No underflow has occurred (default). 1 = Stack underflow in stack extended mode.
XY Bit 8	**XY Select for Stack Extension—**Determines memory space for stack extension	0 = X memory space (default). 1 = Y memory space.
SD Bit 6	**Stop Delay—**Controls the amount of delay after wake-up from STOP mode. A long delay may be necessary to allow the internal clock to stabilize. **Note:** The SD bit is overridden if the PSTP bit in PCTL1 is set, forcing wake-up with no delay.	0 = Long delay—128K DSP_CLK cycles (default). 1 = Short delay—16 DSP_CLK cycles.

Table 4-9. OMR Description

Name	Description	Settings
PCD Bit 5	**PC Relative Logic Disable—**Used to reduce power consumption when PC-relative instructions (branches and DO loops) are not used. A PC-relative instruction issued while the PC bit is set causes undetermined results. If this bit is set and then cleared, software should wait for the instruction pipeline to clear (at least seven instruction cycles) before issuing the next instruction.	0 = PC-relative instructions can be used (default). 1 = PC-relative instructions disabled.
EBD Bit 4	**External Bus Disable—**Setting this bit disables the core external bus drivers, and is recommended for normal operation to reduce power consumption. EBD must be cleared to use Address Tracing.	0 = External bus circuitry enabled (default). 1 = External bus circuitry disabled.
MB Bit 1	**Operating Mode B—**Used to determine the operating mode in certain devices. On the DSP56652, this bit reflects the state of the $\overline{\text{DSP\_IRQ}}$ pin at the negation of $\overline{\text{RESET\_IN}}$.	
MA Bit 0	**Operating Mode A—**Used to determine the operating mode in certain devices. On the DSP56652, this bit is set after reset.	

4.4.2 Patch Address Registers

Program patch logic block provides a way to amend program code in the on-chip DSP ROM without generating a new mask. Implementing the code correction is done by replacing a piece of ROM-based code with a patch program stored in RAM.

There are four patch address registers (PAR0–PAR3) at DSP I/O addresses X:$FFF8–FFF5. Each PAR has an associated address comparator. When an address of a fetched instruction is identical to the address stored in a PAR, that instruction is replaced by a JMP instruction to the PAR’s jump target address, where the patch code resides. The patch registers, register addresses and jump targets are listed in Table 4-10.

Table 4-10. Patch JUMP Targets

Patch Register	Register Address	JUMP Target
PAR0	X:$FFF8	$0018
PAR1	X:$FFF7	$0078
PAR2	X:$FFF6	$0098
PAR3	X:$FFF5	$00F8

For more information, refer to the *DSP56600 Family Manual* (DSP56600FM/AD).

4.4.3 Device Identification Register

The IDR is a 16-bit read-only factory-programmed register used to identify the different DSP56600 core-based family members. This information may be used in testing or by software.

IDR Device Identification Register **X:$FFF9**

	BIT 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT 0
	Revision number				Derivative number = $652											
RESET	0	0	0	0	0	1	1	0	0	1	0	1	0	0	1	0

4.5 I/O Multiplexing

To accommodate all of the functions of the DSP56652 in a 196-pin package, 28 of the pins multiplex two or more functions. Eleven of these pins multiplex various peripherals, primarily with the JTAG Debug Port. The other 17 pins multiplex peripherals with the DSP Address Trace function.

4.5.1 Debug Port and Timer Multiplexing

The eight pins listed in Table 4-11 multiplex various peripherals with the Debug Port. The pins in Table 4-12 also multiplex different peripherals, but are not part of the Debug Port. The functions of these pins are determined by the following controls:

1. Asserting the MUX_CTL pin configures all of the pins in Table 4-11 as debugging signals, effectively creating an alternate set of pins for $\overline{\text{RESET\_IN}}$, $\overline{\text{MCU\_DE}}$, $\overline{\text{DSP\_DE}}$, and the five JTAG signals. These eight pins can be brought out externally to facilitate debugging. Asserting MUX_CTL overrides all other controls for these pins. MUX_CTL does not affect the pins in Table 4-12.
2. Each of eight bits in the GPCR selects the peripheral to which the associated pin is connected. Five GPCR bits control pins in Table 4-11 (if MUX_CTL is not asserted); the other three bits control the pins in Table 4-12.
3. Once a pin is assigned to a peripheral (MUX_CTL = 0 and/or the associated GPCR bit is written), that peripheral's Port Configuration Register determines if the pin is configured for the peripheral function or GPIO.

Table 4-11. Debug Port Pin Multiplexing

Pin No.	GPCR Bit No.	MUX_CTL =1	MUX_CTL =0				
			GPCR Bit = 1		GPCR = 0		
			Module	Pin	Module	Pin	UART
K11	0	$\overline{\text{TRST}}$	SAP	STDA	Interrupt Controller	INT6	$\overline{\text{DSR}}$[1]
J12	1	TMS	SAP	SRDA	Interrupt Controller	INT7	$\overline{\text{DTR}}$ or SCLK[2]
G13	5	$\overline{\text{DSP\_DE}}$	SAP	SC2A	KP	ROW6	$\overline{\text{DCD}}$[3]
G11	6	TCK	SAP	SCKA	KP	ROW7	$\overline{\text{RI}}$[4]
E11	7	$\overline{\text{RESET\_IN}}$	MCU Timer	IC2A	$\overline{\text{UART}}$	$\overline{\text{RTS}}$	—
D14	—	$\overline{\text{MCU\_DE}}$	UART—$\overline{\text{CTS}}$				
E14	—	TDO	UART—TxD				
E12	—	TDI	UART—RxD / MCU Timer—ICI[5]				

1. The $\overline{\text{DSR}}$ function for K11 is enabled by setting GPCR bit 0 AND using the pin as GPIO in the Edge Port.
2. The $\overline{\text{DTR}}$ function for J12 is enabled by setting GPCR bit 1 AND using the pin as an Edge Port interrupt. The SCLK function for J12 is enabled by setting GPCR bit 1 AND setting the CLKSRC bit in UCR2.
3. The $\overline{\text{DCD}}$ function for G13 is enabled by clearing GPCR bit 5 AND using the pin as GPIO in the Keypad Port.
4. The $\overline{\text{RI}}$ function for G11 is enabled by clearing GPCR bit 6 AND using the pin as GPIO in the Keypad Port.
5. When MUX_CTL = 0, the E12 pin is connected to both the UART RxD input and the Timer IC1 input.

Table 4-12. Timer Pin Multiplexing

Pin No.	GPCR Bit #	GPCR Bit = 1		GPCR = 0	
		Port	Pin	Port	Pin
N13	2	MCU Timer	OC1	KP	COL6
M13	3	MCU Timer	PWM	KP	COL7
H14	4	MCU Timer	IC2B	KP	ROW5

Figure 4-4 shows the relationship between these 11 pins and their controls.

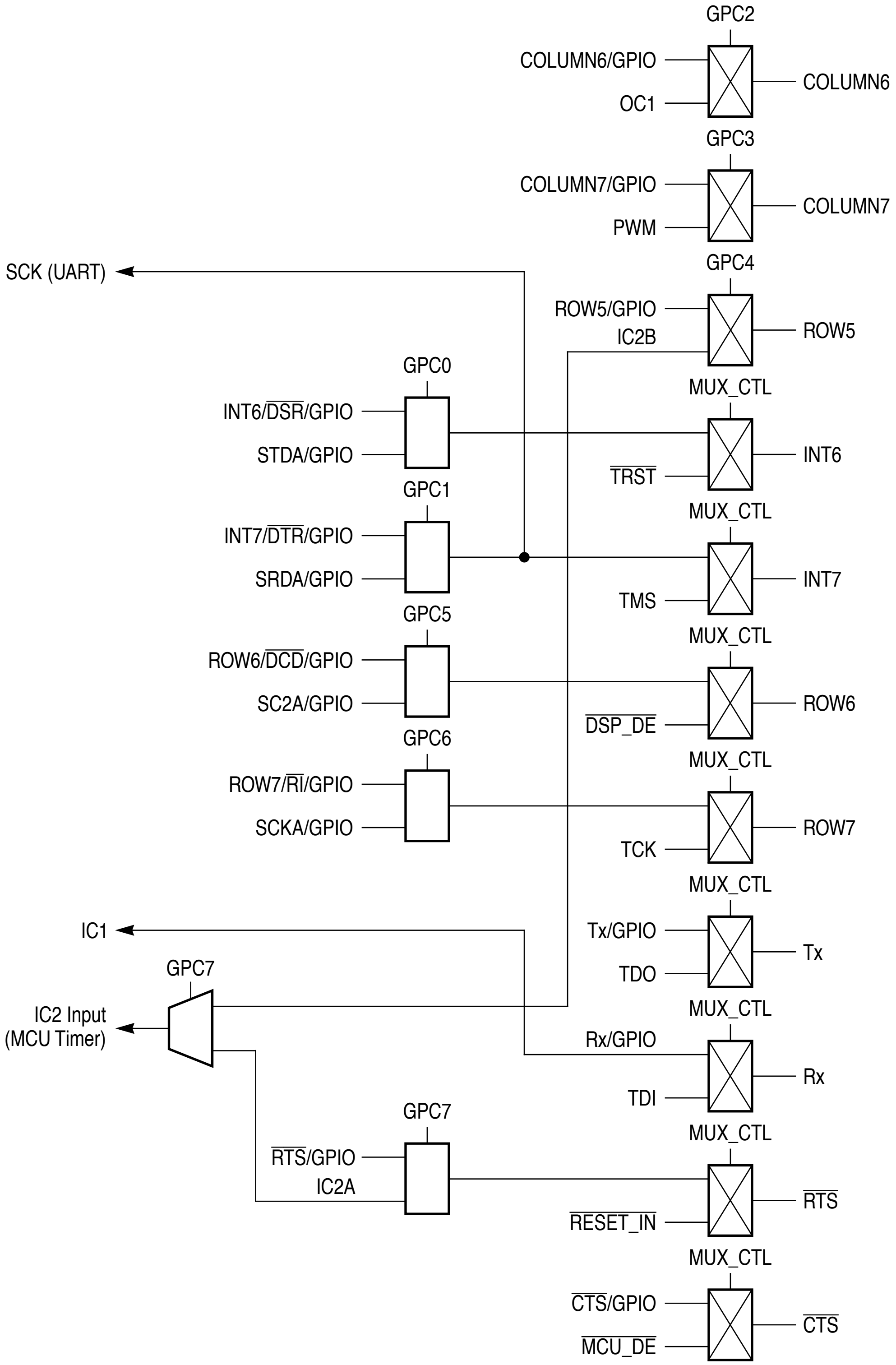


Figure 4-4. MUX Connectivity Scheme

GPCR General Port Control Register **$0020_CC00**

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
	STO								GPC7	GPC6	GPC5	GPC4	GPC3	GPC2	GPC1	GPC0
RESET	—	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 4-13. GPCR Description

Name	Description	Settings
STO Bit 15	**Soft Turn Off—**The value written to this bit is reflected on the STO pin. This bit is not affected by reset or the state of the MUX_CTL pin.	
GPC7 Bit 7	**General Port Control for E11—**determines if pin E11 functions as the UART $\overline{\text{RTS}}$ signal or the Timer IC2 signal. Setting the MUX_CTL pin configures pin E11 as an alternate $\overline{\text{RESET\_IN}}$ signal.	0 = $\overline{\text{RTS}}$ (default). 1 = IC2.
GPC6 Bit 6	**General Port Control for G11—**determines if pin G11 functions as the Keypad ROW7 signal or the SAP SCKA signal. The UART $\overline{\text{RI}}$ signal can be implemented on pin G11 by using ROW7 as a general-purpose output. Setting the MUX_CTL pin configures pin G11 as an alternate JTAG TCK signal.	0 = ROW7 / $\overline{\text{RI}}$. 1 = SCKA.
GPC5 Bit 5	**General Port Control for G13—**determines if pin G13 functions as the Keypad ROW6 signal or the SAP SC2A signal. The UART $\overline{\text{DCD}}$ signal can be implemented on pin G13 by clearing GPC5 and using ROW6 as a general-purpose output. Setting the MUX_CTL pin configures pin G13 as an alternate $\overline{\text{DSP\_DE}}$ signal.	0 = ROW6 / $\overline{\text{DCD}}$. 1 = SC2A.
GPC4 Bit	**General Port Control for H14—**determines if pin H14 functions as the Keypad ROW5 signal or the Timer IC2 signal. This pin is not affected by MUX_CTL.	0 = ROW5 (default). 1 = IC2.
GPC3 Bit 3	**General Port Control for M13—**determines if pin M13 functions as the Keypad COL7 signal or the Timer PWM signal. This pin is not affected by MUX_CTL.	0 = COL7 (default). 1 = PWM.
GPC2 Bit 2	**General Port Control for N13—**determines if pin M13 functions as the Keypad COL6 signal or the Timer OC1 signal. This pin is not affected by MUX_CTL.	0 = COL6 (default). 1 = OC1.

Table 4-13. GPCR Description (Continued)

Name	Description	Settings
GPC1 Bit 1	**General Port Control for J12—**determines if pin J12 functions as the EP INT7 signal or the SAP SRDA signal. Either of two UART signals can be implemented on pin J12 if GPC1 is cleared. The $\overline{\text{DTR}}$ signal requires programming the pin as an interrupt in the edge port. The SCLK signal requires disabling the edge port interrupt and enabling SCLK in UCR2. Setting the MUX_CTL pin configures pin J12 as an alternate JTAG TMS signal.	0 = INT7 / $\overline{\text{DTR}}$ / SCLK 1 = SRDA.
GPC0 Bit 0	**General Port Control for K11—**determines if pin K11 functions as the EP INT6 signal or the SAP STDA signal. The UART $\overline{\text{DSR}}$ signal can be implemented on pin K11 by clearing GPC0, clearing bit 11 in the NIER and FIER to disable the interrupt, and configuring the pin as GPIO. Setting the MUX_CTL pin configures pin K11 as an alternate JTAG $\overline{\text{TRST}}$ signal.	0 = INT6 (default). 1 = STDA.

4.5.2 DSP Address Visibility

DSP internal activity can be accessed for debugging by enabling the DSP Address Visibility Mode. In this mode, the 16 DSP program address lines and an address strobe signal are brought out on the pins listed in Table 4-14.

Table 4-14. Pin Function in DSP Address Visibility Mode

Pin No.	Peripheral Port	Primary Signal	Function In "DSP Trace Address Mode"
P10	—	MOD	DSP_AT (DSP Address Tracing Strobe)
N11	Borrowed from Keypad Port	COLUMN0	DSP_ADDR0
M11		COLUMN1	DSP_ADDR1
P12		COLUMN2	DSP_ADDR2
N12		COLUMN3	DSP_ADDR3
P13		COLUMN4	DSP_ADDR4
M12		COLUMN5	DSP_ADDR5
K14		ROW0	DSP_ADDR6
J13		ROW1	DSP_ADDR7
J11		ROW2	DSP_ADDR8
J14		ROW3	DSP_ADDR9
H13		ROW4	DSP_ADDR10
L7	Borrowed from SmartCard Port	SIMCLK	DSP_ADDR11
P8		SENSE	DSP_ADDR12
M9		SIMDATA	DSP_ADDR13
N9		$\overline{\text{SIMRESET}}$	DSP_ADDR14
K7		PWR_EN	DSP_ADDR15

The Address Visibility Multiplexing is enabled by writing $4 to the OnCE Test and Logic Control Register (OTLCR). The OTCLR is accessed by writing 10011 to the RS[4:0] field in the OnCE Command Register (OCR). For more information on OnCE operation, refer to the *DSP56600 Family Manual*.

Chapter 5
MCU–DSP Interface

The MDI provides a mechanism for transferring data and control functions between the two cores on the DSP56652. The MDI consists of two independent sub-blocks: a shared memory space with read/write access for both processors and a status and message control unit. The primary features of the MDI include the following:

- 1024 × 16-bit shared memory in DSP X data memory space
- interrupt- or poll-driven message control
- flexible, software-controlled message protocols
- MCU can trigger any DSP interrupt (regular or non-maskable) by writing to the command vector control register.
- Each core can wake the other from low-power modes.

The basic block diagram of the MDI module is shown in Figure 5-1.

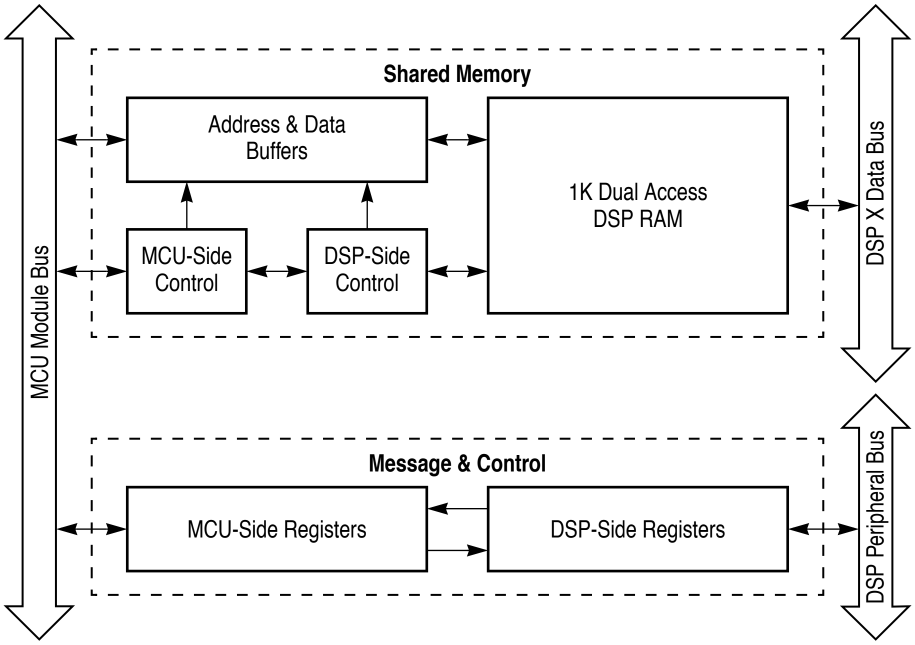


Figure 5-1. MDI Block Diagram

5.1 MDI Memory

The DSP56652 provides special memory areas for the MDI on both the MCU and DSP sides. This section describes where these areas are mapped, how access contention between the two areas is resolved, and memory access timing.

Note: There is no mechanism in MDI hardware to prevent either core from overwriting an area of shared memory written by the other core. It is the responsibility of software to ensure data integrity in shared memory for each core.

5.1.1 DSP-Side Memory Mapping

MDI shared RAM is mapped to the X data memory space of the DSP at the top of its internal X data RAM. From the functional point of view of the DSP, the shared memory is indistinguishable from regular X data RAM. A parallel data path allows the MCU to write to shared memory without restricting or stalling DSP accesses in any way. In case of simultaneous access from both the MCU and the DSP to the same memory space, the DSP access has precedence. The DSP programmer must be aware, however, that data written to that area can be changed by the MCU.

The MDI message control and status registers are mapped to DSP X I/O memory as a regular peripheral, accessible via special I/O instructions.

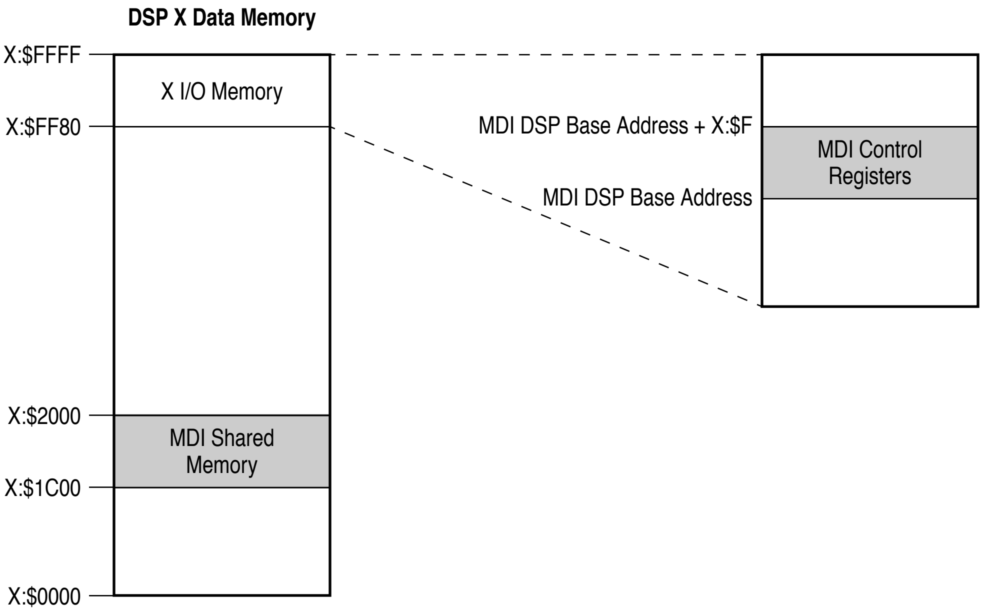


Figure 5-2. MDI: DSP-Side Memory Mapping

5.1.2 MCU-Side Memory Mapping

The MCU allocates a 4-kbyte peripheral space to the MDI, as shown in Figure 5-3. Control and status registers are mapped to the upper 16 words of this space, and shared memory is mapped to the lower 2 kbytes.

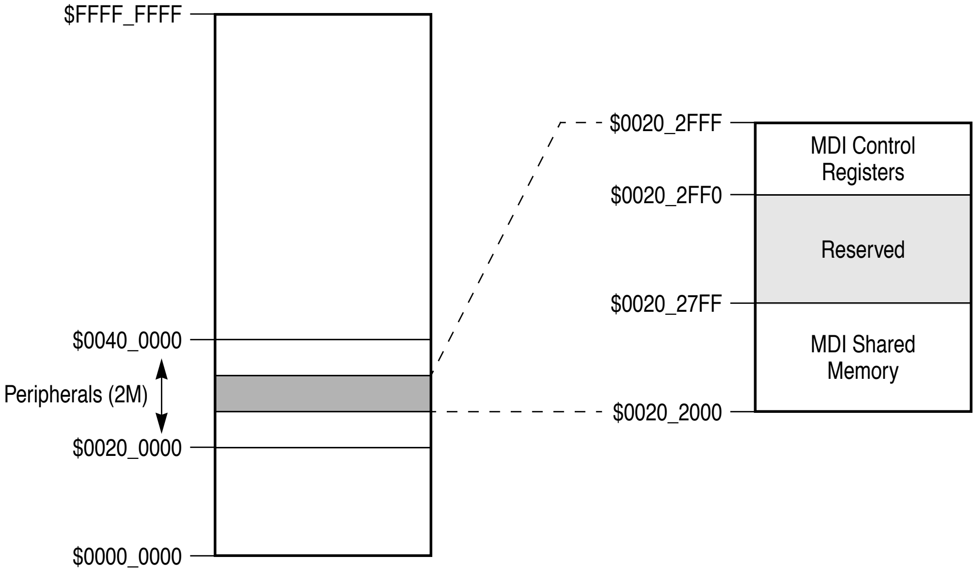


Figure 5-3. MDI: MCU-Side Memory Mapping

Note: Writes to reserved locations are ignored. Reads from reserved locations latch indeterminate data. Neither access terminates in an access error.

The offset conversion formula between the MDI internal address offset (which is also equal to the DSP offset) and the 16-bit MCU addresses offset is

$$OFF_{MCU} = OFF_{INT} * 2$$

All MCU accesses to the MDI shared memory should be evenly aligned, 16-bit accesses to ensure valid operation.

5.1.3 Shared Memory Access Contention

Access contentions are resolved in hardware. DSP access has precedence because it runs on a faster clock than the MCU, which is stalled until the DSP access is completed. “Contention” is defined as simultaneous access (read or write) by both MCU and DSP to the same 1/4 Kword of the shared memory. Simultaneous access to different 1/4K blocks of shared memory or to the MDI control registers proceed without stall.

The MCU side contains a data buffer to store a halfword from a write request, enabling the MCU to write with no stall even if the memory array is busy with a DSP access. However, if a second access (read or write) is attempted before the buffer is cleared, the MDI will stall the MCU.

Some stalls may last less then one MCU clock, and so may not even be evident on the MCU side. On the other hand, several consecutive 1-cycle accesses by the DSP to the MDI memory can stall an MCU access for the equivalent number of clock cycles. For example, Example 5-1 show a program loop that transfers data from X to Y memory. Any attempt by the MCU to access the shared memory while the loop is running will be stalled until the loop terminates.

Example 5-1. Program Loop That Stalls MCU Access to Shared Memory

```
      move   x:(r0)+,a
      move   x:(r0)+,b

      DO #(N/2-1), _BE_NASTY_TO_MCU
      move   x:(r0)+,a    a,y:(r4)+    ;r0 points to MDI memory
      move   x:(r0)+,b    b,y:(r4)+    ;r4 points to other memory
_BE_NASTY_TO_MCU
      move   a,y:(r4)+
      move   b,y:(r4)+
```

To avoid a lengthy MCU stall, the DO loop above can be written to allow two cycles per move, making time slots available for MCU accesses, as illustrated in Example 5-2.

Example 5-2. Program Loop With No Stall

```
      DO     #N,_IM_OK_MCU_OK
      move   x:(r0)+,x0        ;r0 points to MDI memory
      move   x0,y:(r4)+        ;r4 points to other memory
_IM_OK_MCU_OK
```

The second instruction in the loop allows pending MCU accesses to execute.

5.1.4 Shared Memory Timing

The DSP always has priority over the MCU when accessing the shared memory. Every DSP access to MDI shared memory or control register lasts one cycle, and is executed as part of the DSP pipeline without stalling it.

In general, an MCU peripheral access is two clock cycles, excluding instruction fetch time. MCU accesses to MDI control registers are always two clock cycles, but shared memory accesses usually take longer, according to the following parameters:

1. **Clock source of the shared memory:** If the DSP is in STOP mode, the shared memory will operate using the MCU clocks generated at half frequency. If the DSP

is active, it will generate the memory clocks at full frequency and all MCU accesses should be synchronized to it.

2. **Access type:** An MCU write is done to a buffer at the MCU side. If the buffer is empty, the MCU takes two cycles to write to the buffer and proceeds without stall; the MDI writes the buffer to the shared memory later, in a minimum of another two MCU cycles, freeing the buffer. In case of a read, or a write when the buffer is not yet free from a previous write, the access will stall.
3. **Relative frequency of the MCU and the DSP clocks:** An MCU access generates a request to the DSP side that must be synchronized to the DSP clock (2 DSP clocks in the worst case), and an acknowledge from the DSP to the MCU side, that must be synchronized to the MCU clock (2 MCU clocks in the worst case). The synchronization stall therefore depends on the frequency of both processors. The slower the DSP frequency is, relative to the MCU frequency, the longer the access time (measured in MCU clocks). In a typical system configuration, the DSP's frequency is higher or equal to the MCU's frequency. In this assumption, the maximum MCU stall is if the frequencies of the MCU and the DSP are equal. If the DSP frequency is lower than the MCU frequency, the access time (measured in MCU clocks) may in principle be very long, depending on how slow the DSP is.
4. **DSP parallel accesses:** Any DSP access in parallel to an MCU access to the same 1/4K memory block can further stall a pending MCU access. If the DSP does not run consecutive one-cycle accesses and the MCU frequency is not faster than the DSP's frequency, an MCU contention stall will be no more than one MCU cycle.
5. **DSP PLL:** If the PLL is reprogrammed during MCU program execution, (e.g., after a DSP reset) the MCU should not access shared memory until the PLL has reacquired lock. If the MCU attempts to access the MDI shared memory before the PLL acquires lock, the MCU can time out and generate an error. One way to avoid this condition is to take the following steps:
    a. DSP software sets an MDI flag bit immediately after setting the PLL.
    b. MCU software polls the flag bit until it is set before accessing MDI shared memory.

MCU-side access timing is summarized in Table 5-1.

Table 5-1. MCU MDI Access Timing

Access Type	DSP Clocks	MCU Cycles[1]		Comments
		Minimum	Maximum	
Shared memory read	Inactive	11	11	Assumes write buffer is empty.
	Active	4	8	
Shared memory write	Either	2	2	Assumes write buffer is empty.
Buffer busy after shared memory write	Inactive	+ 8	+ 8	Consecutive accesses incur MCU stall cycles.
	Active	+ 2	+ 4	
MCU-DSP shared memory contention	Active	+ 0	+ 1	MCU stalls until DSP access completes. Multiple DSP one-cycle instructions stall the MCU further.
Control registers	Either	2	2	—

[1] Minimum case: DSP clock frequency >> MCU clock frequency.
Maximum case: DSP clock frequency = MCU clock frequency.
(More cycles required if DSP clock < MCU clock.)

5.2 MDI Messages and Control

The MDI provides a means for the MCU and DSP to exchange messages independent of the shared memory array. A typical message might be "I have just written a message of N words, starting at offset X in memory," or "I have just finished reading the last data block sent." For ease and flexibility, the protocol for exchanging these messages is not predefined in hardware but can be implemented with a few simple software commands.

5.2.1 MDI Messaging System

Messages are exchanged between the two processors through special-purpose control registers. Most of these registers are symmetric and work together to exchange messages in the following ways:

1. Each of two 16-bit write-only transmit registers is copied in a corresponding read-only receive register on the other processor's side. These registers can be used to transfer 16-bit messages or frame information about messages written to the shared memory, such as number of words, initial address, and message code type.
2. Writing to a transmit register clears a "transmitter empty" bit in the status register on the transmitter side and sets a "receiver full" bit in the status register on the receiver side, which can trigger a maskable receive interrupt on the receiver side if so programmed.

3. Reading a receive register automatically clears the “receiver full” bit in the status register on the receiver side and sets the “transmitter empty” bit in the status register on the transmitter side, which can trigger a maskable transmit interrupt on the transmitter side if so programmed.
4. Three general purpose flags are provided for each transmitter and reflected in the status register at the receiver side.

The symmetry of the MDI registers is illustrated in Figure 5-4 and Table 5-2.

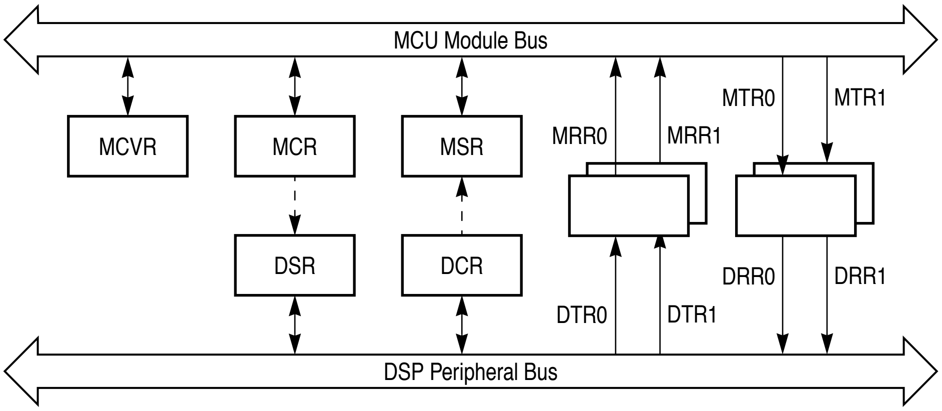
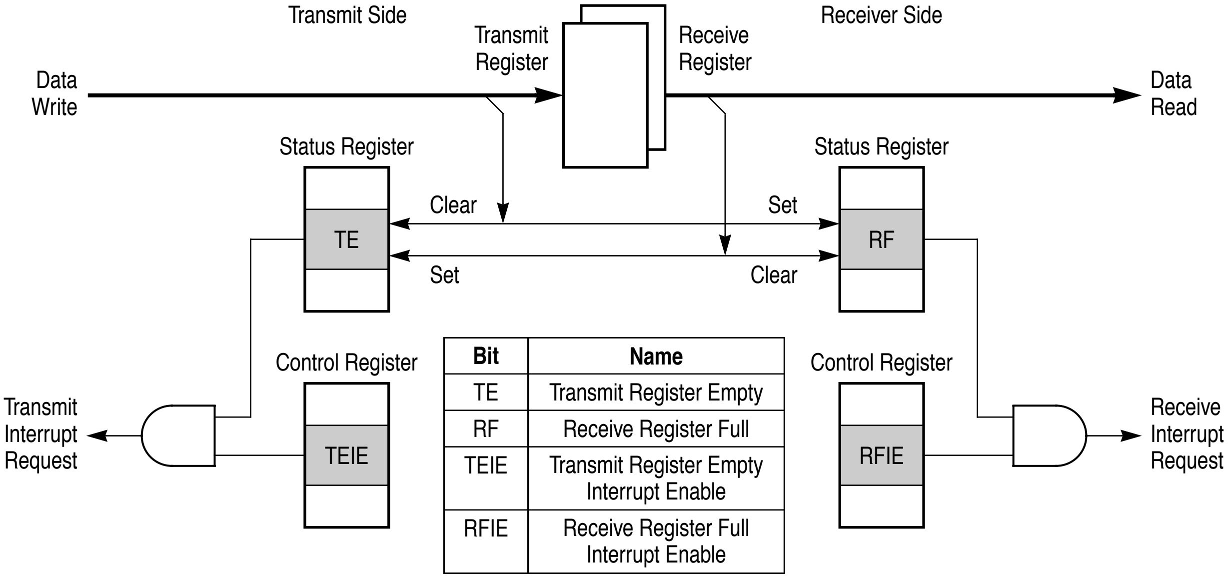


Figure 5-4. MDI Register Symmetry

Table 5-2. MDI Registers and Symmetry

MCU Registers		DSP Registers	
Acronym	Name	Acronym	Name
MRR0	MCU Receive Register 0	DTR0	DSP Transmit Register 0
MRR1	MCU Receive Register 1	DTR1	DSP Transmit Register 1
MTR0	MCU Transmit Register 0	DRR0	DSP Receive Register 0
MTR1	MCU Transmit Register 1	DRR1	DSP Receive Register 1
MSR	MCU Status Register	DCR	DSP Control Register
MCR	MCU Control Register	DSR	DSP Status Register
MCVR	MCU Command Vector Register	—	

The message exchange mechanism is shown in greater detail in Figure 5-5.

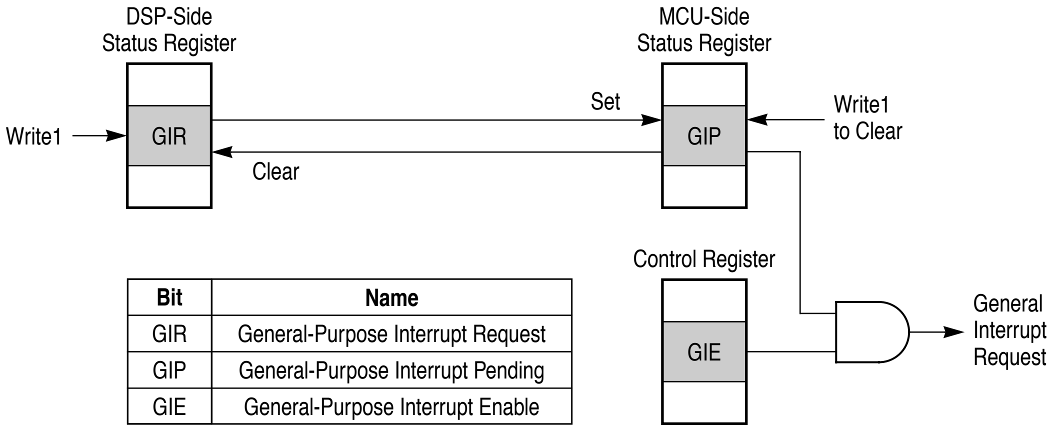


Bit	Name
TE	Transmit Register Empty
RF	Receive Register Full
TEIE	Transmit Register Empty Interrupt Enable
RFIE	Receive Register Full Interrupt Enable

Figure 5-5. MDI Message Exchange

In addition to exchanging messages, the MDI registers also provide the following special-purpose control functions:

1. Each core’s power mode is reflected in the other core’s status register.
2. Each core can issue an interrupt to wake the other core from its low-power modes (STOP and WAIT modes on either side, plus DOZE mode on the MCU side).
3. The MCU can issue a Command Interrupt to the DSP by setting the MC bit in the MCU Command Vector Register (MCVR). Software can write the vector address of this interrupt to a register on the MCU side. The Command Interrupt can be maskable or non-maskable.
4. The MCU can issue a hardware reset to the DSP. (The DSP cannot issue a hardware reset to the MCU.)
5. The DSP can issue two general-purpose interrupt requests to the MCU by setting the DGIR0 or DGIR1 bit in the DSP-Side Status Register (DSR). These interrupts are user-maskable on the MCU side. Figure 5-6 details the mechanism by which the DSP issues a general-purpose interrupt to the MCU.



Bit	Name
GIR	General-Purpose Interrupt Request
GIP	General-Purpose Interrupt Pending
GIE	General-Purpose Interrupt Enable

Figure 5-6. DSP-to-MCU General Purpose Interrupt

The MCU-to-DSP interrupt mechanism (Command Interrupt) differs from Figure 5-6 in the following ways:

1. The interrupt pending bit (the MCP bit in the DSR) is cleared automatically when the interrupt is acknowledged.
2. The trigger bit on the MCU side (the MC bit) is in the MCVR.
3. When a non-maskable interrupt is generated, the interrupt enable bit on the DSP side (the MCIE bit in the DCR) is ignored.

5.2.2 Message Protocols

The message hardware can be used by software to implement message protocols for a wide array of message types. Full support is given for both interrupt and polling management. The following are examples of different message protocols:

- A message of up to 16 bits is written directly to one of the transmit registers.
- Both transmit registers are used to pass a 2-word message. The corresponding receive register of the first word disables its interrupt; the register receiving the second word enables its interrupt. An interrupt is triggered when the second word is received.
- Transmit registers pass frame information describing longer messages written to the shared memory. Such frame information usually includes an initial address, the number of words, and often a message type code.
- A DSP general interrupt or the MCU Command Interrupt signals an event or request that does not include data words, such as acknowledging the read of a long message from the shared memory.

- Fixed-length, formatted data is written in a predetermined location in the shared memory. A general purpose interrupt (DSP) or command interrupt (MCU) signals the other processor that the data is ready.
- One processor uses the three general-purpose flags to inform the other processor of its current program state.

5.2.3 MDI Interrupt Sources

The MDI provides several ways to generate interrupts to both the DSP and MCU.

5.2.3.1 DSP Interrupts

There are five independent ways for the MCU to interrupt the DSP through the MDI:

1. MCU Command Vector interrupt
2. MDI receive/transmit interrupt
3. MDI DSP wake from STOP / general-purpose interrupt (using the $\overline{\text{IRQC}}$ interrupt input)
4. Protocol Timer DSP wake from STOP / general-purpose interrupt (using the $\overline{\text{IRQD}}$ interrupt input)
5. External DSP wake from STOP / general-purpose interrupt (using the $\overline{\text{IRQB}}$ interrupt input)

The first three interrupts are MDI functions. The other two are protocol timer functions that make use of MDI hardware but have no specific MDI instructions. The interrupts can be prioritized in Core Interrupt Priority Register (IPRC). See Table 7-9 on page 7-15.

The relative priority of the MDI receive/transmit interrupts is fixed as follows:

1. Receive register 0 full (RFIE0)
2. Receive register 1 full (RFIE1)
3. Transmit register 0 empty (TEIE0)
4. Transmit register 1 empty (TEIE1)

5.2.3.2 MCU Interrupts

There is only one interrupt request line to the MCU interrupt controller. The interrupt service routine must examine the MCU-Side Status Register (MSR) to determine the interrupt source. The Find First One (FF1) instruction can be used for this purpose. If some of the interrupts are disabled, software can read the MDI Control Register (MCR) and perform an AND operation with the MSR before executing the FF1 instruction. The interrupt service routine should clear the General Purpose Interrupt Pending bits (MGIP[1:0], MSR bits 11–10) to deassert the request to the interrupt controller.

5.2.4 Event Update Timing

An information exchange between the two processors that is reflected in the status register of the receiving processor (an “event”) incurs some latency. This latency is the delay between the event occurrence at one processor and the resulting update in the status register of the other processor. The latency can be expressed as the sum of a number of transmitting-side clocks (TC) and receiving-side clocks (RC).

The minimum event latency occurs when there are no other events pending, and is equal to
TC + 2(RC).

The maximum event latency is incurred when the event occurs immediately after a previous event is issued. It is equal to 4(TC) + 6(RC).

5.2.5 MCU-DSP Troubleshooting

The MCU can use the MDI in the following three ways to identify and correct the source of a DSP malfunction:

1. Examine the DPM bit in the MSR to determine if the DSP is stuck in STOP mode. If so, the MCU can wake the DSP by setting the DWS bit.
2. Issue an NMI using the Command Interrupt (setting the MC bit in the MCVR). The NMI service routine can incorporate a diagnostic procedure designed for such an event. Note that the MNMI bit must also be set to enable non-maskable interrupts.
3. If neither of the first two measures is effective, the MCU can issue a hardware reset to the DSP by setting the DRS bit in the MCR.

5.3 Low-Power Modes

Each side of the MDI is fully active in all low-power modes except STOP. Each processor can enter and exit a low-power mode independently. The processor state is unchanged by a transition to and from a low-power mode—status and control registers do not return to default values.

5.3.1 MCU Low-Power Modes

Various DSP events can awaken the MCU from a low-power mode (WAIT, DOZE, or STOP) by generating a corresponding interrupt. Table 5-3 lists the events and the associated interrupt enable bits in the MCR.

Table 5-3. MCU Wake-up Events

Event	Interrupt Enable Bit in MCR
Transmitting a message to MRR0	15 (MRIE0)
Transmitting a message to MRR1	14 (MRIE1)
Receiving a message from MTR0	13 (MTIE0)
Receiving a message from MTR1	12 (MTIE1)
Setting the DGIR0 bit in the DSR (General Interrupt request 0)	11 (MGIE0)
Setting the DGIR1 bit in the DSR (General Interrupt request 1)	10 (MGIE1)

The software designer should consider the following points before placing the MCU in STOP mode:

1. Compatibility with DSP STOP mode protocol. MCU software should accommodate the possibility that the DSP is in STOP when the MCU awakens from its STOP mode.
2. Pending shared memory writes. A shared memory write that has not completed when the MCU enters STOP mode will execute reliably after the MCU has awakened. Nevertheless, the user may wish to ensure that all shared memory writes are completed before entering STOP. This can be done by polling MSR bit 6 until it is cleared before issuing the STOP instruction.
3. Pending MCU events. MCU software should poll the MEP bit in the MSR until it is cleared just before issuing the STOP instruction. This ensures that the DSP has acknowledged all previous MCU-generated events so that it can be made aware of the MCU power mode change.

5.3.2 DSP Low-Power Modes

The MCU can wake the DSP from WAIT mode by issuing any of the interrupts listed in Section 5.2.3.1 on page 5-10.

MCU software can wake the DSP from STOP in one of the following three ways:

1. A DSP Wake from STOP command (setting the DWS bit in the MSR).
2. A Protocol Timer DSP interrupt.
3. A DSP hardware reset (setting the DHR bit in the MCR).

The MCU can also wake the DSP externally with an external DSP interrupt, external DSP debug request, JTAG DSP debug command, or system reset.

DSP software should ensure that the MCU can track each DSP transition to and from STOP mode before the next one occurs. This is essential for proper control of the shared memory clock (see Section 5.3.3). One way to accomplish this is to provide a minimum delay (measured in MCU clocks) between consecutive DSP entrances to STOP mode. Another method involves waiting for MDI register events to terminate to supply the needed delay. With this method the DSP sends at least one MDI register event and waits until the DEP bit in the DSR is cleared before it enters STOP mode. To be sure that an event takes place, DSP code can issue a dummy event such as the one illustrated in Example 5-3. The DEP check should be the last MDI access before issuing the STOP instruction to guarantee that the MSR is updated properly.

Example 5-3. Dummy Event to Allow MCU to Track DSP Power Mode Change

```
            movep       x:<<DCR,x0
            movep       x0,x:<<DCR;   ;dummy event - write back flags
            nop                       ;nops for pipeline delay
            nop
            nop
_wait       jset        #DEP,x:<<DSR,_wait
            stop
```

After a DSP wake from STOP command, $\overline{\text{IRQC}}$ should be deasserted by writing “1” to the DWSC bit in the DSR. Similarly, after a protocol timer interrupt event, $\overline{\text{IRQD}}$ should be deasserted by writing “1” to the DTIC bit in the DSR. Clearing either of these bits just as the DSP exits STOP can serve as the MDI register event for the delay required before the next entry to STOP mode.

5.3.3 Shared Memory in DSP STOP Mode

The shared memory array operates from the DSP clock for either processor unless the DSP is in STOP mode. MCU access to the shared memory is internally synchronized to the DSP clock. Memory access signals from the MCU require 2 DSP cycles to synchronize to the DSP clock, and 2 MCU cycles to synchronize the DSP acknowledgment to the MCU clock. If the DSP runs at a relatively low frequency, extra wait states are added to the MCU access.

Note: The synchronization wait states are not related to wait states resulting from memory contention.

When the DSP is in STOP mode and the MCU is in normal mode, the shared memory operates from the MCU clock. The memory controller is alerted when the DSP has exited

STOP mode and stalls any pending MCU shared memory access until the memory clocks are switched back to the DSP.

Note: Waking the DSP from STOP can take several MCU clocks. The parameters affecting the relative time length include the DSP frequency relative to the MCU frequency, the need for PLL relock, and the state of the SD bit in the OMR. If the total wake from STOP delay is greater than 128 MCU clocks, a pending MCU shared memory access can be lost due to an MCU time-out interrupt. MCU shared memory writes that are separated by MSR bit 6 checks are not subject to this loss because the write is done to a buffer and the MCU bus is released.

5.4 Resetting the MDI

The MDI can be reset by any of the conditions in Table 5-4.

Table 5-4. MDI Reset Sources

Reset Type	Action	Description
MDI Reset	Setting the MDIR bit in the MCR	Only the MDI system is reset—all status and control registers are returned to their default values. None of the rest of the DSP56652 system is affected. **Note:** MDIR assertion is ignored if the DSP is in STOP mode.
DSP Hardware Reset	Setting the DHR bit in the MCR	In addition to the MDI reset conditions above, the entire DSP side is reset. Memory, including MDI shared memory, is not affected. MCU software should poll the DRS bit (MSR bit 7) to determine when the reset sequence on the DSP side has ended (and wait for PLL relock if the PLL is reprogrammed—see page 5-5) before accessing the shared memory.
System Reset	Power on reset $\overline{\text{RESET\_IN}}$ asserted Watchdog timer time-out	The entire system, including memory, is reset.

Note that the DSP software RESET instruction does *not* reset the MDI.

Before initiating an MDI reset, the following items should be considered:

1. **Pending shared memory write—**If an MCU write to the shared memory is pending in the write buffer when an MDI reset is initiated, the access may be lost. To ensure that the data is written, software should poll the MSMP bit in the MSR until it is cleared before triggering the MDI reset.
2. **DSP MDI operations—**MDIR assertion is asynchronous to DSP operation, and can cause unpredictable behavior if it occurs while the DSP is testing an MDI

register bit with an instruction such as `jset #DTE0,x:DSR,tx_sbr`. MCU software should verify that the DSP is not engaged in MDI signalling activity before asserting MDIR. This can be done by performing the following steps:

a. Disable the DSP interrupt event in the Protocol Timer by clearing the DSIE bit in the PTIER.

b. Verify that both DWS and MTIR (MSR bits 8 and 9) are cleared.

The instruction immediately following assertion of the MDIR bit may be overridden by the reset sequence, with all registers retaining their reset values. Therefore, software should wait at least one instruction before writing to MDI registers.

5.5 MDI Software Restriction Summary

Tables 5-5 through 5-7 summarize the various constraints on MDI software.

Table 5-5. General Restrictions

Action	Restriction
Writing to a transmit register	Wait for a Transmitter Empty interrupt or poll the Transmitter Empty bit in the status register
Reading from a receive register	Wait for a Receiver Full interrupt or poll the Receiver Full bit in the status register.

Table 5-6. DSP-Side Restrictions

Action	Restriction
Setting DGIR(0,1) to issuing general interrupt request	Verify that DGIR(0,1) is cleared
Configuring $\overline{\text{IRQC}}$ and $\overline{\text{IRQD}}$	Define $\overline{\text{IRQC}}$ as level-triggered by clearing the ICTM bit in the IPRC. Define $\overline{\text{IRQD}}$ as level-triggered by clearing the IDTM bit in the IPRC.
Delay between MDI register write and reflection in DSR	A delay of up to four instructions can occur between an MDI register write and the resulting change in the DSR. Refer to the 56600 Family Manual, Appendix B, Section 5 (“Peripheral Pipeline Restrictions”) for a description of possible problems and work-arounds. Testing the DEP bit in the DSR requires one additional clock delay above the 56600 manual description.
Continuous one-cycle accesses to the Shared Memory	Can stall MCU. Refer to Example 5-2 on page 5-4 for sample code that avoids lengthy MCU stalls.
Entering DSP STOP mode	Enable $\overline{\text{IRQC}}$—write a non-zero value to the ICPL bits in the IPRC. Enable $\overline{\text{IRQD}}$—write a non-zero value to the IDPL bits in the IPRC. Ensure minimum delay from previous STOP mode (Section 5.3.2 on page 5-12). Ensure the DEP bit in the DSR is cleared.

Table 5-6. DSP-Side Restrictions

Action	Restriction
Clearing serviced interrupts	Write 1 to the DWSC bit in the DSR to clear $\overline{\text{IRQC}}$. Write 1 to the DTIC bit in the DSR to clear $\overline{\text{IRQD}}$.

Table 5-7. MCU-Side Restrictions

Action	Restriction
Byte-wide writes to shared memory	The MDI latches all 16 bits when receiving data written to it. In byte-wide writes, the MCU drives only the written 8 bits; the unspecified byte in the shared memory location may contain corrupt data.
Writing to MCVR	Ensure that the MC bit in the MCVR is cleared before writing.
Setting the DWS bit in the MSR	Ensure DWS is cleared before setting it.
PT timer DSP interrupt	If the MSIR bit in the MSR is set when the protocol timer issues a dsp_int event (i.e., a previous DSP interrupt event has not been serviced) the second interrupt request is lost.
Entering MCU STOP mode	Verify that the MEP bit in the MSR is clear.
MDI reset	Before setting the MDIR bit in the MCR or DHR (MCR bit 7), do the following: 1. Disable the DSP Protocol Timer interrupt by clearing the DSIE bit in the PT Interrupt Enable Register (PTIER). 2. Verify that the DWS bit in the MSR is cleared to ensure that the DSP has serviced the last wake-up from STOP. 3. Verify that the DTIC bit in the DSR is cleared to ensure that there are no outstanding protocol timer interrupt requests. 4. Poll the MSMP bit in the MSR until it is cleared to ensure all shared memory writes occur. In addition, before setting MDIR, do the following: 1. Verify that the DSP side is not engaged in MDI activity (e.g. by issuing NMI). 2. Check that the DPM bit in the MSR is cleared, indicating that DSP is not in STOP mode. (Hardware will ignore the MDIR bit if DSP is in STOP mode). After asserting MDIR, delay at least one instruction time before writing to an MDI register to ensure it is not overwritten by reset. After any MDI reset (MCU or DSP hardware reset, asserting MDIR, or asserting DHR) poll the DRS bit in the MSR until it is cleared before accessing the shared memory to ensure DSP reset is complete.
After DSP reset	Ensure that the DSP PLL has been relocked (e.g., item 5 on page 5-5) before the MCU accesses shared memory.

5.6 MDI Registers

In general, the MDI registers on the DSP side and MCU side are symmetrical. They are summarized in Table 5-8.

Table 5-8. MDI Signalling and Control Registers

	MCU Side		DSP Side	
Function	**Name**	**Address**	**Name**	**Address**
MCU Command Vector Register	MCVR	$0020_2FF2	—	
Control Register	MCR	$0020_2FF4	DCR	X:$FF8A
Status Register	MSR	$0020_2FF6	DSR	X:$FF8B
Transmit Register 1	MTR1	$0020_2FF8	DTR1	X:$FF8C
Transmit Register 0	MTR0	$0020_2FFA	DTR0	X:$FF8D
Receive Register 1	MRR1	$0020_2FFC	DRR1	X:$FF8E
Receive Register 0	MRR0	$0020_2FFE	DRR0	X:$FF8F

The correspondence between transmit registers on one side and receive registers on the other side is listed in Table 5-9.

Table 5-9. MCU–DSP Register Correspondence

MCU Register	MCU Address	DSP Register	DSP Address
MTR1	$0020_2FF8	DRR1	X:$FF8E
MTR0	$0020_2FFA	DRR0	X:$FF8F
MRR1	$0020_2FFC	DTR1	X:$FF8C
MRR0	$0020_2FFE	DTR0	X:$FF8D

5.6.1 MCU-Side Registers

MCVR MCU Command Vector Register **$0020_2FF2**

	BIT 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT 0
	▨	▨	▨	▨	▨	▨	▨	MC	MCV[6:0]							MNMI
RESET	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0

Table 5-10. MCVR Description

Name	Type[1]	Description	Settings
MC Bit 8	R/1S	**MCU Command—**Used to initiate a DSP interrupt. Setting the MC bit sets the MCP bit in the DSR. If the MNMI bit in this register is set, a non-maskable MCU command interrupt is issued at the DSP side. If MNMI is cleared and the MCIE bit in the DCR is set, a maskable interrupt request is issued at the DSP side. The MC bit is cleared only when the command interrupt is serviced on the DSP side, providing a way for the MCU to monitor interrupt service status. The MCVR cannot be written while the MC bit is set.	0 = No outstanding DSP command interrupt (default). 1 = DSP command interrupt has been issued and has not been serviced.
MCV[6:0] Bits 7–1	R/W	**MCU Command Vector—**Vector address displacement for the DSP command interrupt. With this mechanism the MCU can activate any interrupt from the DSP interrupt table. The actual vector value is twice the value of MCV[6:0]. The MCV bits can only be written if the MC bit is cleared.	
MNMI Bit 0	R/W	**MCU Non-Maskable Interrupt—**Determines if the Command Interrupt issued to the DSP by setting the MC bit is maskable or non-maskable. The MNMI bit can only be written if the MC bit is cleared.	0 = Maskable interrupt issued when MC is set, if DSP DCR bit 8 (maskable interrupt enable) is set (default). 1 = Non-maskable interrupt generated when MC is set. DCR bit 8 is ignored.

[1] R = Read only.
R/W = Read/write
R/1S = Read; write with 1 to set (write with 0 ignored).

MCR MCU-Side Control Register **$0020_2FF4**

	BIT 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT 0
	MRIE0	MRIE1	MTIE0	MTIE1	MGIE0	MGIE1			DHR	MDIR				MDF[2:0]		
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The MCR is a 16-bit read/write register that enables the MDI interrupts on the MCU side and enables the trigger events on the DSP side (e.g. awaken from Stop mode, hardware reset, flag update, etc.).

Note: Either the EMDI bit in the NIER or the EFMDI bit in the FIER must be set in order to generate any of the interrupts enabled in the MCR (see page 7-7).

Table 5-11. MCR Description

Name	Type[1]	Description	Settings
MRIE0 Bit 15	R/W	**MCU Receive Interrupt Enable 0**—When MRIE0 is set, a receive interrupt request 0 is issued when the MRF0 bit in the MSR is set. When MRIE0 is cleared, MRF0 is ignored and no receive interrupt request 0 is issued.	0 = Receive interrupt 0 request disabled (default). 1 = Enabled.
MRIE1 Bit 14	R/W	**MCU Receive Interrupt Enable 1**—When MRIE1 is set, a receive interrupt request 1 is issued when the MRF1 bit in the MSR is set. When MRIE1 is cleared, MRF1 is ignored and no receive interrupt request 1 is issued.	0 = Receive interrupt 1 request disabled (default). 1 = Enabled.
MTIE0 Bit 13	R/W	**MCU Transmit Interrupt Enable 0**—If MTIE0 is set, a transmit interrupt 0 request is generated when the MTE0 bit in the MSR is set. If MTIE0 bit is cleared, MTE0 is ignored and no transmit interrupt request 0 is issued.	0 = Transmit interrupt 0 request disabled (default). 1 = Enabled.
MTIE1 Bit 12	R/W	**MCU Transmit Interrupt Enable 1**—If MTIE1 is set, a transmit interrupt 1 request is generated when the MTE1 bit in the MSR is set. If MTIE1 bit is cleared, MTE1 is ignored and no transmit interrupt request 1 is issued.	0 = Transmit interrupt 1 request disabled (default). 1 = Enabled.
MGIE0 Bit 11	R/W	**MCU General Interrupt Enable 0**—If this bit is set, a general interrupt 0 request is issued when the MGIP0 bit in the MSR is set. If MGIE0 is clear, MGIP0 is ignored and no general interrupt request 0 is issued.	0 = General interrupt 0 request disabled (default). 1 = Enabled.
MGIE1 Bit 10	R/W	**MCU General Interrupt Enable 1**—If this bit is set, a general interrupt 1 request is issued when the MGIP1 bit in the MSR is set. If MGIE1 is clear, MGIP1 is ignored and no general interrupt request 1 is issued.	0 = General interrupt 1 request disabled (default) 1 = Enabled

Table 5-11. MCR Description (Continued)

Name	Type[1]	Description	Settings
DHR Bit 7	R/W	**DSP Hardware Reset—**Setting DHR issues a hardware reset to the DSP. Clearing DHR de-asserts the reset. Setting DHR also causes MDI reset, returning all MDI control and status bits to their default values (except the DHR bit itself). DHR should be held asserted for a minimum of three CKIL cycles. (See Reset, Mode Select, and Interrupt Timing in the DSP56652 Technical Data Sheet.) After clearing DHR, software should poll the DRS bit in the MSR until it is cleared before attempting an access to MDI shared memory. If an MDI reset (caused by MDIR or DHR being set) is done while an MCU write to the shared memory is pending in the write buffer, the access may be lost.	
MDIR Bit 6	R0/1S	**MDI Reset—**Setting MDIR resets the message and control sections on both DSP and MCU sides. All control and status registers except DHR are returned to their default values and all internal states are cleared. Data in the shared memory array remains intact; only the access control logic is affected. After setting MDIR, software should poll DRS to determine when the reset sequence on the DSP side has ended before accessing the shared memory.	
MDF[2:0] Bits 2–0	R/W	**MCU-to-DSP Flags—**General-purpose flag bits that are reflected on the DSP side in the DF[2:0] bits in the DSR.	

[1] R/W = Read/write
R0/1S = Always read as 0; write with 1 to set (write with 0 ignored).

MSR MCU-Side Status Register $0020_2FF6

	BIT 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT 0
	MRF0	MRF1	MTE0	MTE1	MGIP0	MGIP1	MTIR	DWS	DRS	MSMP	DPM	MEP	▨	MF[2:0]		
RESET	0	0	1	1	0	0	0	0	1	0	—	0	0	—	—	—

Table 5-12. MSR Description

Name	Type[1]	Description	Settings
MRF0 Bit 15	R	**MCU Receive Register 0 Full—**Set when the DSP writes to DTR0, indicating to the MCU that the reflected data is available in MRR0. MRF0 is cleared when the MCU reads MRR0.	0 = Latest MRR0 data has been read (default). 1 = New data in MRR0.
MRF1 Bit 14	R	**MCU Receive Register 0 Full—**Set when the DSP writes to DTR1, indicating to the MCU that the reflected data is available in MRR1. MRF1 is cleared when the MCU reads MRR1.	0 = Latest MRR1 data has been read (default). 1 = New data in MRR1.
MTE0 Bit 13	R	**MCU Transmit Register 0 Empty—**Cleared when the MCU writes to MTR0; set when the DSP reads the reflected data in DRR0.	0 = DRR0 has not been read. 1 = DRR0 has been read (default).
MTE1 Bit 12	R	**MCU Transmit Register 1 Empty—**Cleared when the MCU writes to MTR1; set when the DSP reads the reflected data in DRR1.	0 = DRR1 has not been read. 1 = DRR1 has been read (default).
MGIP0 Bit 11	R/1C	**MCU General Interrupt 0 Pending—** Indicates that the DSP has requested an interrupt by setting the DGIR0 bit in the DSR.	0 = No interrupt request (default). 1 = DSP has issued interrupt request 0.
MGIP1 Bit 10	R/1C	**MCU General Interrupt 1 Pending—** Indicates that the DSP has requested an interrupt by setting the DGIR1 bit in the DSR.	0 = No interrupt request (default). 1 = DSP has issued interrupt request 1.
MTIR Bit 9	R	**MCU Protocol Timer Interrupt Request—** Set by the protocol timer when it issues a dsp_int event (see Table 10-4 on page 10-13) which asserts DSP $\overline{\text{IRQD}}$ (waking the DSP from STOP mode) and $\overline{\text{IRQA}}$, which is wire-or'd to $\overline{\text{IRQD}}$. MTIR is cleared when the DSP sets the DTIC bit in the DSR (Table 5-18 on page 5-25) at the end of its $\overline{\text{IRQD}}$ service routine. For proper MTIR operation, $\overline{\text{IRQD}}$ should be enabled via IPRC bits 10–9 and made level-sensitive by clearing IPRC bit 11. Software should verify that MTIR is cleared before issuing an MDI reset (setting the MDIR bit in the MCR).	0 = No outstanding MTIR-generated interrupt request (default). 1 = DSP has not serviced last MTIR-generated interrupt.

Table 5-12. MSR Description (Continued)

Name	Type[1]	Description	Settings
DWS Bit 8	R/1S	**DSP Wake From STOP—**Set by MCU software to wake the DSP from STOP mode. Setting DWS also asserts DSP $\overline{\text{IRQC}}$ (waking the DSP from STOP mode) and $\overline{\text{IRQA}}$, which is wire-or'd to $\overline{\text{IRQC}}$. DWS is cleared when the DSP sets the DWSC bit in the DSR (Table 5-18 on page 5-25) at the end of its IRQC service routine. $\overline{\text{IRQC}}$ should be enabled via the ICPL bit in the IPRC and made level-sensitive by clearing the ICTM bit in the IPRC. Software should verify that DWS is cleared before issuing an MDI reset.	0 = No outstanding DWS-generated interrupt request (default). 1 = DSP has not serviced last DWS-generated interrupt.
DRS Bit 7	R	**DSP Reset State—**Set by any DSP reset: • MCU system reset • DSP hardware reset (caused by setting the DHR bit in the MCR) • MDI reset (caused by setting the MDIR bit in the MCR) DRS is cleared by DSP hardware as it completes the reset sequence. Software should ensure that DRS is cleared before accessing MDI shared memory.	0 = DSP has completed the most recent reset sequence. 1 = DSP has not completed the most recent reset sequence (default).
MSMP Bit 6	R	**MCU Shared Memory Access Pending—** Set by an MCU write to MDI shared memory. Cleared when write access is complete. Software should ensure that MSMP is cleared before issuing an MDI reset to ensure that no pending write is lost.	0 = No outstanding MCU-MDI write (default). 1 = Last MCU write to MDI shared memory has not been completed.
DPM Bit 5	R	**DSP Power Mode—**Reflects the DSP mode of operation.	0 = DSP is in normal or WAIT mode (default). 1 = DSP is in STOP mode.
MEP Bit 4	R	**MCU-Side Event Pending—**Set when the MCU sends an event update request to the DSP side. Cleared when the event update acknowledge has been received. An "event" is any hardware message that should be reflected in the DSR on the DSP-side (e.g., "transmit register 0 written"). Software should poll MEP until it is cleared before entering STOP mode. Reading the MSR to check the MEP bit should be the last MDI access before entering STOP, otherwise the MEP can be set as a result of that additional action. If MEP is not properly verified, entering the MCU STOP power mode may not to be reflected at the DSR.	0 = Last event update request to DSP has been acknowledged. 1 = Event update request to DSP pending.
MF[2:0] Bits 2–0		**MCU Flags—**General-purpose flag bits reflecting the state of DMF[2:0] (DCR bits 2–0).	0 = Corresponding DMF bit cleared. 1 = Corresponding DMF bit set.

1. R = Read only.
R/1S = Read, or write with 1 to set (write with 0 ignored).
R/1C = Read, or write with 1 to clear (write with 0 ignored).

MTR1 MCU Transmit Register 1 **$0020_2FF8**

	BIT 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT 0
	Transmitted data from MCU to DSP															
RESET	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Table 5-13. MTR1 Description

MTR1 is a 16-bit write-only register. Data written to MTR1 is reflected on the DSP side in DRR1. MTR1 and DRR1 are not double buffered. Writing to MTR1 overwrites the data in DRR1, clears the MCU Transmit Register 1 Empty bit (MTE1) in the MSR, and sets the DSP Receive Register 1 Full bit (DRF1) in the DSR. It can also trigger a receive interrupt on the DSP side if the DRIE1 bit in the DCR is set. A single 8-bit write to MTR1 also updates all status information.

MTR0 MCU Transmit Register 0 **$0020_2FFA**

	BIT 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT 0
	Transmitted data from MCU to DSP															
RESET	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Table 5-14. MTR0 Description

MTR0 is a 16-bit write-only register. Data written to MTR0 is reflected on the DSP side in DRR0. MTR0 and DRR0 are not double buffered. Writing to MTR0 overwrites the data in DRR0, clears the MCU Transmit Register 0 Empty (MTE0) bit in the MSR, and sets the DSP Receive Register 0 Full bit (DRF0) in the DSR. It can also trigger a receive interrupt on the DSP side if the DRIE0 bit in the DCR is set. A single 8-bit write to MTR0 also updates all status information.

MRR1 MCU Receive Register 1 **$0020_2FFC**

	BIT 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT 0
	Transmitted data from MCU to DSP															
RESET	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Table 5-15. MRR1 Description

MRR1 is a 16-bit read-only register that reflects the data written on the DSP side to DTR1. Reading MRR1 clears the MCU Receive Register 1 Full bit (MRF1) in the MSR and sets the DSP Transmit Register 1 Empty bit (DTE1) in the DSR. It can also trigger a transmit interrupt on the DSP side if the DTIE1 bit in the DCR is set. A single 8-bit read from MRR1 also updates all status information.

MRR0 MCU Receive Register 0 **$0020_2FFE**

	BIT 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT 0
	Transmitted data from MCU to DSP															
RESET	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Table 5-16. MRR0 Description

MRR0 is a 16-bit read-only register that reflects the data written on the DSP side to DTR0. Reading MRR0 clears the MCU Receive Register 0 Full bit (MRF0) in the MSR and sets the DSP Transmit Register 0 Empty bit (DTE0) in the DSR. It can also trigger a transmit interrupt on the DSP side if the DTIE0 bit in the DCR is set. A single 8-bit read from MRR0 also updates all status information.

5.6.2 DSP-Side Registers

DCR DSP-Side Control Register **X:$FF8A**

	BIT 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT 0
	DTIE0	DTIE1	DRIE0	DRIE1				MCIE						DMF[2:0]		
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Note: The MDIPL bits in the Peripheral Interrupt Priority Register (IPRP) must written with a non-zero value in order to generate any of the interrupts enabled in the DCR (see page 7-7).

Table 5-17. DCR Description

Name	Description	Settings
DTIE0 Bit 15	**DSP Transmit Interrupt Enable 0—**If DTIE0 is set, a transmit interrupt 0 request is generated when the DTE0 bit in the DSR is set. If DTIE0 bit is cleared, DTE0 is ignored and no transmit interrupt request 0 is issued.	0 = Transmit interrupt 0 request disabled (default). 1 = Enabled.
DTIE1 Bit 14	**DSP Transmit Interrupt Enable 1—**If DTIE1 is set, a transmit interrupt 1 request is generated when the DTE1 bit in the DSR is set. If DTIE1 bit is cleared, DTE1 is ignored and no transmit interrupt request 1 is issued.	0 = Transmit interrupt 1 request disabled (default). 1 = Enabled.
DRIE0 Bit 13	**DSP Receive Interrupt Enable 0—**When DRIE0 is set, a receive interrupt request 0 is issued when the DRF0 bit in the DSR is set. When DRIE0 is cleared, DRF0 is ignored and no receive interrupt request 0 is issued.	0 = Receive interrupt 0 request disabled (default). 1 = Enabled.
DRIE1 Bit 12	**DSP Receive Interrupt Enable 1—**When DRIE1 is set, a receive interrupt request 1 is issued when the DRF1 bit in the DSR is set. When DRIE1 is cleared, DRF1 is ignored and no receive interrupt request 1 is issued.	0 = Receive interrupt 1 request disabled (default). 1 = Enabled.
MCIE Bit 8	**MCU Command Interrupt Enable—**If this bit is set, the MCP bit in the DSR is set, and the MNMI bit in the MCVR is clear, a maskable command interrupt is issued. If MNMI is set, MCIE is ignored. In this case, if the MCP bit in the DSR is set, a non-maskable interrupt is issued.	0 = Maskable interrupts disabled (default). 1 = Maskable interrupts enabled.
DMF[2:0] Bits 2–0	**DSP-to-MCU Flags—**General-purpose flag bits that are reflected on the MCU side in the MF[2:0] bits in the MSR.	

DSR DSP-Side Status Register **X:$FF8B**

	BIT 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT 0
	DTE0	DTE1	DRF0	DRF1	DGIR0	DGIR1	DTIC	MCP	DWSC	MPM1	MPM0	DEP		DF[2:0]		
RESET	1	1	0	0	0	0	0	0	0	—	—	0	0	—	—	—

Table 5-18. DSR Description

Name	Type[1]	Description	Settings
DTE0 Bit 15	R	**DSP Transmit Register 0 Empty—**Indicates if the MCU has read the most recent transmission to MRR0. This bit is subject to DSP pipeline restrictions (See Table 5-6 on page 5-15.)	0 = Last transmission to MRR0 has not been read 1 = Last transmission to MRR0 has been read (default).
DTE1 Bit 14	R	**DSP Transmit Register 1 Empty—**Indicates if the MCU has read the most recent transmission to MRR1. This bit is subject to DSP pipeline restrictions. (See Table 5-6 on page 5-15.)	0 = Last transmission to MRR1 has not been read 1 = Last transmission to MRR1 has been read (default).
DRF0 Bit 13	R	**DSP Receive Register 0 Full—**Set when the MCU writes to MTR0, indicating to the DSP that the reflected data is available in DRR0. DRF0 is cleared when the DSP reads DRR0.	0 = Latest DRR0 data has been read (default). 1 = New data in DRR0.
DRF1 Bit 12	R	**DSP Receive Register 1 Full—**Set when the MCU writes to MTR1, indicating to the DSP that the reflected data is available in DRR1. DRF1 is cleared when the DSP reads DRR1.	0 = Latest DRR1 data has been read (default). 1 = New data in DRR1.
DGIR0 Bit 11	R/1S	**DSP General Interrupt Request 0—**Setting this bit generates an interrupt request to the MCU if the MGIE0 bit in the MCR is set. It is reflected in the MGIP0 bit in the MSR. It is cleared when the MCU clears MGIP0, indicating to the DSP that the MCU has serviced the interrupt.	0 = No interrupt request 0 (default). 1 = DSP has issued interrupt request 0.
DGIR1 Bit 10	R/1S	**DSP General Interrupt Request 1—**Setting this bit generates an interrupt request to the MCU if the MGIE1 bit in the MCR is set. It is reflected in the MGIP1 bit in the MSR. It is cleared when the MCU clears MGIP1, indicating to the DSP that the MCU has serviced the interrupt.	0 = No interrupt request 1 (default). 1 = DSP has issued interrupt request 1.
DTIC Bit 9	1S	**DSP Protocol Timer Interrupt Clear—**Used by the Protocol Timer DSP interrupt ($\overline{\text{IRQD}}$) service routine to clear the interrupt. Writing “1” to this bit clears the MTIR bit in the MSR, thus deasserting $\overline{\text{IRQD}}$ (and $\overline{\text{IRQA}}$, which is wire-or’d to $\overline{\text{IRQD}}$) and enabling MTIR to receive another interrupt. DTIC always reads zero.	
MCP Bit 8	R	**MCU Command Pending—**Set when the MC bit in the MCVR is set (page 5-18); cleared when the interrupt generated by setting MC is serviced.	0 = No outstanding DSP command interrupt (default). 1 = DSP command interrupt has been issued and has not been serviced.

Table 5-18. DSR Description (Continued)

Name	Type[1]	Description	Settings
DWSC Bit 7	1S	**DSP Wake from STOP and Interrupt Clear—**Used by the MDI Wake from STOP and general interrupt ($\overline{\text{IRQC}}$) service routine to clear the interrupt. Writing “1” to this bit clears the DWS bit in the MSR, thus de-asserting $\overline{\text{IRQC}}$ (and $\overline{\text{IRQA}}$) and enabling DWS to receive another interrupt.	
MPM[1:0] Bits 6–5	R	**MCU Power Mode—**Reflect the MCU power mode.	00 = STOP 01 = WAIT 10 = DOZE 11 = Normal
DEP Bit 4	R	**DSP-Side Event Pending—**Set when the DSP sends an event update request to the MCU side. Cleared when the event update acknowledge has been received. An “event” is any hardware message that should be reflected in the MSR on the MCU-side (e.g., “transmit register 0 written”). Software should poll DEP until it is cleared before entering STOP mode. Reading the DSR to check the DEP bit should be the last MDI access before entering STOP, otherwise the DEP can be set as a result of that additional action. Allow three NOPs (or their equivalent timing) after an instruction that sets an event before DEP is updated to accommodate pipeline effects. Proper verification of DEP value can prevent loss of shared memory accesses and failure to inform the MCU side of events while the DSP is in STOP mode.	0 = Last event update request to MCU has been acknowledged (default). 1 = Event update request to MCU pending.
DF[2:0] Bits 2–0	R	**MCU Flags—**Reflect the MDF[2:0] bits in the MSR.	0 = Corresponding MDF bit cleared. 1 = Corresponding MDF bit set.

[1] R = Read only.
1S = Write 1 only (write with 0 ignored).
R/1S Read; write 1 only (write with 0 ingored)

DTR1 DSP Transmit Register 1 **X:$FF8C**

	BIT 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT 0
	Transmitted data from MCU to DSP															
RESET	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Table 5-19. DTR1 Description

DTR1 is a 16-bit write-only register. Data written to DTR1 is reflected on the MCU side in MRR1. DTR1 and MRR1 are not double buffered. Writing to DTR1 overwrites the data in MRR1, clears the DTE1 bit in the DSR, and sets the MRF1 bit in the MSR. It can also trigger a receive interrupt on the MCU side if the MRIE1 bit in the MCR is set.

DTR0 DSP Transmit Register 0 **X:$FF8D**

	BIT 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT 0
	Transmitted data from MCU to DSP															
RESET	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Table 5-20. DTR0 Description

DTR0 is a 16-bit write-only register. Data written to DTR0 is reflected on the MCU side in MRR0. DTR0 and MRR0 are not double buffered. Writing to DTR0 overwrites the data in MRR0, clears the DTE0 bit in the DSR, and sets the MRF0 bit in the MSR. It can also trigger a receive interrupt on the MCU side if the MRIE0 bit in the MCR is set.

DRR1 DSP Receive Register 1 **X:$FF8E**

	BIT 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT 0
	Transmitted data from DSP to DSP															
RESET	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Table 5-21. DRR1 Description

DRR1 is a 16-bit read-only register that reflects the data written on the MCU side to MTR1. Reading DRR1 clears the DRF1 bit in the DSR, sets the DTE1 bit in the MSR, and can trigger a transmit interrupt on the MCU side if the MTIE1 bit in the MCR is set.

DRR0 DSP Receive Register 0 **X:$FF8F**

	BIT 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT 0
	Transmitted data from DSP to DSP															
RESET	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Table 5-22. DRR0 Description

DRR0 is a 16-bit read-only register that reflects the data written on the MCU side to MTR0. Reading DRR0 clears the DRF0 bit in the DSR, sets the DTE0 bit in the MSR, and can trigger a transmit interrupt on the MCU side if the MTIE0 bit in the MCR is set.

Chapter 6
External Interface Module

The EIM provides signals and logic to connect memory and other external devices to the DSP56652. EIM features include the following:

- Twenty-two-bit external address bus and 16-bit external data bus
- Six chip selects for external devices, each of which provides
  - A 4-Mbyte range
  - Programmable wait state generator
  - Selectable protection
  - Programmable data port size
  - General output signal if not used as a chip select
- External or internal boot ROM device selection
- Bus watchdog counter for all bus cycles
- External monitoring of internal bus cycles

Figure 6-1 shows a block diagram of the EIM.

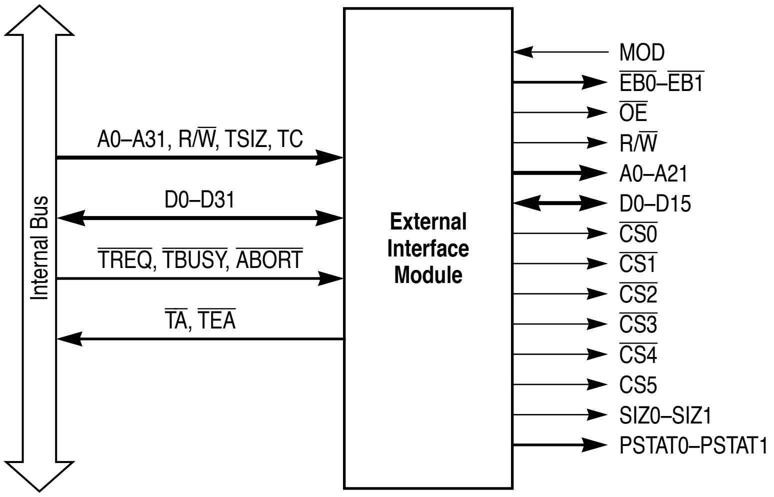


Figure 6-1. EIM Block Diagram

Figure 6-2 shows an example of an EIM interface to memory and peripherals.

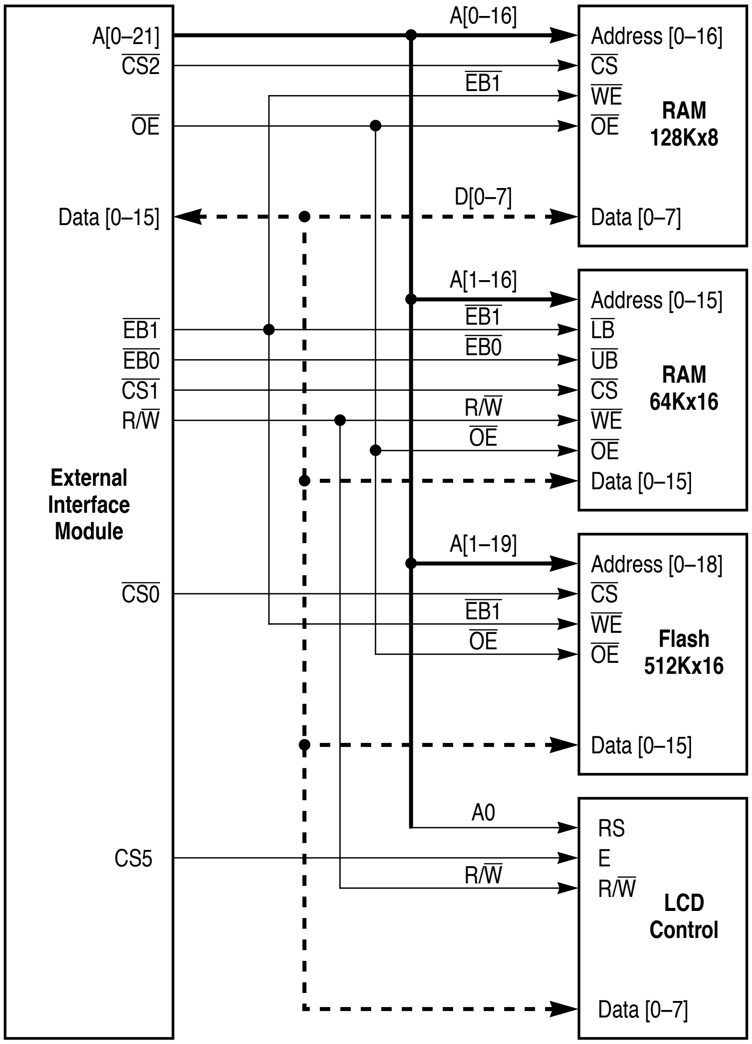


Figure 6-2. Example EIM Interface to Memory and Peripherals

6.1 EIM Signals

The EIM signal descriptions in Section 2.4, “External Interface Module,” are repeated and expanded in Table 6-1 for convenience.

Table 6-1. EIM Signal Description

Signal Name	Type	Reset State	Signal Description
A0–A21	Output	Driven low	**Address bus—**These signals specify the address for external memory accesses. If there is no external bus activity, A0–A21 remain at their previous values to reduce power consumption.
D0–D15	Input/ Output	Input	**Data bus**—These signals provide the bidirectional data bus for external memory accesses. They remain in their previous logic state when there is no external bus activity to reduce power consumption.
R/$\overline{\text{W}}$	Output	Driven high	**Read/write**—This signal indicates the bus access type. A high signal indicates a bus read. A low signal indicates a write to the bus. This signal can also be used as a memory write enable ($\overline{\text{WE}}$) signal. When accessing a peripheral chip, the signal acts as a read/write.
$\overline{\text{EB0}}$	Output	Driven high	**Enable byte 0**—When driven low, this signal indicates access to data byte 0 (D8–D15) during a read or write cycle. This pin may also act as a write byte enable, if so programmed.
$\overline{\text{EB1}}$	Output	Driven high	**Enable byte 1**—When driven low, this signal indicates access to data byte 1 (D0–D7) during a read or write cycle. This pin may also act as a write byte enable, if so programmed.
$\overline{\text{OE}}$	Output	Driven high	**Output Enable—**When driven low, this signal indicates that the current bus access is a read cycle and enables slave devices to drive the data bus with a read.
MOD	Input	Input	**Mode Select**—This signal selects the MCU boot mode during hardware reset. It should be driven at least four CKIL clock cycles before $\overline{\text{RESET\_OUT}}$ is deasserted. • MOD driven high—MCU fetches the first word from internal MCU ROM. • MOD driven low—MCU fetches the first word from the external memory ($\overline{\text{CS0}}$).
$\overline{\text{CS0}}$	Output	Chip-driven	**Chip select 0—**This signal is asserted low based on the decode of the internal address bus bits A[31:24] and the state of the MOD pin at reset. It is often used as the external flash memory chip select. After reset, CS0 access has a default of 15 wait states and a port size of 16 bits.
$\overline{\text{CS1}}$–$\overline{\text{CS4}}$	Output	Driven high	**Chip selects 1–4—**These signals are asserted low based on the decode of the internal address bus bits A[31:24] of the access address. When not configured as chip selects, these signals become general purpose outputs (GPOs). After reset, these signals are GPOs that are driven high.
CS5	Output	Driven low	**Chip select 5—**This signal is asserted high based on the decode of the internal address bus bits A[31:24] of the access address. When not configured as a chip select, this signal functions as a GPO. After reset, this signal is a GPO that is driven low.

6.2 Chip Select Address Ranges

Each of the six chip select signals corresponds to a 16-Mbyte block in the MCU address space. Note that only 22 address lines are available, so only the first four Mbytes in each chip select space can be addressed. An access above the 4-Mbyte limit modulo-wraps back into the addressable space and is not recommended. Table 6-2 lists the allocated and addressable ranges for each chip select.

Table 6-2. Chip Select Address Range

Chip Select	A[31:24]	Allocated Memory Space (16 Mbytes)	Addressable Range (4 Mbytes)
$\overline{\text{CS0}}$	01000000	\$4000_0000–\$40FF_FFFF	\$4000_0000–\$403F_FFFF
$\overline{\text{CS1}}$	01000001	\$4100_0000–\$41FF_FFFF	\$4100_0000–\$413F_FFFF
$\overline{\text{CS2}}$	01000010	\$4200_0000–\$42FF_FFFF	\$4200_0000–\$423F_FFFF
$\overline{\text{CS3}}$	01000011	\$4300_0000–\$43FF_FFFF	\$4300_0000–\$433F_FFFF
$\overline{\text{CS4}}$	01000100	\$4400_0000–\$44FF_FFFF	\$4400_0000–\$443F_FFFF
CS5	01000101	\$4500_0000–\$45FF_FFFF	\$4500_0000–\$453F_FFFF

6.3 EIM Features

This section discusses the following features of the EIM:

- Configurable bus sizing
- External boot ROM control
- Bus watchdog operation
- Error condition reporting
- External display of internal bus activity
- Emulation Port
- General-purpose outputs

6.3.1 Configurable Bus Sizing

The EIM supports byte, halfword, and word operands, allowing access to 8- and 16-bit ports. It does not support misaligned transfers. The port size for each chip select is programmed through the DSZ[1:0] bits in the associated CS control register. In addition, the portion of the data bus used for transfer to or from an 8-bit port is programmable via the same bits. An 8-bit port can reside on external data bus bits D[15:8] or D[7:0]. Connecting 8-bit devices to D[15:8] reduces the load on the lower data lines.

A word access to or from an 8-bit port requires four bus cycles to complete. A word access to or from a 16-bit port requires two bus cycles to complete. A halfword access to or from an 8-bit port requires two bus cycles to complete. In a multi-cycle transfer, the lower two address bits (A[1:0]) are incremented appropriately.

The EIM contains a data multiplexer that routes the four bytes of the MCU interface data bus to their required positions for proper interface to memory and peripherals.

Table 6-3 summarizes the possible transfer sizes, alignments, and port widths as well as the SIZ1–SIZ0 signals, A1–A0 signals, and DSZ[1:0] bits used to generate them.

6.3.2 External Boot ROM Control

The MOD input signal is used to specify the location of the boot ROM device during hardware reset. If an external boot ROM is used instead of the internal ROM, the $\overline{\text{CS0}}$ output can be used to select the external ROM coming out of reset.

If MOD is driven low at least four CKIL clock cycles before $\overline{\text{RESET\_OUT}}$ deassertion, the internal MCU ROM is disabled and $\overline{\text{CS0}}$ is asserted for the first MCU cycle. The MCU fetches the reset vector from address $0 of the $\overline{\text{CS0}}$ memory space, which is located at the absolute address $4000_0000 in the MCU address space. The internal MCU ROM is disabled for the first MCU cycle only and is available for subsequent accesses. Out of Reset, $\overline{\text{CS0}}$ is configured for 15 wait states and a 16-bit port size. If MOD is driven high at least four CKIL clock cycles before $\overline{\text{RESET\_OUT}}$ deassertion, the internal ROM is enabled and the MCU fetches the reset vector from internal ROM at address $0000_0000.

6.3.3 Bus Watchdog Operation

The EIM contains a bus watchdog timer that monitors the length of all request accesses from the MCU. If an access does not terminate (i.e., the bus watchdog timer does not receive an internal Transfer Acknowledge ($\overline{\text{TA}}$) signal or Transfer Error Acknowledge ($\overline{\text{TEA}}$) signal) within 128 clock cycles of being initiated, the bus watchdog timer expires and forces the access to be terminated by negating the Chip Select output and any control signals that were asserted during the access. The bus watchdog timer then asserts a $\overline{\text{TEA}}$ signal back to the MCU, resulting in an access error exception. The bus watchdog timer is automatically reset after the termination of each access. If for some reason an internal MCU peripheral does not terminate its access to the MCU, or if the MCU accesses an unmapped location, the bus watchdog times out and prevents the MCU from locking up.

Table 6-3. Interface Requirements for Read and Write Cycles

Transfer Size	Signal Encoding				Port Width	Active Interface Bus Sections[1]			
	SIZ1	SIZ0	A1	A0	DSZ[1:0]	Internal D[31:24]	Internal D[23:16]	Internal D[15:8]	Internal D[7:0]
Byte	0	1	0	0	00	D[15:8]	—	—	—
					01	D[7:0]	—	—	—
					10	D[15:8]	—	—	—
			0	1	00	—	D[15:8]	—	—
					01	—	D[7:0]	—	—
					10	—	D[7:0]	—	—
			1	0	00	—	—	D[15:8]	—
					01	—	—	D[7:0]	—
					10	—	—	D[15:8]	—
			1	1	00	—	—	—	D[15:8]
					01	—	—	—	D[7:0]
					10	—	—	—	D[7:0]
Halfword	1	0	0	x	00	D[15:8]	D[15:8]	—	—
					01	D[7:0]	D[7:0]	—	—
					10	D[15:8]	D[7:0]	—	—
			1	x	00	—	—	D[15:8]	D[15:8]
					01	—	—	D[7:0]	D[7:0]
					10	—	—	D[15:8]	D[7:0]
Word	0	0	x	x	00	D[15:8]	D[15:8]	D[15:8]	D[15:8]
					01	D[7:0]	D[7:0]	D[7:0]	D[7:0]
					10	D[15:8]	D[7:0]	D[15:8]	D[7:0]

[1] Bytes labeled with a dash are not required. They are ignored on read transfers and driven with undefined data on write transfers.

6.3.4 Error Conditions

The following conditions cause a Transfer Error Acknowledge ($\overline{\text{TEA}}$) to be asserted to the MCU:

- An access to a disabled chip-select (i.e., an access to a mapped chip-select address space where the CSEN bit in the corresponding CS control register is clear).
- A write access to a write-protected chip-select address space (i.e., the WP bit in the corresponding CS control register is set).
- A user access to a supervisor-protected chip-select address space (i.e., the SP bit in the corresponding CS control register is set).
- A bus watchdog time-out when an access does not terminate within 128 clocks of being initiated.
- A user access to a supervisor-protected internal ROM, RAM, or peripheral space (i.e., the corresponding SP bit in the EIM Configuration register is set).

6.3.5 Displaying the Internal Bus (Show Cycles)

Although the MCU can transfer data between internal modules without using the external bus, it may be useful to display an internal bus cycle on the external bus for debugging purposes. Such external bus cycles, called show cycles, are enabled by the SHEN[1:0] bits in the EIM Configuration Register (EIMCR).

When show cycles are enabled, the EIM drives the internal address bus A[21:0] onto the external address bus pins A21–A0. In addition, the internal data bus D[31:16] or D[15:0] is driven onto the external data bus pins D15–D0 according to the HDB bit in the EIMCR.

6.3.6 Programmable Output Generation

Any chip select signal except $\overline{\text{CS0}}$ can be used as general-purpose output by clearing the CSEN bit in the corresponding CS control register. (When the CSEN bit in the CS0 register is cleared, $\overline{\text{CS0}}$ is inactive.)

6.3.7 Emulation Port

The DSP56652 provides a six-pin Emulation Port for debugging to provide information about the data size and pipeline status of the current bus cycle. The SIZ[1:0] pins indicate the data size using the encoding shown in Table 6-4. The PSTAT[3:0] pins provide pipeline information as shown in Table 6-5. The Emulation Port is enabled by the EPEN bit in the EIMCR and serve as GPIO pins if the port is not enabled.

Table 6-4. SIZ[1:0] Encoding

SIZ1	SIZ0	Transfer Size
0	0	Word (32 bits)
0	1	Byte (8 bits)
1	0	Halfword (16 bits)
1	1	Reserved

Table 6-5. PSTAT[3:0] Encoding

PSTAT3	PSTAT2	PSTAT1	PSTAT0	Internal Processor Status
0	0	0	0	Execution Stalled
0	0	0	1	Execution Stalled
0	0	1	0	Execute Exception
0	0	1	1	Reserved
0	1	0	0	Processor in Stop, Wait, or Doze mode
0	1	0	1	Execution Stalled
0	1	1	0	Processor in Debug Mode
0	1	1	1	Reserved
1	0	0	0	Launch instruction[1]
1	0	0	1	Launch ldm, stm, ldq, stq
1	0	1	0	Launch Hardware Accelerator instruction
1	0	1	1	Launch lrw
1	1	0	0	Launch change of Program Flow instruction
1	1	0	1	Launch rte or rfi
1	1	1	0	Reserved
1	1	1	1	Launch jmpi or jsri

1. Except rte, rfi, ldm, stm, ldq, stq, lrw, hardware accelerator, or change of flow instructions

6.4 EIM Registers

CSCR0	Chip Select 0 Control Register	**$0020_1000**
CSCR1	Chip Select 1 Control Register	**$0020_1004**
CSCR2	Chip Select 2 Control Register	**$0020_1008**
CSCR3	Chip Select 3 Control Register	**$0020_100C**
CSCR4	Chip Select 4 Control Register	**$0020_1010**
CSCR5	Chip Select 5 Control Register	**$0020_1014**

	31–16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT 0
		WSC[3:0]				WWS	EDC	CSA	OEA	WEN	EBC	DSZ[1:0]		SP	WP	PA	CSEN
RESET	CS0	1	1	1	1	1	0	0	0	0	1	1	0	0	0		1
	CS1	—	—	—	—	—	—	—	—	—	—	—	—	—	—	1	0
	CS2	—	—	—	—	—	—	—	—	—	—	—	—	—	—	1	0
	CS3	—	—	—	—	—	—	—	—	—	—	—	—	—	—	1	0
	CS4	—	—	—	—	—	—	—	—	—	—	—	—	—	—	1	0
	CS5	—	—	—	—	—	—	—	—	—	—	—	—	—	—	0	0

Table 6-6. CSCRn Description

<table>
<tr><th>Name</th><th>Description</th><th>Settings</th></tr>
<tr><td>WSC[3:0]
Bits15–12</td><td>Wait State Control Bits—Determine the number of wait states for an access to the external device connected to the Chip Select. When WWS is cleared, setting WSC[3:0] = 0000 results in one-clock transfers, WSC[3:0] = 0001 results in two-clock transfers, and WSC[3:0] = 1111 results in 16-clock transfers. When WSC[3:0] = 0000, the WEN, OEA, and CSA bits are ignored.</td><td>
<table>
<tr><th rowspan="3">WSC [3:0]</th><th colspan="4">Number of Wait States</th></tr>
<tr><th colspan="2">WWS = 0</th><th colspan="2">WWS = 1</th></tr>
<tr><th>Read</th><th>Write</th><th>Read</th><th>Write</th></tr>
<tr><td>0000</td><td>0</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td>0001</td><td>1</td><td>1</td><td>1</td><td>2</td></tr>
<tr><td>0010</td><td>2</td><td>2</td><td>2</td><td>3</td></tr>
<tr><td>:</td><td>:</td><td>:</td><td>:</td><td>:</td></tr>
<tr><td>1101</td><td>13</td><td>13</td><td>13</td><td>14</td></tr>
<tr><td>1110</td><td>14</td><td>14</td><td>14</td><td>15</td></tr>
<tr><td>1111</td><td>15</td><td>15</td><td>15</td><td>15</td></tr>
</table>
</td></tr>
</table>

Table 6-6. CSCRn Description (Continued)

Name	Description	Settings
WWS Bit 11	**Write Wait State**—Specifies whether an additional wait state is inserted for write cycles. When WWS is set, an additional wait state is inserted for write cycles (unless WSC[3:0] = 1111, which results in a 16- clock cycle write time, regardless of the WWS bit). Read cycles are not affected. When this bit is cleared, reads and writes are of the same length. Setting this bit is useful for writing to slower memories (such as Flash memories) that require additional data setup time.	0 = Reads and writes are same length. 1 = Writes have an additional wait state (except when WSC[3:0] = 1111).
EDC Bit 10	**Extra Dead Cycle**—When set, inserts an idle cycle after a read cycle for back-to-back external transfers, unless the next cycle is a read cycle to the same $\overline{\text{CS}}$ bank to eliminate data bus contention. This is useful for slow memory and peripherals that have long $\overline{\text{CS}}$ or $\overline{\text{OE}}$ to output data tri-state times.	0 = Back-to-back external transfers occur normally. 1 = Extra idle cycle inserted in back-to-back external transfers unless the next cycle is a read cycle to the same $\overline{\text{CS}}$.
CSA Bit 9	**Chip Select Assert**—When CSA is set, Chip Select is asserted one clock cycle later during both read and write cycles, and an idle cycle is inserted between back-to-back external transfers. Useful for devices that require additional address setup time and address/data hold times. If WSC[3:0] = 0000, the CSA bit is ignored.	0 = Chip Select asserted normally (i.e., as early as possible); no idle cycle inserted. 1 = Chip Select asserted one cycle later; idle cycle inserted in back-to-back external transfers.
OEA Bit 8	**$\overline{\text{OE}}$ Assert**—When OEA is set, $\overline{\text{OE}}$ is asserted one half-clock later during a read to the CS's address space. Cycle length is not affected, and write cycles are not affected. If WSC[3:0] = 0000, OEA is ignored and $\overline{\text{OE}}$ is asserted for half a clock only. If EBC in the corresponding register is cleared, the $\overline{\text{EB0–1}}$ outputs are similarly affected.	0 = $\overline{\text{OE}}$ asserted normally (i.e., as early as possible). 1 = $\overline{\text{OE}}$ asserted one half cycle later during a read.
WEN Bit 7	**Write $\overline{\text{EB}}$ Negate**—When WEN is set, $\overline{\text{EB0–1}}$ are negated one half-clock earlier during a write to the CS's address space. Cycle length is not affected, and read cycles are not affected. If WSC[3:0] = 0000, WEN is ignored and is $\overline{\text{EB0–1}}$ are asserted for half a clock only. WEN is useful for meeting data hold time requirements for slow memories.	0 = $\overline{\text{EB0–1}}$ negated normally (i.e., as late as possible). 1 = $\overline{\text{EB0–1}}$ negated one half cycle earlier during a write.
EBC Bit 6	**Enable Byte Control**—When EBC is set, only write accesses assert the $\overline{\text{EB0–1}}$ outputs, thus configuring them as byte write enables. EBC should be set for accesses to dual x8 memories.	0 = $\overline{\text{EB0–1}}$ asserted for both reads and writes. 1 = $\overline{\text{EB0–1}}$ asserted for writes only.
DSZ[1:0] Bits 5–4	**Data Port Size**—These bits define the width of the device data port.	00 = 8-bit port on D[15:8] pins. 01 = 8-bit port on D[7:0] pins. 10 = 16-bit port on D[15:0] pins. 11 = Reserved.

Table 6-6. CSCRn Description (Continued)

Name	Description	Settings
SP Bit 3	**Supervisor Protect—**Prohibits User Mode accesses to the CS address space. When SP is set, a read or write to the CS space while in User Mode generates a $\overline{\text{TEA}}$ error and the CS signal is not asserted.	0 = User Mode access allowed. 1 = User Mode access prohibited.
WP Bit 2	**Write Protect—**Prohibits writes to the CS address space. When WP is set, a write attempt to the CS space generates a $\overline{\text{TEA}}$ error and the CS signal is not asserted.	0 = Writes allowed. 1 = Writes prohibited.
PA Bit 1	**Pin Assert—**Controls the Chip Select pin when it is operating as a general-purpose output (i.e., the CSEN bit is cleared). This bit is ignored if the CSEN bit is set. Note that Chip Select 0 does not have a PA bit.	0 = CS pin at logic low. 1 = CS pin at logic high.
CSEN Bit 0	**Chip Select Enable—**When CSEN is set, the CS pin is asserted during an access to its address space. When CSEN is cleared, an access to the CS address space generates a TEA error and the CS pin is not asserted.	0 = CS pin disabled. 1 = CS pin enabled.

EIMCR EIM Configuration Register **\$0020_1018**

	BIT 31	7	6	5	4	3	2	1	BIT 0
			EPEN	SPIPER	SPRAM	SPROM	HDB	SHEN1[:0]	
RESET	0		0	1	1	1	0	0	0

Table 6-7. EIMCR Description

Name	Description	Settings
EPEN Bit 6	**Emulation Port Enable—**Controls the functions of the Emulation Port pins, SIZ[1:0] and PSTAT[3:0].	0 = Pins function as GPIO (default). 1 = Emulation Port drives the pins with the MCU SIZ[1:0] and PSTAT[3:0] signals.
SPIPER Bit 5	**Supervisor Protect Internal Peripheral—**Prohibits User Mode access to all internal peripheral space. When SPIPER is set, a read or write to the internal peripheral space while in User Mode generates a $\overline{\text{TEA}}$ error. This bit does not affect CSCR0–5 or EIMCR, which can only be accessed in supervisor mode.	0 = User Mode access to internal peripherals allowed. 1 = User Mode access to internal peripherals prohibited (default).
SPRAM Bit 4	**Supervisor Protect Internal RAM—**Prohibits User Mode access to internal RAM. When SPRAM is set, a read or write to the internal RAM while in User Mode generates a $\overline{\text{TEA}}$ error.	0 = User Mode access to internal RAM allowed. 1 = User Mode access to internal RAM prohibited (default).
SPROM Bit 3	**Supervisor Protect Internal ROM—**Prohibits User Mode access to internal ROM. When SPROM is set, a read or write to the internal ROM while in User Mode generates a $\overline{\text{TEA}}$ error.	0 = User Mode access to internal ROM allowed. 1 = User Mode access to internal ROM prohibited (default).
HDB Bit 2	**High Data Bus—**selects the internal halfword to be placed on the external data bus during a Show Cycle. This bit is ignored when SHEN[1:0] are cleared.	0 = Lower halfword (D[15:0]) (default). 1 = Upper halfword (D[31:16]).
SHEN[1:0] Bits 1–0	**Show Cycle Enable—**These bits enable the internal buses to be reflected on the external buses during accesses to internal RAM, ROM, or peripherals. They can also delay internal termination to the MCU during idle cycles caused by EDC or CSA being set (page 6-10). This ensures that all internal transfers can be externally monitored, although this setting can impact performance.	00 = Show cycles disabled (default). 01 = Show cycles enabled. Internal termination to the MCU during idle cycles caused by EDC or CSA being set is not delayed, and internal transfers that occur during these EDA/CSA idle cycles will not be visible externally. 10 = Show cycles enabled. Internal termination to the MCU during idle cycles caused by EDC or CSA being set is delayed by one cycle. This ensures that all internal transfers can be externally monitored, at the expense of performance. 11 = Reserved.

EMDDR Emulation Port Data Direction Register **$0020_C800**

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT 0
											EMDD5 (PSTAT3)	EMDD4 (PSTAT2)	EMDD3 (PSTAT1)	EMDD2 (PSTAT0)	EMDD1 (SIZ1)	EMDD0 (SIZ0)
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 6-8. QDDR Description

Name	Description	Settings
EMDD[5:0] Bits 7–0	**Emulation Port Data Direction[5:0]—**determines whether each pin functions as an input or an output when the port functions as GPIO (Emulation Port is disabled).	0 = Input (default) 1 = Output

EMDR Emulation Port Data Register **$0020_C802**

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT 0
											EMD5 (PSTAT3)	EMD4 (PSTAT2)	EMD3 (PSTAT1)	EMD2 (PSTAT0)	EMD1 (SIZ1)	EMD0 (SIZ0)
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 6-9. QPDR Description

Name	Description
EMD[5:0] Bits 7–0	**Emulation Port GPIO Data [5:0]—**Each of these bits contains data for the corresponding Emulation Port pin if the port is configured as GPIO. Writes to EMDR are stored in an internal latch, and driven on any port pin configured as an output. Reads of this register return the value sensed on input pins and the latched data driven on outputs.

Chapter 7
Interrupts

This section describes both the MCU and DSP interrupt controllers, including the various interrupt and exception sources and how they are configured and prioritized. The Edge I/O port, which provides eight pins for external MCU interrupts, is also described.

7.1 MCU Interrupt Controller

The MCU interrupt controller combines the speed of a highly microcoded architecture with the flexibility of polling techniques commonly employed in RISC designs. The result is a centralized mechanism that permits polling and prioritizing of the 32 interrupt sources with minimal software overhead. This mechanism includes the following features:

- **Find-First-One instruction.** This instruction provides a fast mechanism to prioritize pending interrupt requests. It scans the contents of a register and reports the position of the most significant set bit.
- **Highest priority status.** Any interrupt can be configured as the highest priority, in which case it is assigned a vectored interrupt. Directly-vectored interrupts can be serviced with fewer instructions than autovectored interrupts, because polling to determine the interrupt's source is not required. For more information refer to the *M•CORE Reference Manual*.
- **Alternate register set.** The MCU provides an alternate register set for interrupts, including general registers, status register and program counter, eliminating the need to save program context to the stack.
- **Fast interrupts.** Critical interrupts can be processed using separate, dedicated program counter and status shadow registers not used by the other interrupts. Any source can be programmed to generate a normal or fast interrupt.
- **Individual enable bits.** Each interrupt source is individually configured.

7.1.1 Functional Overview

The MCU interrupt controller is comprised of six registers:

- ISR—The Interrupt Source Register reflects the current state of all interrupt sources within the chip.
- NIER—The Normal Interrupt Enable Register provides a centralized place to enable/disable interrupt requests and to assign interrupt sources to a normal interrupt.
- NIPR—The Normal Interrupt Pending Register reflects the current state of all pending non-masked normal interrupt requests.
- FIER—The Fast Interrupt Enable Register provides a centralized place to enable/disable interrupt requests and to assign interrupt sources to a fast interrupt.
- FIPR—The Fast Interrupt Pending register reflects the current state of all pending non-masked fast interrupt requests.
- ICR—The Interrupt Control Register selects the highest priority interrupt and its vector.

Figure 7-1 is a block diagram of the MCU interrupt controller.

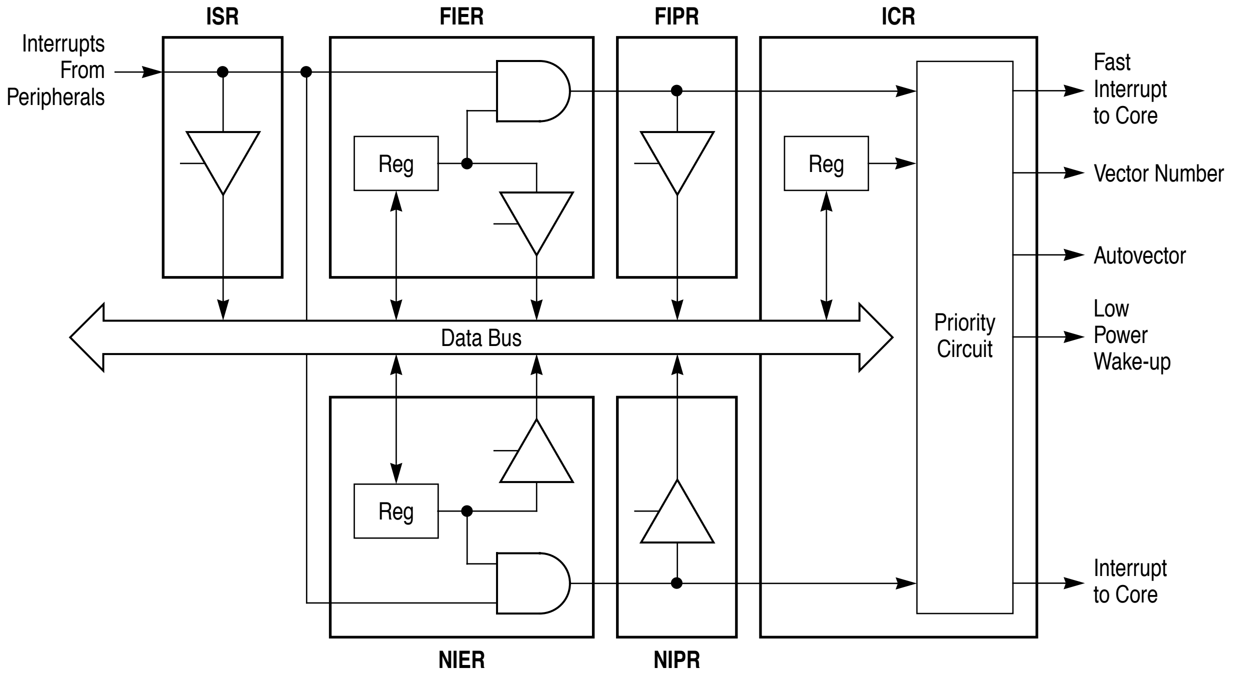


Figure 7-1. MCU Interrupt Controller

7.1.2 Exception Priority

The MCU core imposes the following priority (from highest to lowest) among the various exceptions:

- Hardware Reset

- Software Reset
- Hardware Breakpoint
- Fast Interrupt
- Normal Interrupt
- Instruction Generated Exceptions
- Trace

The interrupt controller registers prioritize the peripheral interrupts by designating each request as either an autovectored normal interrupt, autovectored fast interrupt, or vectored fast interrupt. Figure 7-2 illustrates the priority mechanism in flowchart format.

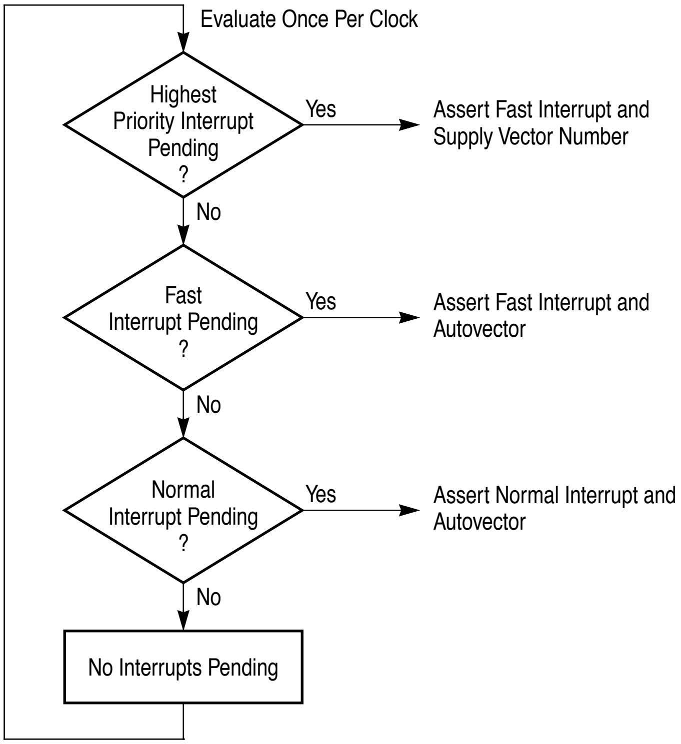


Figure 7-2. Hardware Priority Flowchart

7.1.3 Enabling MCU Interrupt Sources

Three steps are required to enable MCU interrupt sources:

1. Assign each interrupt to either normal or fast processing, and set the appropriate bits in the NIER or FIER.

Each interrupt source can be assigned to either of two interrupt request inputs, normal or fast. Fast requests are serviced before normal requests; there is no difference in latency.

The choice of interrupt request for each source depends on several factors driven by the end application, including:

- Rate of service requests
- Latency requirements
- Access to the alternate register bank
- Length of service routine
- Total number of interrupt sources in the system

Each interrupt source is enabled as a normal or fast interrupt by setting the appropriate bit in either the NIER or FIER. The enable bit should not be set in both registers simultaneously or both a normal and fast interrupt request will be generated.

2. Enable interrupts in the core by setting the following bits in the M•CORE Program Status Register:
   - Exception Enable (EE)
   - Interrupt Enable (IE)
   - Fast Interrupt Enable (FE)

   Refer to the *M•CORE Reference Manual* for more information on this register.

Steps 1 and 2 are normally done once during system initialization.

3. For each source from which interrupts are to be used, program the appropriate peripheral registers to generate interrupt requests.

7.1.4 Interrupt Sources

Table 7-1 lists each MCU interrupt source, the ISR bit that indicates when the interrupt is asserted, and a page reference to the register that enables the interrupt. Several interrupt sources are logically ORed because there are more sources than there are inputs to the interrupt controller. In these cases, the peripheral's status register must be queried to determine the source of the interrupt within the peripheral.

Table 7-1. MCU Interrupt Sources

Interrupt Source	Remarks	ISR Bit Name & No.		Source(s)	Where Enabled	Page
MDI[1]	6 ORed	MDI	23	MCU Transmit Interrupt 0, 1 MCU Receive Interrupt 0, 1 MCU General Interrupt 0, 1	MCR	5-19

Table 7-1. MCU Interrupt Sources (Continued)

Interrupt Source	Remarks	ISR Bit Name & No.		Source(s)	Where Enabled	Page
Edge I/O port[1,2]	8 separate	INT7–INT0	12–5	INT7–INT0 Pin Asserted	—	
QSPI	4 ORed	QSPI	24	QSPI HALT Command QSPI Trigger Collision QSPI Queue Pointer Wraparound	SPCR	8-13
				End of Transfer	QSPI Control RAM	8-22
PIT	1 separate	PIT	16	Periodic Interrupt Timer = 0	PITCSR	9-3
GPT	8 ORed	TPW	17	PWM Count Rollover GP Timer Count Overflow PWM Output Compare Input Capture 1, 2, 4 Output Compare 1, 3	TPWIR	9-16
Protocol Timer	3 separate + 5 ORed	PT2–PT0	28–26	PT Events mcu_int 2, 1, 0	PTIER	10-18
		PTM	25	PT Error PT HALT Command PT Reference Slot Counter = 0 PT Channel Frame Counter = 0 PT Channel Time Interval Counter = 0		
UART	2 separate + 2 ORed	URX	31	UART Receiver Ready	UCR1	11-11
		UTX	29	UART Transmitter Ready UART Transmitter Empty		
		URTS	13	$\overline{\text{RTS}}$ Pin State Change		
SmartCard	1 separate + 4 ORed	SMPC	30	SIM Sense Change	SCPIER	12-13
		SCP	22	SCP Transmit Complete SCP Receive FIFO Not Empty SCP Receive FIFO Full SCP Receive Error		
Keypad Interface	1 separate	KPD	14	KPD Key Closure	KPCR	13-5
Software	3 separate	S2–S0	2–0	Software Interrupts 2, 1, 0	—	

1. The MDI and Edge I/Ointerrupts are asynchronous. All other interrupts are synchronous.
2. The Edge I/O interrupts can be edge- or level-sensitive. All other interrupts are level-sensitive only.

7.1.5 MCU Interrupt Registers

Note: All Interrupt Controller registers require full 32-bit accesses.

ISR Interrupt Source Register **$0020_0000**

	BIT 31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	BIT 16
	URX	SMPC	UTX	PT2	PT1	PT0	PTM	QSPI	MDI	SCP	▨	▨	▨	▨	TPW	PIT
RESET[1]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	BIT 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT 0
	▨	KPD	URTS	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0	▨	▨	S2	S1	S0
RESET[1]	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

1. The state of each defined bit out of reset is determined by the interrupt request input of the associated peripheral; normally, the request is inactive.

The ISR is a read-only that reflects the status of all interrupt request inputs to the interrupt controller. The requests are synchronized so that reading the ISR always returns a stable value. All unused bits always read as 0, except for S[2:0], which always read as 1. Writes to this register have no effect.

Table 7-2. ISR Description

Name	Bit(s)	Interrupt Source	Setting
URX	31	UART Receiver Ready	0 = No interrupt request. 1 = Interrupt request pending.
SMPC	30	SIM Position Change	
UTX	29	UART Transmitter (2 ORed)	
PT2–0	28–26	Protocol Timer 2–0	
PTM	25	Protocol Timer (5 ORed)	
QSPI	24	QSPI (4 ORed)	
MDI	23	MDI (6 ORed)	
SCP	22	SCP (3 ORed)	
TPW	17	Timer/PWM (8 ORed)	
PIT	16	PIT	
KPD	14	Keypad Interface	
URTS	13	UART RTS	
INT7–0	12–5	External Interrupt 7–0	
S2–0	2–0	Software Interrupt 2–0	

NIER Normal Interrupt Enable Register **$0020_0004**

	BIT 31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	BIT 16
	EURX	ESMPC	EUTX	EPT2	EPT1	EPT0	EPTM	EQSPI	EMDI	ESCP					ETPW	EPIT
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	BIT 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT 0
		EKPD	EURTS	EINT7	EINT6	EINT5	EINT4	EINT3	EINT2	EINT1	EINT0			ES2	ES1	ES0
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

FIER Fast Interrupt Enable Register **$0020_0008**

	BIT 31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	BIT 16
	EFURX	EFSMPC	EFUTX	EFPT2	EFPT1	EFPT0	EFPTM	EFQSPI	EFMDI	EFSCP					EFTPW	EFPIT
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	BIT 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT 0
		EFKPD	EFURTS	EFINT7	EFINT6	EFINT5	EFINT4	EFINT3	EFINT2	EFINT1	EFINT0			NES2	NES1	NES0
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The NIER is used to enable pending interrupt requests to the core. Each defined bit in this register corresponds to an MCU interrupt source. If an interrupt is asserted and the corresponding NIER bit is set, the interrupt controller asserts a normal interrupt request to the core. If the corresponding NIER bit is cleared (i.e., if the interrupt is masked), the interrupt is not passed to the core and does not affect the high priority interrupt circuit. All interrupts are masked out of reset.

Register bits corresponding to unused interrupts may be read and written but have no affect on interrupt controller operation. Only word writes will update the NIER. Byte or half-word writes will terminate normally, but will not update the register.

The FIER works identically to the NIER, except that a fast interrupt is generated for a given request rather than a normal interrupt. Care should be taken to avoid setting the same bit position in both registers or both a normal and fast interrupt will be generated.

Table 7-3. NIER/FIER Description

Name		Bit(s)	Interrupt	Setting
NIER	**FIER**			
EURX	**EFURX**	31	UART Receiver Ready	0 = Interrupt source masked. 1 = Interrupt source enabled.
ESMPC	**EFSMPC**	30	SCP Position Change	
EUTX	**EFUTX**	29	UART Transmitter	
EPT2–0	**EFPT2–0**	28–26	Protocol Timer 2–0	
EPTM	**EFPTM**	25	Protocol Timer Interrupts	
EQSPI	**EFQSPI**	24	QSPI	
EMDI	**EFMDI**	23	MDI	
ESCP	**EFSCP**	22	SCP RxD, TxD, or Error	
ETPW	**EFTPW**	17	Timer/PWM	
EPIT	**EFPIT**	16	PIT	
EKPD	**EFKPD**	14	Keypad Interface	
EURTS	**EFURTS**	13	UART RTS	
EINT7–0	**EFINT7–0**	12–5	External Interrupt 7–0	
ES2–0[1]	**EFS2–0**[1]	2–0	Software Interrupts	

[1] Setting any of the software interrupt enable bits (ES2–0, NES2–0) immediately generates an interrupt to the MCU.

NIPR Normal Interrupt Pending Register **$0020_000C**

	BIT 31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	BIT 16
	NURX	NSMPC	NUTX	NPT2	NPT1	NPT0	NPTM	NQSPI	NMDI	NSCP					NTPW	NPIT
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	BIT 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT 0
		NKPD	NURTS	NINT7	NINT6	NINT5	NINT4	NINT3	NINT2	NINT1	NINT0			NS2	NS1	NS0
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The NIPR is used to monitor impending normal interrupts. Writes to this register are ignored. All unused bits always read as 0, except for bits 2–0, which always read as 1.

FIPR Fast Interrupt Pending Register **$0020_0010**

	BIT 31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	BIT 16
	FURX	FSMPC	FUTX	FPT2	FPT1	FPT0	FPTM	FQSPI	FMDI	FSCP					FTPW	FPIT
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	BIT 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT 0
		FKPD	FURTS	FINT7	FINT6	FINT5	FINT4	FINT3	FINT2	FINT1	FINT0			FS2	FS1	FS0
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The FIPR works in the same fashion as the NIPR to monitor fast interrupts.

Table 7-4. NIPR and FIPR Description

Name		Bit(s)	Interrupt	Setting
NIPR	FIPR			
NURX	FURX	31	UART Receiver Ready	0 = No interrupt request. 1 = Interrupt request pending.
NSMPC	FSMPC	30	SIM Position Change	
NUTX	FUTX	29	UART Transmitter	
NPT2–0	FPT2–0	28–26	Protocol Timer 2–0	
NPTM	FPTM	25	Protocol Timer Interrupts	
NQSPI	FQSPI	24	QSPI	
NMDI	FMDI	23	MDI	
NSCP	FSCP	22	SCP RxD, TxD, or Error	
NTPW	FTPW	17	Timer/PWM	
NPIT	FPIT	16	PIT	
NKPD	FKPD	14	Keypad Interface	
NURTS	FURTS	13	UART RTS	
NINT7–0	FINT7–0	12–5	External Interrupt 7–0	
NS2–0	FS2–0	2–0	Software Interrupt 2–0	

ICR Interrupt Control Register **$0020_0014**

	BIT 31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	BIT 16
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	BIT 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT 0
	EN				Source Number					Vector Number						
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The ICR selects a fast interrupt source to elevate to the highest priority, and specifies the vector to be used to service the interrupt. Only word writes will update the ICR. Byte or half-word writes will terminate normally, but will not update the register.

Table 7-5. ICR Description

Name	Description	Settings
EN Bit 15	**Enable Highest Priority Interrupt Hardware**	0 = Priority hardware disabled (default). 1 = Priority hardware enabled.
Bits 11–7	**Source Number—**Bit position of source to raise to the highest priority.	
Bits 6–0	**Vector Number—**Vector number to supply when highest priority interrupt is pending. Refer to the M•CORE Reference Manual for the appropriate vector number.	

7.2 DSP Interrupt Controller

The interrupt controller on the DSP side of the DSP56652 is based on the 56600 core. Its operation is described in Section 7.3 of the *DSP56600 Family Manual*.

7.2.1 DSP Interrupt Sources

Table 7-6 on page 7-11 lists all of the DSP interrupt sources according to their interrupt vectors. The vectors are offsets from the program address written to the Vector Base Address (VBA) register in the program control unit.

If more than one interrupt request is pending when an instruction is executed, the interrupt source with the highest priority level is serviced first. When multiple interrupt requests having the same IPL are pending, a second fixed-priority structure within that IPL determines which interrupt source is serviced. Table 7-7 shows the relative priority order of the DSP interrupts. Priority level 3 is the highest, and 0 the lowest. Level 3 vectors cannot change their priority level, but all other vectors can be assigned a level of 0, 1, or 2. The table lists these vectors in their relative priority if they are assigned the same priority level.

$\overline{\text{IRQA}}$–$\overline{\text{D}}$ are wired internally as shown in Figure 7-3. $\overline{\text{IRQA}}$ is the DSP wake from stop interrupt, and is wire-ORed to the other three interrupts because they are all intended to wake the DSP as well. $\overline{\text{IRQA}}$ should be disabled by clearing IPRC bits 10–9.

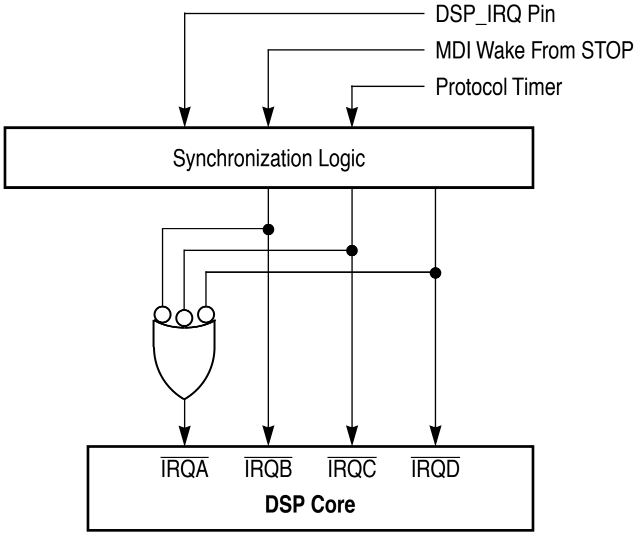


Figure 7-3. Internal $\overline{\text{IRQA}}$–$\overline{\text{D}}$ Connection

Table 7-6. DSP Interrupt Sources

VBA Offset	IPL	Interrupt Source	VBA Offset	IPL	Interrupt Source
\$00	3	Reserved	\$40	0 - 2	SAP Receive Data
\$02	3	Stack Error	\$42	0 - 2	SAP Receive Data With Overrun Error
\$04	3	Illegal Instruction	\$44	0 - 2	SAP Receive Last Slot
\$06	3	Debug Request Interrupt	\$46	0 - 2	SAP Transmit Data
\$08	3	Trap	\$48	0 - 2	SAP Transmit Data with Underrun Error
\$0A–\$0E	3	Reserved	\$4A	0 - 2	SAP Transmit Last Slot
\$10	0 -	$\overline{\text{IRQA}}$[1]	\$4C	0 - 2	SAP Timer Counter Rollover
\$12	0 -	$\overline{\text{IRQB}}$ ($\overline{\text{DSP\_IRQ}}$)	\$4E	0 - 2	Reserved
\$14	0 -	$\overline{\text{IRQC}}$ (MDI)	\$50	0 - 2	BBP Receive Data
\$16	0 -	$\overline{\text{IRQD}}$ (Protocol Timer)	\$52	0 - 2	BBP Receive Data With Overrun Error
\$18	0 -	Reserved	\$54	0 - 2	BBP Receive Last Slot
\$1A	0 -	Reserved	\$56	0 - 2	BBP Receive Frame Counter
\$1C	0 -	Reserved	\$58	0 - 2	BBP Transmit Data
\$1E	0 -	Reserved	\$5A	0 - 2	BBP Transmit Data with Underrun Error
\$20	0 -	Protocol Timer CVR0	\$5C	0 - 2	BBP Transmit Last Slot
\$22	0 -	Protocol Timer CVR1	\$5E	0 - 2	BBP Transmit Frame Counter

Table 7-6. DSP Interrupt Sources (Continued)

VBA Offset	IPL	Interrupt Source	VBA Offset	IPL	Interrupt Source
$24	0 -	Protocol Timer CVR2	$60	0 - 2 / 3	MDI MCU default command / MCU NMI[2]
$26	0 -	Protocol Timer CVR3	$62	0 - 2	MDI Receive 0
$28	0 -	Protocol Timer CVR4	$64	0 - 2	MDI Receive 1
$2A	0 -	Protocol Timer CVR5	$66	0 - 2	MDI Transmit 0
$2C	0 -	Protocol Timer CVR6	$68	0 - 2	MDI Transmit 1
$2E	0 -	Protocol Timer CVR7	$6A...$F	0 - 2	Reserved
$30	0 -	Protocol Timer CVR8			
$32	0 -	Protocol Timer CVR9			
$34	0 -	Protocol Timer CVR10			
$36	0 -	Protocol Timer CVR11			
$38	0 -	Protocol Timer CVR12			
$3A	0 -	Protocol Timer CVR13			
$3C	0 -	Protocol Timer CVR14			
$3E	0 -	Protocol Timer CVR15			

1. IRQA should be disabled.
2. Any Interrupt starting address (including a reserved address) can be used for MCU NMI (IPL = 3) or the MCU command interrupt (IPL = 0-2). These interrupts are issued by setting the appropriate bits in MCVR. See Table 5-10 on page 5-18.

Table 7-7. Interrupt Source Priorities within an IPL

<table>
<tr><th colspan="4">Level 3 (Non-maskable)</th></tr>
<tr><td>Highest</td><td colspan="3">Hardware $\overline{\text{RESET}}$</td></tr>
<tr><td></td><td colspan="3">Stack Error</td></tr>
<tr><td></td><td colspan="3">Illegal Instruction</td></tr>
<tr><td></td><td colspan="3">Debug Request Interrupt</td></tr>
<tr><td></td><td colspan="3">Trap</td></tr>
<tr><td>Lowest</td><td colspan="3">MDI MCU NMI</td></tr>
<tr><th colspan="4">Levels 0, 1, 2 (Maskable)</th></tr>
<tr><td>Highest</td><td>$\overline{\text{IRQA}}$</td><td></td><td>Protocol Timer CVR0</td></tr>
<tr><td></td><td>$\overline{\text{IRQB}}$ - from $\overline{\text{DSP\_IRQ}}$ pin</td><td></td><td>Protocol Timer CVR1</td></tr>
<tr><td></td><td>$\overline{\text{IRQC}}$ - from MDI</td><td></td><td>Protocol Timer CVR2</td></tr>
<tr><td></td><td>$\overline{\text{IRQD}}$ - from Protocol Timer</td><td></td><td>Protocol Timer CVR3</td></tr>
<tr><td></td><td>MDI MCU command</td><td></td><td>Protocol Timer CVR4</td></tr>
<tr><td></td><td>BBP Receive Data with Overrun Error</td><td></td><td>Protocol Timer CVR5</td></tr>
<tr><td></td><td>BBP Receive Data</td><td></td><td>Protocol Timer CVR6</td></tr>
<tr><td></td><td>BBP Receive Last Slot</td><td></td><td>Protocol Timer CVR7</td></tr>
<tr><td></td><td>BBP Receive Frame Counter</td><td></td><td>Protocol Timer CVR8</td></tr>
<tr><td></td><td>BBP Transmit Data with Underrun Error</td><td></td><td>Protocol Timer CVR9</td></tr>
<tr><td></td><td>BBP Transmit Last Slot</td><td></td><td>Protocol Timer CVR10</td></tr>
<tr><td></td><td>BBP Transmit Data</td><td></td><td>Protocol Timer CVR11</td></tr>
<tr><td></td><td>BBP Transmit Frame Counter</td><td></td><td>Protocol Timer CVR12</td></tr>
<tr><td></td><td>SAP Receive Data with Overrun Error</td><td></td><td>Protocol Timer CVR13</td></tr>
<tr><td></td><td>SAP Receive Data</td><td></td><td>Protocol Timer CVR14</td></tr>
<tr><td></td><td>SAP Receive Last Slot</td><td></td><td>Protocol Timer CVR15</td></tr>
<tr><td></td><td>SAP Transmit Data with Underrun Error</td><td></td><td>MDI Receive 0</td></tr>
<tr><td></td><td>SAP Transmit Last Slot</td><td></td><td>MDI Receive 1</td></tr>
<tr><td></td><td>SAP Transmit Data</td><td></td><td>MDI Transmit 0</td></tr>
<tr><td></td><td>SAP Timer Counter Rollover</td><td>Lowest</td><td>MDI Transmit 1</td></tr>
</table>

7.2.2 Enabling DSP Interrupt Sources

Two steps are required to enable DSP interrupt sources:

1. Assign the desired priority level to each peripheral and write to the Peripheral Interrupt Priority Register (IPRP).

Each of the four peripherals that can interrupt the DSP (MDI, PT, SAP, and BBP) as well as the MDI Command interrupt can be assigned a priority level from 0 (lowest) to 2 (highest). This assignment is done by writing to the IPRP. The choice of priority level for each peripheral depends on several factors driven by the end application, including:

- Rate of service requests
- Latency requirements
- Access to the alternate register bank
- Length of service routine
- Total number of interrupt sources in the system

This step is normally done once during system initialization.

2. Program the appropriate peripheral registers to generate the desired interrupt requests.

7.2.3 DSP Interrupt Control Registers

IPRP Interrupt Priority Register, Peripherals **X:$FFFE**

	BIT 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT 0
	▨	▨	▨	▨	▨	▨	MDIPL[1:0]		PTPL[1:0]		SAPPL[1:0]		BBPPL[1:0]		MDCPL[1:0]	
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 7-8. IPRP Description

Name	Description	Setting
MDIPL[1:0] Bits 9–8	**MDI Interrupt Priority Level**	00 = Disabled (default). 01 = Priority level 0. 10 = Priority level 1. 11 = Priority level 2.
PTPL[1:0] Bits 7–6	**Protocol Timer Interrupt Priority Level**	
SAPPL[1:0] Bits 5–4	**SAP Interrupt Priority Level**	
BBPPL[1:0] Bits 3–2	**BBP Interrupt Priority Level**	
MDCPL[1:0] Bits 1–0	**MDI Command Priority Level**	

IPRC Interrupt Priority Register, Core **X:$FFFF**

	BIT 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT 0
					IDTM	IDPL[1:0]		ICTM	ICPL[1:0]		IBTM	IBPL[1:0]		IATM	IAPL[1:0]	
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 7-9. IPRC Description

Name	Description	Setting
IDTM Bit 11	**Interrupt D Trigger Mode—**Should remain level-sensitive.	0 = Level-sensitive (default). 1 = Edge sensitive.
ICTM Bit 8	**Interrupt C Trigger Mode—**Should remain level-sensitive.	
IBTM Bit 5	**Interrupt B Trigger Mode—**Should remain level-sensitive.	
IATM Bit 2	**Interrupt A Trigger Mode**	
IDPL[1:0] Bits 10–9	**Interrupt D Priority Level—**This interrupt should be enabled before the DSP enters STOP mode.	00 = Disabled (default). 01 = Priority level 0. 10 = Priority level 1. 11 = Priority level 2.
ICPL[1:0] Bits 7–6	**Interrupt C Priority Level—**This interrupt should be enabled before the DSP enters STOP mode.	
IDPL[1:0] Bits 4–3	**Interrupt B Priority Level—**This interrupt is generated by the DSP_IRQ pin. It should be activated using a software protocol between the DSP and the external source, signaling the external device when to deassert the interrupt.	
IAPL[1:0] Bits 1–0	**Interrupt A Priority Level—**This interrupt should remain disabled.	

7.3 Edge Port

The Edge Port (EP) consists of eight GPIO pins, INT7–0, each of which can generate an interrupt if the associated bit in the NIER or FIER is set. This port is controlled by four configuration registers:

- EPPAR—The EP Pin Assignment Register configures the trigger mechanism for each pin: level-sensitive or rising and/or falling edge triggered.
- EPFR—The EP Flag Register contains bits that are set when the associated Edge I/O inputs are triggered.
- EPDDR—The EP Data Direction Register configures each pin as either an input or output.

- EPDR—The EP Data Register serves as a GPIO buffer. A write to this register determines the data driven on output pins; data received on input pins can be read from this register.

A diagram of an Edge I/O pin is shown in Figure 7-4.

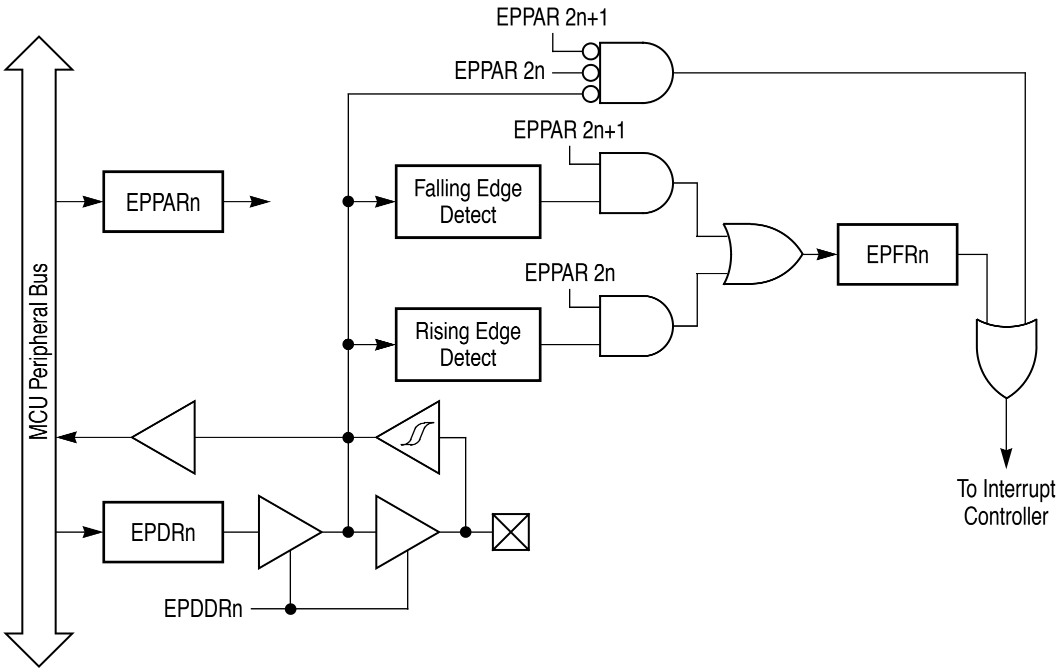


Figure 7-4. Edge I/O Pin

EPPAR Edge Port Pin Assignment Register **$0020_9000**

	BIT 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT 0
	EPPA7		EPPA6		EPPA5		EPPA4		EPPA3		EPPA2		EPPA1		EPPA0	
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 7-10. EPPAR Description

Name	Description	Settings
EPPA7–0	**Edge Port Pin Assignment 7–0—**Each pair of bits determines the trigger mechanism for an Edge I/O input. Interrupt requests are always generated from this block, but may be masked within the MCU interrupt controller. The functionality of this register is independent of the programmed pin direction. Pins configured as level-sensitive are inverted so that a logic low on the external pin represents a valid interrupt request. Level-sensitive interrupt inputs are not latched. The interrupt source must assert the signal until it is acknowledged by software to guarantee that a level-sensitive interrupt request is acknowledged. Pins configured as edge-sensitive interrupts are latched and need not remain asserted. Pins programmed as edge-detecting are monitored regardless of the configuration as input or output.	00 = Level-sensitive (default). 01 = Rising edge-sensitive. 10 = Falling edge-sensitive. 11 = Both rising and falling edge-sensitive.

EPDDR Edge Port Data Direction Register **$0020_9002**

	BIT 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT 0
	▨	▨	▨	▨	▨	▨	▨	▨	EPDD7	EPDD6	EPDD5	EPDD4	EPDD3	EPDD2	EPDD1	EPDD0
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 7-11. EPDDR Description

Name	Description	Settings
EPDD[7:0] Bits 7–0	Each of these bits controls the data direction of the corresponding Edge I/O pin. Pin direction is independent of its programmed level/edge mode.	0 = Input (default) 1 = Output

EPDR Edge Port Data Register **$0020_9004**

	BIT 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT 0
									EPD7	EPD6	EPD5	EPD4	EPD3	EPD2	EPD1	EPD0
RESET	0	0	0	0	0	0	0	0	—	—	—	—	—	—	—	—

Table 7-12. EPDR Description

Name	Description
EPD[7:0] Bits 7–0	Each of these bits contains data for the corresponding Edge I/O pin. Writes to EPDR are stored in an internal latch and driven on any port pin configured as an output. Reads of this register return the value sensed on input pins and the latched data driven on outputs.

EPFR Edge Port Flag Register **$0020_9006**

	BIT 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT 0
									EPF7	EPF6	EPF5	EPF4	EPF3	EPF2	EPF1	EPF0
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 7-13. EPFR Description

Name	Description
EPF7–0 Bits 7–0	**Edge Port Flag 7–0**—Each bit in this register is set when the associated pin detects the edge input programmed in the corresponding EPPA bit. The bit remains set until it is cleared by writing a “1” to it. A pin configured as level-sensitive does not affect this register. A write to EPDR that triggers a pin’s level or edge will set the corresponding EPF bit. The outputs of this register drive the corresponding input of the interrupt controller for those bits configured as edge detecting.

Chapter 8
Queued Serial Peripheral Interface

The Queued Serial Peripheral Interface (QSPI) is a full-duplex synchronous serial interface providing SPI-compatible data transfer between the DSP56652 and up to five peripherals. Four prioritized data queues (Queue3–Queue0; Queue3 is highest priority) can be triggered by the protocol timer and the MCU. Each queue can contain several sub-queues, and each sub-queue can be transferred to any of the five peripherals. The queues can be variable sizes of 8- or 16-bit multiples.

Note: A *queue* is defined as a series of data that is transferred sequentially. Data can be 8 or 16 bits, and each data entry occupies a 16-bit location in QSPI Data RAM. Each datum in an 8-bit data queue occupies the lower byte of its RAM location; the upper byte is zero-filled.

A *sub-queue* is a sequence of data within a queue that is transmitted without interruption.

8.1 Features

The primary QSPI features include the following:

- Full-duplex, three wire synchronous transfers
- Half-duplex, two wire synchronous transfers
- End-of-transfer interrupt flag
- Programmable serial clock polarity and serial clock phase
- Programmable delay between chip-select and serial clock
- Programmable baud rates
- Programmable queue lengths and continuous transfer mode
- Programmable peripheral chip-selects
- Programmable queue pointers
- Four transfer activation triggers
- Programmable delay after transfer
- Automatic loading of programmable address at end of queue
- Pause enable at queue entry boundaries

Several of these features are not found on standard SPIs and are further described below.

8.1.1 Programmable Baud Rates

Each of the peripheral chip-select lines in the QSPI has its own programmable baud rate. The frequency of the internally-generated serial clock can range from MCU_CLK to (MCU_CLK ÷ 504).

8.1.2 Programmable Queue Lengths and Continuous Transfers

The number of entries in a queue is programmable, allowing the QSPI to transfer up to 63 halfwords or bytes without MCU intervention. Continuous transfers of information to several peripherals can be activated with a single trigger, resulting in greatly reduced MCU/QSPI interaction.

8.1.3 Programmable Peripheral Chip-Selects

Five chip-select pins are provided for connection to up to five SPI peripherals. Software can activate any one pin at a given time, and each pin can be programmed to be active high or active low. The active chip-select signal can be changed at any time, including during a queue transfer.

8.1.4 Programmable Queue Pointers

Each of the four queues has a programmable queue pointer that contains the RAM address for the next data to be transmitted or received. The MCU can configure the QSPI to switch from one task to another by writing the address of the next task to the queue pointer during queue setup.

8.1.5 Four Transfer Activation Triggers

QSPI transfers are activated by any of four transfer triggers from the protocol timer or the MCU. Each timer or MCU transfer trigger initiates a transfer of successive data from RAM, starting at the address pointed to by the queue pointer for that trigger.

8.1.6 Programmable Delay after Transfer

Some serial peripherals require additional chip-select hold time after a transfer is completed. To simplify the interface to these devices, a delay of 1 to 128 serial clock cycles between queues can be programmed at the completion of a queue transfer.

8.1.7 Loading a Programmable Address at the End of Queue

A queue can be configured so that its last data entry is written to its queue pointer, thus programming the start address for the next queue trigger from the queue itself. This enables wrapping to the beginning of the queue or branching from one sequence to another when a new transfer trigger activates the queue.

8.1.8 Pause Enable at Queue Entry Boundaries

A queue transfer can be programmed to terminate at queue entry boundaries by inserting a PAUSE command in the control halfword of the queue entry at that boundary. This feature enables each of the four transfer triggers to provide programmable multiple-task support and considerably reduces MCU intervention.

8.2 QSPI Architecture

This section describes the QSPI pins, control registers, functional modules, and special-purpose RAM. Most of these components are shown in the QSPI flow diagram in Figure 8-1.

The QSPI port can also function as GPIO, which is governed by three control registers that are also described.

8.2.1 QSPI Pins

The QSPI pin description in Section 8.2 is repeated in Table 8-1 for convenience. All pins are GPIO when not programmed otherwise, and default as general-purpose inputs after reset.

Note: The DSP56652 QSPI always functions as SPI master.

Table 8-1. Serial Control Port Signals

Signal Name	Type	Reset State	Signal Description
SPICS0–SPICS4	Output	GPI	**Serial peripheral interface chip select 0–4—**These output signals provide chip select signals for the Queued Serial Peripheral Interface (QSPI). The signals are programmable as active high or active low. SPICS0–3 have internal pull-up resistors, and SPICS4 has an internal pull-down resistor.
SCK	Output	GPI	**Serial clock—**This output signal provides the serial clock from the QSPI for the accessed peripherals. The delay (number of clock cycles) between the assertion of the chip select signals and the first transmission of the serial clock is programmable. The polarity and phase of SCK are also programmable.
MISO	Input	GPI	**Synchronous master in slave out—**This input signal provides serial data input to the QSPI. Input data can be sampled on the rising or falling edge of SCK and received in QSPI RAM most significant bit or least significant bit first.
MOSI	Output	GPI	**Synchronous master out slave in—**This output signal provides serial data output from the QSPI. Output data can be sampled on the rising or falling edge of SCK and transmitted most significant bit or least significant bit first.

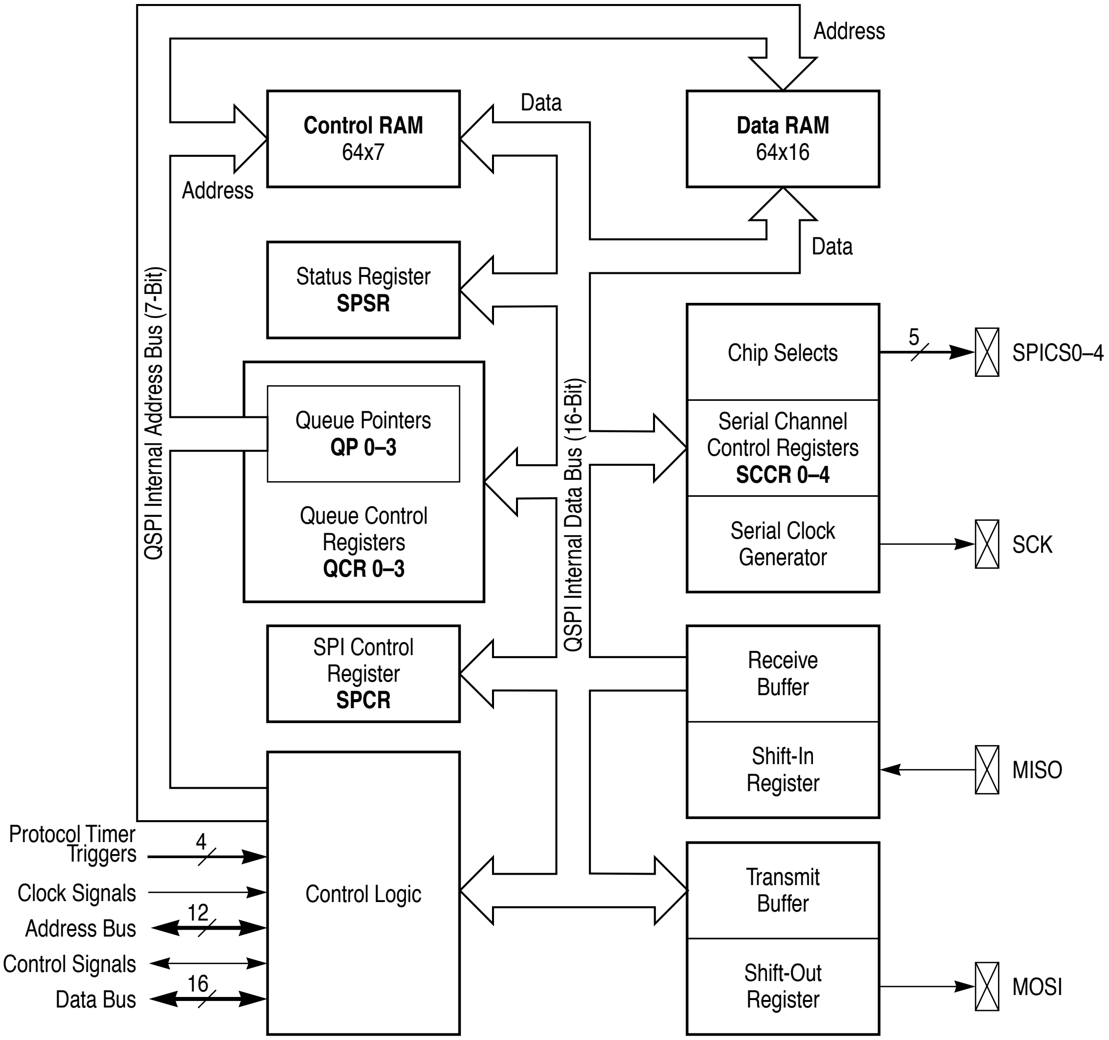


Figure 8-1. QSPI Signal Flow

8.2.2 Control Registers

A brief summary of the control registers for QSPI and GPIO operation is given below. More detailed descriptions can be found in Section 8.4 on page 8-12.

The QSPI uses the following control registers:

- SPSR—The Serial Peripheral Status Register indicates which of the four queues is active, executing or has ended a transfer with an interrupt. It also contains flags for a HALT request acknowledge, trigger collision, or queue pointer wraparound.
- SPCR—The Serial Peripheral Control Register enables QSPI operation, enables the four queues, sets the polarity of the five chip selects, enables trigger accumulation (queue 1 only), initiates a QSPI HALT, selects QSPI behavior in DOZE mode, and enables interrupts for HALT acknowledge, trigger collision and queue wraparound.
- QCR3–0—Each of Queue Control Registers 3–0 contains the queue pointer for its associated queue, enables use of the last data entry in the queue as the start address of the next queue, and determines if the queue responds to a HALT at the next sub-queue boundary or queue command. QCR1 also contains a counter for a trigger accumulator.
- SCCR4–0—Each of Serial Channel Control Registers 4–0 controls the serial clock frequency, phase, and polarity for its associated channel, the delay between chip-select assertion and the serial clock activation, the delay between chip-select deassertion and the start of the next transfer, and the order of data transmission (least significant bit first or last).

These registers determine the GPIO functions of the QSPI pins:

- QPCR—The QSPI Port Configuration Register configures each of the eight pins as either QSPI or GPIO.
- QDDR—The QSPI Data Direction Register determines if each pin that is configured as GPIO is an input or an output.
- QPDR—The QSPI Port Data Register contains the data that is latched on each GPI pin and written to each GPO pin.

8.2.3 Functional Modules

The QSPI functional modules include the following:

- The **chip select module** uses data from a queue's control RAM entry to select the appropriate SPICS pin and Serial Channel Control Register (SCCR) for the serial transfer of the queue entry.
- The **serial clock generator** derives the serial clock SCK from the system clock based on information in the active SCCR.
- The **shift-in register** uses SCK to shift in received data bits at the MISO pin and assembles the bits into a received data halfword or byte. When the last bit is received the data is immediately latched in the receive buffer so that the shift in register can receive the next data with no delay.
- If receive is enabled, the **receive buffer** latches each received byte or halfword from the shift in register and writes it to the QSPI Data RAM at the address contained in the queue pointer in the queue's QCR.
- The **shift-out register** uses SCK to shift out transmitted data bits at the MOSI pin. It loads the next data from the transmit buffer to be transferred immediately after the last bit of the current datum is sent, enabling smooth transmission with no delay.
- The **transmit buffer** holds the next byte or halfword to be transmitted. While the current datum is being transmitted, the QSPI loads the next datum to the transmit buffer from Data RAM. The address of the next datum is contained in the queue pointer in the queue's QCR.

8.2.4 RAM

There are two byte-addressable QSPI RAM segments:

- The **Data RAM** is a 64 × 16 bit block that stores transmitted and received QSPI data. The MCU writes data to be transmitted in the Data RAM. If receive is enabled, the QSPI writes received data from the receive buffer to the Data RAM, overwriting the transmitted data. The MCU can then read the received data from RAM.
- The **Control RAM** is a 64 × 16 bit block that contains a control halfword for each datum in the Data RAM. The control information includes chip select or QSPI command (end of queue, end of transmission, or no activity), data width of a queue entry (8 or 16 bits), receive enable, and pause at end of a sub-queue.

Each datum and corresponding control halfword constitute a queue entry. The MCU initializes the Data RAM and the Control RAM by loading them with transmission data and queue transfer control information.

8.3 QSPI Operation

The QSPI operates in master mode and is always in control of the SPI bus. Data is transferred as either least or most significant bit first, depending on the LSBF*n* bit in SCCR*n*. A transfer can be either 8 or 16 bits, depending on the value of the BYTE bit for the queue entry. When the BYTE bit is set, the least significant byte of the Data RAM entry is transferred, and if receiving is enabled, the least significant byte of the data halfword is valid while the most significant byte is filled with 0s.

The QSPI has priority in using its internal bus. If an MCU access occurs while the QSPI is using the bus, the MCU waits for one cycle.

8.3.1 Initialization

The following steps are required to begin QSPI operation:

1. Write the QPCR to configure unused pins for GPIO and the rest for QSPI.
2. Write the SPCR to adjust Chip Select pin polarities and enable queues and interrupts.
3. Write the QCRs to initialize the queue pointers and determine behavior when executing queues are preempted.
4. Write the SCCR registers to adjust the baud rate, phase, polarity, and delays for the SCK for each CS pin, as well as the order bits are sent.
5. Write the Data RAM with information to be transmitted for each queue.
6. Write the Control RAM with control information for each queue, including
    c. Data width (8 or 16 bits)
    d. Enable data reception if applicable
    e. Chip select or queue termination
7. Enable the QSPI by setting the QSPE bit in the SPCR.

At this point, the QSPI awaits a queue trigger to initiate a transfer.

8.3.2 Queue Transfer Cycle

A QSPI transfer is initiated by a transfer trigger. There are eight possible sources of transfer triggers, four from the MCU and four from the Protocol Timer. An MCU trigger is activated by writing to one of the four trigger addresses at $0020_5FF8–$0020_5FFE. The content of the write is ignored; the write itself is the trigger.

In normal operation, the following sequence occurs:

1. The MCU or Protocol Timer issues a transfer trigger.
2. The targeted queue becomes *active*. The QSPI asserts QA*n* in the SPSR. (The MCU has previously enabled operation for this queue by setting its enable bit QE*n* in the SPCR.)
3. If no higher priority queue is transferring data, the targeted queue begins *executing*. The QSPI asserts QX*n* in the SPSR.
4. The QSPI uses the queue pointer QP*n* in QCR*n* to determine the offset of the queue's entry in RAM.
5. The QSPI reads the datum and command halfword of the queue entry from RAM, and writes the datum to the transmit buffer.
6. The datum is latched into the shift-out register
7. The QSPI selects the peripheral chip-select line from the PCS field in the control halfword and asserts it.
8. The shift-out register uses SCK to shift its contents out to the peripheral through the MOSI pin.
9. Received datum is also clocked in to the shift-in register if the RE bit in the queue entry's control halfword is set.
10. As transfer begins, QP*n* is incremented, the next datum and control halfword are read from RAM, and the datum is latched in the transmit out buffer.
11. All 8 or 16 bits in the queue entry are transmitted. When the last bit is transferred, the next datum in the transmit buffer is immediately latched into the shift-out register and received datum, if any, is immediately latched to the receive buffer so that there is no delay between transfers.

Steps 7–11 repeat until the cycle is ended or broken by a QSPI command, a higher priority QSPI trigger, or a power-down mode.

8.3.3 Ending a Transfer Cycle

A transfer cycle ends when all data in the queue has been transferred. This condition is indicated in the last control halfword by either setting the PAUSE bit or programming the PCS field with NOP or EOQ (refer to the Control RAM description on page 8-22).

8.3.3.1 PAUSE

At the completion of transfer of each queue entry, the QSPI checks whether the PAUSE bit is set in the control halfword for that entry. If so, the QSPI assumes it has reached the end of the programmed queue and clears the QA*n* and the QX*n* flags. If EOTIE is detected in the PCS field of the control halfword for that queue entry, the QSPI sets the EOT*n* flag in the SPSR and generates an interrupt to the MCU. If the PAUSE bit is cleared, the QSPI continues the transfer process.

8.3.3.2 NOP and EOQ

Each time the QSPI loads a queue entry from RAM (step 5 or 10 in the transfer cycle on page 8-9) it checks for EOQ (end of queue) or NOP (no operation) in the PCS field. If the QSPI detects one of these codes, it assumes it will reach the end of the programmed queue after it completes the transfer of the current datum and clears the QA*n* and QX*n* flags. If EOTIE is detected for the present queue entry, the QSPI asserts the EOT*n* flag and generates an interrupt to the MCU.

The EOQ command can also be used to program the next entry point for the queue without MCU intervention. If LE*n* in QCR*n* is set when a cycle terminates with EOQ, the QSPI writes the 6 least significant bits of the queue entry’s datum into the QP*n* field of QCR*n*.

8.3.4 Breaking a Transfer Cycle

Normally, once a queue is started, transfer continues until an end of queue is indicated. When the queue completes its transfer, the next active queue with the highest priority begins execution. However, a queue can be interrupted at a sub-queue boundary to enable a higher priority queue to execute rather than waiting for the current queue to finish. If a higher priority queue is triggered while a lower priority queue is executing and the HMD bit of the lower priority queue’s QCR is cleared, the QSPI suspends the execution of the lower priority queue at the next sub-queue boundary and starts executing the higher priority queue. The QA bit of the suspended queue remains set, and the QSPI resumes execution of the lower priority queue after it has completed the execution of the higher priority queue.

A *sub-queue boundary* is a queue entry whose control halfword contains a cleared CONT bit and/or a PCS field that activates a different SPICS line than the currently active one.

Setting the CONT bit keeps the current chip select line active. Clearing the CONT bit deasserts the current chip-select line and stops the current transfer. If the CONT bit is set and the chip selection in the next queue entry is different than that of the present one, the chip select line remains active between queue entry transfers and is deactivated two MCU_CLK cycles before the new chip select line for the next queue entry is activated.

If both the CONT bit and the PAUSE bit in the queue entry's control halfword are set, or if EOQ is detected in the next control halfword, the chip select line also continues to be activated after the sub-queue/queue transfer has been completed.

8.3.5 Halting the QSPI

When the MCU wants to “soft disable” the QSPI at a queue boundary, it asserts the HALT bit in SPCR. If the QSPI is in the process of transferring a queue, it suspends the transfer at the next sub-queue or queue boundary, depending on the queue’s HMD bit. It then asserts the HALTA bit in the SPSR and QSPI operation stops. If the HLTIE bit in the SPCR is set, asserting HALTA generates an interrupt to the MCU. The QSPI state machines and the QSPI registers are not reset during the HALT process, and the QSPI resumes operation where it left off when the MCU deasserts HALTA. During the HALT mode, the QSPI continues to accept new transfer triggers from the protocol timer and MCU, and the MCU can access any of the QSPI registers and RAM addresses.

The MCU can immediately disable the QSPI by clearing the QSPE bit in the SPCR. All QSPI state machines and the SPSR are reset. Data in an ongoing transfer can be lost, and the external SPI device can be disrupted.

8.3.6 Error Interrupts

If a queue pointer contains $3F when the next control halfword is fetched and the QP is not loaded from the data halfword, it wraps around to $00 and the QPWF flag in the SPSR is set. If the WIE bit in the SPCR is set, an interrupt is generated to the MCU.

If a trigger for a queue occurs while the queue is active the TRC flag in the SPSR is set. If the TRCIE bit in the SPCR is set, an interrupt is generated to the MCU.

8.3.7 Low Power Modes

If the QSPI detects a DOZE signal and the DOZE bit in the SPCR is set, the QSPI halts its operation as if the HALT bit had been set. When the MCU exits DOZE mode, it must clear the HALTA bit to resume QSPI operation.

When the QSPI detects a STOP signal, it halts immediately by shutting off its clocks. The status of the QSPI is left unchanged, but any ongoing transfer is lost and the peripheral can be disrupted.

8.4 QSPI Registers and Memory

This section describes the QSPI control registers, data and control RAM, and GPIO registers. These areas are summarized in Table 8-2.

Table 8-2. QSPI Register/Memory Summary

Address[1]	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CONT. RAM $000 - $07F										BYTE	RE	PAUSE	CONT	PCS / EOTIE / NOP / EOQ		
$080 - $3FF	Reserved															
DATA RAM $400 - $47F	MOST SIGNIFICANT BYTE								LEAST SIGNIFICANT BYTE							
$480 - $EFF	Reserved															
QPCR $F00									PC7 (SCK)	PC6 (MOSI)	PC5 (MISO)	PC4 (CS4)	PC3 (CS3)	PC2 (CS2)	PC1 (CS1)	PC0 (CS0)
QDDR $F02									PD7 (SCK)	PD6 (MOSI)	PD5 (MISO)	PD4 (CS4)	PD3 (CS3)	PD2 (CS2)	PD1 (CS1)	PD0 (CS0)
QPDR $F04									D7 (SCK)	D6 (MOSI)	D5 (MISO)	D4 (CS4)	D3 (CS3)	D2 (CS2)	D1 (CS1)	D0 (CS0)
SPCR $F06	CSPOL4	CSPOL3	CSPOL2	CSPOL1	CSPOL0	QE3	QE2	QE1	QE0	HLTIE	TRCIE	WIE		HALT	DOZE	QSPE
QCR0 $F08	LE0	HMD0									QP0					
QCR1 $F0A	LE1	HMD1					TRCNT1				QP1					
QCR2 $F0C	LE2	HMD2									QP2					
QCR3 $F0E	LE3	HMD3									QP3					
SPSR $F10	QX3	QX2	QX1	QX0	QA3	QA2	QA1	QA0		HALTA	TRC	QPWF	EOT3	EOT2	EOT1	EOT0
SCCR0 $F12	CPHA0	CKPOL0	LSBF0	DATR0			CSCKD0			SCKDF0						
SCCR1 $F14	CPHA1	CKPOL1	LSBF1	DATR1			CSCKD1			SCKDF1						
SCCR2 $F16	CPHA2	CKPOL2	LSBF2	DATR2			CSCKD2			SCKDF2						
SCCR3 $F18	CPHA3	CKPOL3	LSBF3	DATR3			CSCKD3			SCKDF3						
SCCR4 $F1A	CPHA4	CKPOL4	LSBF4	DATR4			CSCKD4			SCKDF4						
$F1C - $FF7	Reserved															
$FF8	MCU Trigger for Queue 0															
$FFA	MCU Trigger for Queue 1															
$FFC	MCU Trigger for Queue 2															
$FFE	MCU Trigger for Queue 3															

[1] All addresses are offsets from $0020_5000.

8.4.1 QSPI Control Registers

The following registers govern QSPI operation:

- SPCR—Serial Port Control Register
- QCR0–3—QSPI Control Registers
- SPSR—Serial Port Status Register
- SCCR0–4—Serial Channel Control Registers

SPCR Serial Port Control Register **$0020_5F06**

	BIT 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT 0
	CSPOL4	CSPOL3	CSPOL2	CSPOL1	CSPOL0	QE3	QE2	QE1	QE0	HLTIE	TRCIE	WIE	TACE	HALT	DOZE	QSPE
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Note: Either the EQSPI bit in the NIER or the EFQSPI bit in the FIER must be set in order to generate any of the interrupts enabled in the SPCR (see page 7-7).

Table 8-3. SPCR Description

Name	Description	Settings
CSPOL[4:0] Bits 15–11	**Chip Select Polarity[4:0]—**These bits determine the active logic level of the QSPI chip select outputs.	0 = SPICSn is active low (default). 1 = SPICSn is active high.
QE[3:0] Bits 10–7	**Queue Enable[3:0]—**Each of these bits enables its respective queue to be triggered. If QEn is cleared, a trigger to this queue is ignored. If QEn is set, a trigger for Queue n makes Queue n active, and the QAn bit in the SPSR is asserted. Queue n executes when it is the highest priority active queue. Each of these bits can be set or cleared independently of the others.	0 = Queue n is disabled (default). 1 = Queue n is enabled.
HLTIE Bit 6	**HALTA Interrupt Enable—**Enables an interrupt when the HALTA status flag in the SPSR asserted.	0 = Interrupt disabled (default). 1 = Interrupt enabled.
TRCIE Bit 5	**Trigger Collision Interrupt Enable—**Enables an interrupt when the TRC status flag in the SPSR is asserted.	0 = Interrupt disabled (default). 1 = Interrupt enabled.
WIE Bit 4	**Wraparound Interrupt Enable—**Enables an interrupt when the QPWF status flag in the SPSR is asserted.	0 = Interrupt disabled (default). 1 = Interrupt enabled.
TACE Bit 3	**Trigger Accumulation Enable—**Enables trigger accumulation for Queue 1. The trigger count is contained in the TRCNT1 field in QCR1. When TACE is set, a trigger to Queue 1 increments TRCNT1, and completion of a Queue 1 transfer decrements TRCNT1. Note: This function and the TRCNT1 field in QCR1 are available only for Queue 1. Setting or clearing the TACE bit has no effect on Queues 3, 2, or 0.	0 = Trigger accumulation is disabled and the TRCNT1 field is cleared (default). 1 = Trigger accumulation enabled.

Table 8-3. SPCR Description (Continued)

Name	Description	Settings
HALT Bit 2	**Halt Request—**When the MCU sets the HALT bit, the QSPI finishes any ongoing serial transfer, asserts HALTA, and halts. If a queue is executing when HALT is asserted, the QSPI checks the value of the HMD bit in its QCR. If HMD is clear, the QSPI halts only at the next PAUSE, NOP or EOQ commands. If HMD is set, the QSPI halts at the next sub-queue boundary. During Halt mode the QSPI continues to accept new transfer triggers from the protocol timer and MCU, and the MCU can access any of the QSPI registers and RAM addresses. The HALT bit is cleared when HALTA is deasserted, so that only one MCU access is required to exit the Halt state. The QSPI state machines and the SPCR are not reset during the Halt process, so the QSPI resumes operation where it left off. NOTE: The HALT bit is checked only at sub-queue or queue boundaries. If the HALT bit is asserted and then deasserted before a sub-queue transfer has completed, the QSPI does not recognize a Halt request.	0 = Normal operation. 1 = Halt request.
DOZE Bit 1	**DOZE Enable—**Determines the QSPI response to Doze mode. If the DOZE bit is set when DOZE mode is identified, the QSPI finishes any ongoing serial transfer and halts as if the HALT bit were set. When the DOZE mode is exited, the MCU must clear HALTA for the QSPI to resume operation. If the DOZE bit is cleared, DOZE mode is ignored.	0 = QSPI ignores DOZE mode (default). 1 = QSPI halts in DOZE mode.
QSPE Bit 0	**QSPI Enable—**Setting QSPE enables QSPI operation. The QSPI begins monitoring transfer triggers from the Protocol Timer and the MCU. Both the QSPI and the MCU have access to the QSPI RAM. Clearing QSPE disables the QSPI. All QSPI state machines and the status bits in the SPSR are reset; other registers are not affected. The MCU can use the QSPI RAM and access its registers, and all the QSPI pins revert to GPIO configuration, regardless of the value of the QPCR bits. To avoid losing an ongoing data transfer and disrupting an external device, issue a HALT request and wait for HALTA before clearing QSPE. Pending transfer triggers will still be lost.	0 = QSPI disabled; QSPI pins are GPIO (default). 1 = QSPI enabled.

QCR0 Queue Control Register 0 $0020_5F08

	BIT 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT 0
	LE0	HMD0	▨	▨	▨	▨	▨	▨	▨	▨	QP0					
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

QCR1 Queue Control Register 1 $0020_5F0A

	BIT 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT 0
	LE1	HMD1	▨	▨	▨	▨	TRCNT1				QP1					
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

QCR2 Queue Control Register 2 $0020_5F0C

	BIT 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT 0
	LE2	HMD2	▨	▨	▨	▨	▨	▨	▨	▨	QP2					
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

QCR3 Queue Control Register 3 $0020_5F0E

	BIT 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT 0
	LE3	HMD3	▨	▨	▨	▨	▨	▨	▨	▨	QP3					
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The MCU can read and write QCR0–3. The QSPI can read these registers but can only write to the queue pointer fields, QP[5:0]. Writing to an active QCR is prohibited while it is executing a transfer. It is highly recommended that writing to the QCRs be done only when the QSPI is disabled or in HALT state.

Table 8-4. QCR Description

Name	Description	Settings
LEn Bit 15	**Load Enable for Queue n—**Enables loading a new value to the queue pointer (QPn) of Queue n. If LEn is set when the QSPI reaches an End Of Queue (EOQ) command (PCS = 111 in the Queue n control halfword) the value of the least significant byte of the data halfword of that queue entry is loaded into QPn. This allows the next triggering of queue n to resume transfer at the address loaded from the data halfword.	0 = QP loading disabled (default). 1 = QP loading enabled.
HMDn Bit 14	**Halt Mode for Queue n—**Defines the point at which the execution of queue n is halted when the MCU sets the HALT bit in the SPCR or when a higher priority transfer trigger is activated.	0 = Halts at any sub-queue boundary. 1 = Halts only at PAUSE, NOP, or EOQ.

Table 8-4. QCR Description (Continued)

Name	Description	Settings
TRCNT1 Bits 9-6	**Trigger Count for Queue 1—**When the TACE bit in the SPCR is set, trigger accumulation is enabled, and the TRCNT1 field can take values other than 0. If Queue 1 receives a transfer trigger while it is active (i.e., the QA1 bit in the SPSR is set), the TRCNT1 field is incremented. The TRCNT1 field is decremented when a Queue 1 transfer completes. As many as 16 triggers can be accumulated and subsequently processed. The TRCNT1 field cannot be incremented beyond the value of 1111 or decremented below 0. If a trigger for Queue 1 arrives when the TACE bit and all the bits of TRCNT field are set, the TRC flag in the SPSR is asserted to signify a trigger collision. If transfer of Queue 1 is completed when all the bits in TRCNT field are cleared, QA1 is deasserted. This field can only be read by the MCU; writes to this field are ignored. There is no TRCNT field in QCR0, QCR2, or QCR3. Bits 9–6 of these registers are reserved.	
QPn Bits 5–0	**Queue Pointer for Queue n—**This field contains the address of the next queue entry for the associated queue. The MCU initializes the QP to point to the first address in a queue. As queue n executes, the QPn is incremented each time a queue entry is fetched from RAM. If an EOQ command is identified in the queue entry's control halfword, and the LEn bit is asserted, the six least significant bits of the data halfword in the queue entry are loaded into QPn before queue execution is completed. This initializes the queue pointer for the next queue without MCU intervention. A write to the QP field while its queue is executing is disregarded. NOTE: The QP range is \$00–\$3F for 64 queue entries. Because the queue entries themselves are 16-bit halfwords that are byte-addressable, the actual offset that QP points to is two times the number contained in QP.	

SPSR Serial Port Status Register **$0020_5F10**

	BIT 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT 0
	QX3	QX2	QX1	QX0	QA3	QA2	QA1	QA0		HALTA	TRC	QPWF	EOT3	EOT2	EOT1	EOT0
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The MCU can read the SPSR to obtain status information, and can write to it in order to clear the HALTA, TRC, QPWF, and EOT[3:0] status flags. Only the QSPI can assert bits in this register.

Table 8-5. SPSR Description

Name	Type 1	Description	Settings
QX[3:0] Bits 15–12	R	**Queue Executing—**The QSPI sets a queue's QX bit when the queue begins execution, and clears the bit when queue execution stops. Queue execution begins when a queue is active and no higher priority (higher-numbered) queue is executing. Execution stops under any of the following circumstances: • The queue transfer is completed and the queue becomes inactive • A higher priority queue is asserted • The MCU issues a HALT command.	0 = Queue not executing (default). 1 = Queue executing.
QA[3:0] Bits 11–8	R	**Queue Active—**The QSPI sets a queue's QA bit when it receives a transfer trigger for that queue, and clears the bit upon completion of the queue transfer.	0 = Queue not active (default). 1 = Queue active.
HALTA Bit 6	R/1C	**Halt Acknowledge Flag—**The QSPI asserts this bit when it has come to an orderly halt at the request of the MCU via an assertion of the HALT bit. If the HALT bit is asserted while the QSPI is transferring a queue, the QSPI continues the transfer until it either reaches the first sub-queue boundary, or until it reaches a PAUSE, NOP, or EOQ command, depending on the value of the HMD bit for that queue. Then the QSPI asserts HALTA, clears the QX bit for the executing queue, and halts. If the HALT bit is asserted while the QSPI is idle, HALTA is asserted and the QSPI halts immediately. If the HLTIE bit is set in the SPCR, an interrupt is generated to the MCU when HALTA is asserted. The MCU clears HALTA by writing it with 1.	0 = No Halt since last acknowledge or current halt has not been acknowledged (default). 1 = Current Halt has been acknowledged.

Table 8-5. SPSR Description (Continued)

Name	Type 1	Description	Settings
TRC Bit 5	R/1C	**Trigger Collision—**Asserted when a transfer trigger for one of the queues occurs while the queue is activated (QAn = 1). Software should allow sufficient time for a queue to finish executing a queue in normal operation before the queue is retriggered. If the TRCIE bit is set in the SPCR, assertion of the TRC bit generates an interrupt to the MCU. This bit can be cleared by the MCU by writing a value of logic 1 into it. For Queue 1, TRC is only asserted when the trigger counter (TRCNT) = 1111b and a new trigger occurs. The MCU clears TRC by writing it with 1.	0 = No collision. 1 = A collision has occurred.
QPWF Bit 4	R/1C	**Queue Pointer Wraparound Flag—**If a queue pointer contains the value $7F and is incremented to read the next word in the queue (step 10), the QP wraps around to address $00 and QPWF is asserted. If the WIE bit in the SPCR has been set, an MCU interrupt is generated. The MCU clears QPWF by writing it with 1. **Note:** QPWF is not asserted when a QP is explicitly written with $00 as a result of an EOQ command from Control RAM.	0 = No wraparound. 1 = A wraparound has occurred.
EOT[3:0] Bits 3–0	R/1C	**End of Transfer—**When the PCS field of a queue entry is EOTIE (PCS = 110), the QSPI asserts the associated EOT bit and generates an interrupt to the MCU. Because the source for this interrupt is the execution of a command in RAM, it may be difficult in some cases to detect the control halfword that was the source for the interrupt. The MCU clears each EOT by writing it with 1.	0 = No end of transfer. 1 = End of transfer has occurred.

1. R = Read only.
R/1C = Read, or write with 1 to clear (write with 0 ingored).

SCCR0	Serial Channel Control Register 0	**$0020_5F12**
SCCR1	Serial Channel Control Register 1	**$0020_5F14**
SCCR2	Serial Channel Control Register 2	**$0020_5F16**
SCCR3	Serial Channel Control Register 3	**$0020_5F18**
SCCR4	Serial Channel Control Register 4	**$0020_5F1A**

	BIT 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT 0
	CPHA	CKPOL	LSBF	DATR[2:0]			CSCKD[2:0]			SCKDF[6:0]						
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Each of these registers controls the baud-rate, timing, delays, phase and polarity of the serial clock (SCK) and the bit order for a corresponding chips select line, SPICS0–4. The MCU has full access to these registers, while the QSPI has only read access to them. The MCU cannot write to the SCCR of an active line, and it is highly recommended that writes to the SCCRs only be done when the QSPI is disabled or in HALT state.

Table 8-6. SCCR Description

Name	Description	Settings
CPHAn Bit 15	**Clock Phase for SPICSn—**Together with CKPOLn, this bit determines the relation between SCK and the data stream on MOSI and MISO. When the CPHAn bit is set, data is changed on the first transition of SCK when the SPICSn line is active. When the CPHAn bit is cleared, data is latched on the first transition of SCK when the SPICSn line is active. The timing diagrams for QSPI transfer when CPHAn is 0 and when CPHAn is 1 are shown in Figure 8-2 on page 8-21.	0 = Data is changed on the first transition of SCK (default). 1 = Data is latched on the first transition of SCK.
CKPOLn Bit 14	**Clock Polarity for SPICSn—**Selects the logic level of SCK when the QSPI is not transferring data (the QSPI is inactive). When the CKPOLn bit is set, the inactive state for SCK is logic 1. When the CKPOLn bit is cleared, the inactive state for SCK is logic 0. CKPOL is useful when changes in SCK polarity are required while SPICSn is inactive. The timing diagrams for QSPI transfer when CKPOLn is 0 and when CKPOLn is 1 are shown in Figure 8-2 on page 8-21.	0 = Inactive SCK state = logic low (default). 1 = Inactive SCK state = logic high.

Table 8-6. SCCR Description (Continued)

<table>
<tr><th>Name</th><th>Description</th><th>Settings</th></tr>
<tr><td>LSBFn
Bit 13</td><td>Transfer Least Significant Bit First for SPICSn—These bits select the order in which data is transferred over the MOSI and MISO lines when the SPICSn line is activated for the transfer. When the LSBFn is set, data is transferred least significant bit (LSB) first. When the LSBFn bit is cleared, data is transferred most significant bit (MSB) first. When the BYTE bit in the control halfword is asserted, only the least significant byte of the data halfword is transferred (the MSB is then bit 7), so the data must be right-aligned.</td><td>0 = MSB transferred first (default).
1 = LSB transferred first.</td></tr>
<tr><td>DATRn[2:0]
Bits 12–10</td><td>Delay After Transfer for SPICSn—These bits controls the delay time between deassertion of the associated SPICS line (when queue or sub-queue transfer is completed), and the time a new queue transfer can begin. Delay after transfer can be used to meet the deselect time requirement for certain peripherals.</td><td rowspan="2"><table>
<tr><th>DATR[2:0]
CSCKD[2:0]</th><th>Delay (SCK Cycles)</th></tr>
<tr><td>000</td><td>1 (default)</td></tr>
<tr><td>001</td><td>2</td></tr>
<tr><td>010</td><td>4</td></tr>
<tr><td>011</td><td>8</td></tr>
<tr><td>100</td><td>16</td></tr>
<tr><td>101</td><td>32</td></tr>
<tr><td>110</td><td>64</td></tr>
<tr><td>111</td><td>128</td></tr>
</table></td></tr>
<tr><td>CSCKDn[2:0]
Bits 9–7</td><td>CS Assertion to SCK Activation Delay—These bits control the delay time between the assertion of the associated chip-select pin and the activation of the serial clock. This enables the QSPI port to accommodate peripherals that require some activation time.</td></tr>
<tr><td>SCKDFn[6:0]
Bits 6–0</td><td>SCK Division Factor—These bits determine the baud rate for the associated peripheral. The SCKDF field includes two division factors. The MSB (SCKDF6) is a prescaler bit that divides MCU_CLK by a factor of 4 if set or by 1 if cleared, while SCKDF[5:0] divide MCU_CLK by a factor of 1 to 63 ($00–$3E). There is an additional division by 2. The effective SCK baud rate is
$$\frac{MCU_CLK}{(SCKDF[5:0] + 1) \cdot (3 \cdot SCKDF[6] + 1) \cdot 2}$$
The lone exception is SCKDF[6:0] = $7F, in which case SCK = MCU_CLK.</td><td>SCKDF Examples<table>
<tr><th>SCKDF[6:0]</th><th>Division Factor</th></tr>
<tr><td>000_0000</td><td>2</td></tr>
<tr><td>000_0001</td><td>4</td></tr>
<tr><td>000_0111</td><td>16</td></tr>
<tr><td>100_0000</td><td>8</td></tr>
<tr><td>100_1011</td><td>96</td></tr>
<tr><td>111_1110</td><td>504</td></tr>
<tr><td>111_1111</td><td>1</td></tr>
</table></td></tr>
</table>

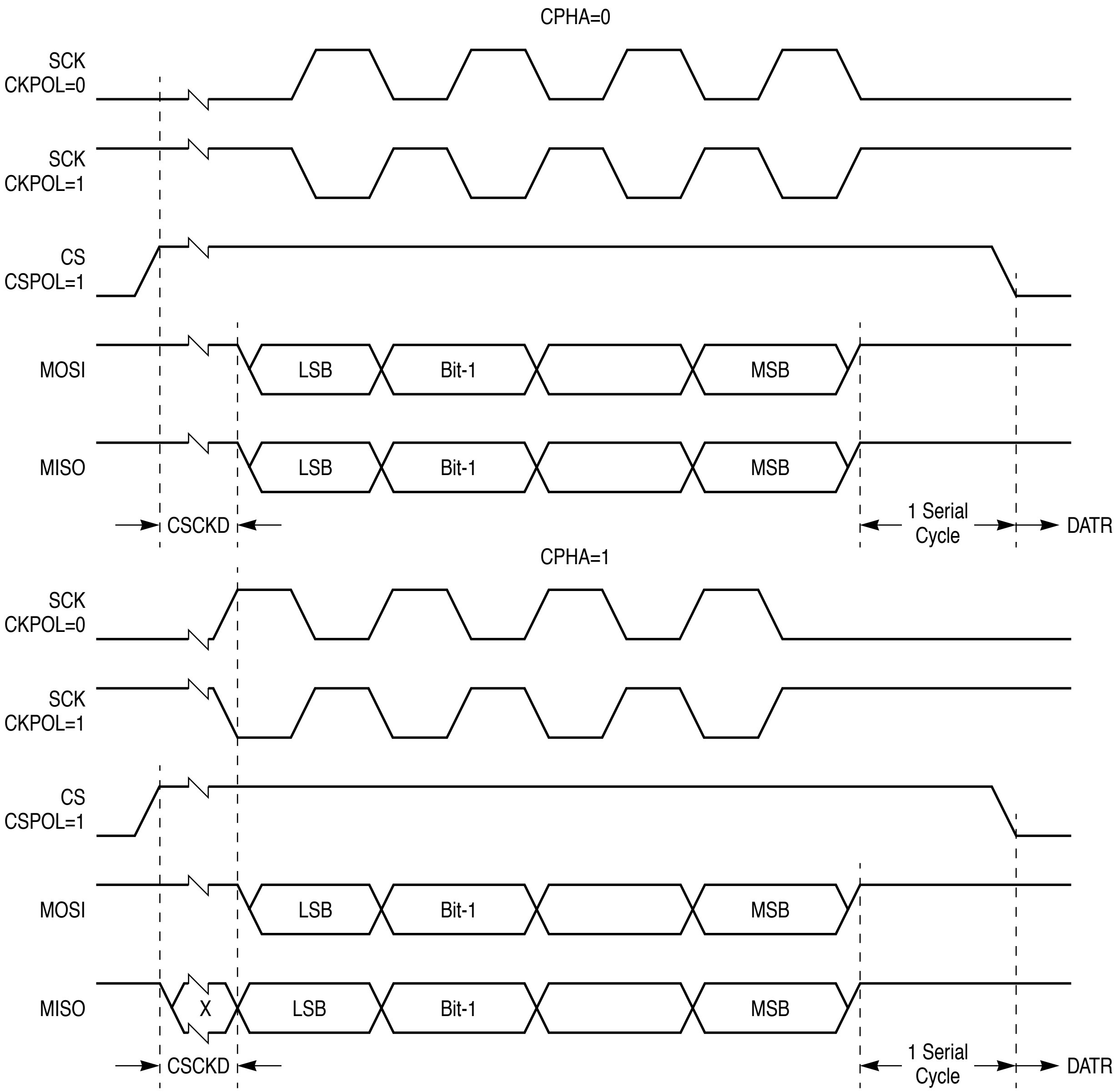


Figure 8-2. QSPI Serial Transfer Timing

8.4.2 MCU Transfer Triggers

The last four 16-bit addresses in the utilized memory area, $0020_5FF8 to $$0020_5FFE, are used for MCU triggers to Queue0–Queue3, respectively. When the MCU writes to one of these addresses, the QSPI generates a trigger for the appropriate queue in the same fashion as a protocol timer trigger. The content of the write is irrelevant.

8.4.3 Control And Data RAM

Data to be transferred reside in Data RAM, and each 16-bit data halfword has a corresponding 16-bit control halfword in Control RAM with the same address offset. Each data halfword / control halfword pair constitutes a queue entry. There are a total of 64 queue entries. The values in RAM are undefined at Reset and should be explicitly programmed.

8.4.3.1 Control RAM

Only the 7 LSBs (bits 6–0) of each 16-bit queue control halfword are used; the 9 MSBs (bits 15–7) of each control halfword always read 0. The MCU can read and write to control RAM, while the QSPI has read only access.

Control RAM **$0020_5000 to $0020_507F**

	BIT 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT 0
										BYTE	RE	PAUSE	CONT	PCS[2:0]		
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 8-7. QSPI Control RAM Description

Name	Description	Settings
BYTE Bit 6	**BYTE Enable—**This bit controls the width of transferred data halfwords. When BYTE is set, the QSPI transfers only the 8 least significant bits of the corresponding 16-bit queue entry in Data RAM. If receiving is enabled, a received halfword is also 8 bits. The received byte is written to the least significant 8 bits of the data halfword, and the most significant byte of the data halfword is filled with 0s. When BYTE is cleared, the QSPI transfers the full 16 bits of the queue entry's data halfword.	0 = 16-bit data transferred. 1 = 8-bit data transferred.
RE Bit 5	**Receive Enable—**This bit enables or disables data reception by the QSPI. The QSPI enables reception of data from the MISO pin for each queue entry in which the RE bit is set, and writes the received halfword into the data halfword of that queue entry. The received halfword will overwrite the transmitted data that was previously stored in that RAM address.	0 = Receive disabled. 1 = Receive enabled.

Table 8-7. QSPI Control RAM Description (Continued)

<table>
<tr><th>Name</th><th>Description</th><th>Settings</th></tr>
<tr><td>PAUSE
Bit 4</td><td>PAUSE—This bit specifies whether the QSPI pauses after the transfer of a queue entry. When the QSPI identifies an asserted PAUSE bit in a queue entry’s control halfword, the QSPI recognizes that it has reached the end of a queue. After transfer of that queue entry, the QSPI terminates execution of the queue by clearing the associate QX and QA bits in SPSR. It then processes the next activated queue with the highest priority. When the QSPI identifies a cleared PAUSE bit, it proceeds to transfer the next entry in that queue.</td><td>0 = Not a queue boundary.
1 = Queue boundary.</td></tr>
<tr><td>CONT
Bit 3</td><td>Continuous Chip-Select—Specifies if the chip-select line is activated or deactivated between transfers. When the CONT bit is set, the chip-select line continues to be activated between the transfer of the present queue entry and the next one. When the CONT bit is cleared, the chip-select line is deactivated after the transfer of the present queue entry.</td><td>0 = Deactivate chip select.
1 = Keep chip select active.</td></tr>
<tr><td>PCS[2:0]
Bits 2–0</td><td>Peripheral Chip Select Field—Determines the action to be taken at the end of the current queue entry transfer:

SPICn Activated—The specified chip select line is asserted.

NOP—No SPICS line activated. At the end of the current transfer the QSPI deasserts the SPICS lines and waits for a new transfer trigger to resume operation. The queue pointer is set to point to the next queue entry.

EOTIE—End of Transfer interrupt enabled. The value of the PCS field from the previous queue entry determines the SPICS line asserted for this transfer. At the end of the current transfer the QSPI asserts the associated EOT flag in the SPSR and generates an interrupt to the MCU.

EOQ—End of Queue. The QSPI completes the transfer of the current queue entry, clears the QA and QX bits of the current queue, and processes the next active queue with the highest priority. If the LE bit in the QCR of the current queue is asserted, the value in the least significant byte of the data halfword in that queue entry is written into the queue’s QP.</td><td><table>
<tr><th>PCS[2:0]</th><th>QSPI Action</th></tr>
<tr><td>000</td><td>SPIC0 Activated</td></tr>
<tr><td>001</td><td>SPIC1 Activated</td></tr>
<tr><td>010</td><td>SPIC2 Activated</td></tr>
<tr><td>011</td><td>SPIC3 Activated</td></tr>
<tr><td>100</td><td>SPIC4 Activated</td></tr>
<tr><td>101</td><td>NOP[1]</td></tr>
<tr><td>110</td><td>EOTIE</td></tr>
<tr><td>111</td><td>EOQ[1]</td></tr>
</table>
[1] All other bits in the control halfword are disregarded.</td></tr>
</table>

8.4.3.2 Data RAM

Data halfwords can contain transmission data stored in RAM by the MCU or data received by the QSPI from external peripherals. The MCU can read the received data halfwords from RAM. Data is transmitted and received by the QSPI as either least or most significant bit first, depending on the LSBF bit in SCCR for the associated channel. Access to the RAM is arbitrated between the QSPI and the MCU. Because of this arbitration, wait states can be inserted into MCU access times when the QSPI is in operation.

Received data is written to the same address at which the transmitted data is stored and overwrites it, so care must be taken to ensure that no data is lost when receiving is enabled.

8.4.4 GPIO Registers

Any of the eight QSPI pins can function as GPIO. The registers governing GPIO functions are described below.

QPCR QSPI Port Configuration Register **$0020_5F00**

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT 0
									QPC7 (SCK)	QPC6 (MOSI)	QPC5 (MISO)	QPC4 (SPICS4)	QPC3 (SPICS3)	QPC2 (SPICS2)	QPC1 (SPICS1)	QPC0 (SPICS0)
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 8-8. QPCR Description

Name	Description	Settings
QPC[7:0] Bits 7–0	**QSPI Pin Configuration—**Each bit determines whether its associated pin functions as QSPI or GPIO.	0 = GPIO (default). 1 = QSPI.

QDDR — QSPI Data Direction Register — $0020_5F02

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT 0
									QDD7 (SCK)	QDD6 (MOSI)	QDD5 (MISO)	QDD4 (SPICS4)	QDD3 (SPICS3)	QDD2 (SPICS2)	QDD1 (SPICS1)	QDD0 (SPICS0)
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 8-9. QDDR Description

Name	Description	Settings
QDD[7:0] Bits 7–0	**QSPI Data Direction[7:0]—**Determines whether each pin that is configured as GPIO functions as an input or an output, whether or not the QSPI is enabled.	0 = Input (default). 1 = Output.

QPDR — QSPI Port Data Register — $0020_5F04

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	BIT 0
									QPD7 (SCK)	QPD6 (MOSI)	QPD5 (MISO)	QPD4 (SPICS4)	QPD3 (SPICS3)	QPD2 (SPICS2)	QPD1 (SPICS1)	QPD0 (SPICS0)
RESET	0	0	0	0	0	0	0	0	—	—	—	—	—	—	—	—

Table 8-10. QPDR Description

Name	Description
QPD[7:0] Bits 7–0	**QSPI Port GPIO Data [7:0]—**Each of these bits contains data for the corresponding QSPI pin if it is configured as GPIO. Writes to QPDR are stored in an internal latch, and driven on any port pin that is configured as an output. Reads of this register return the value sensed on input pins and the latched data driven on outputs.

Chapter 9
Timers

This section describes three of the four DSP56652 timer modules controlled by the MCU:

- The periodic interval timer (PIT) creates a periodic signal that is used to generate a regularly timed interrupt. It operates in all low power modes.
- The watchdog timer protects against system failures by resetting the DSP56652 if it is not serviced periodically. The watchdog can operate in both WAIT and DOZE low power modes. Its time-out intervals are programmable from 0.5 to 32 seconds (for a 32 kHz input clock).
- The pulse width modulator (PWM) and general purpose (GP) timers run on independent clocks derived from a common MCU_CLK prescaler. The PWM can be used to synthesize waveforms. The GP timers can measure the interval between external events or generate timed signals to trigger external events.

The protocol timer is described in Chapter 10.

9.1 Periodic Interrupt Timer

The PIT is a 16-bit “set-and-forget” timer that provides precise interrupts at regular intervals with minimal processor intervention. The timer can count down either from the maximum value ($FFFF) or the value written in a modulus latch.

9.1.1 PIT Operation

Figure 9-1 shows a block diagram of the PIT.

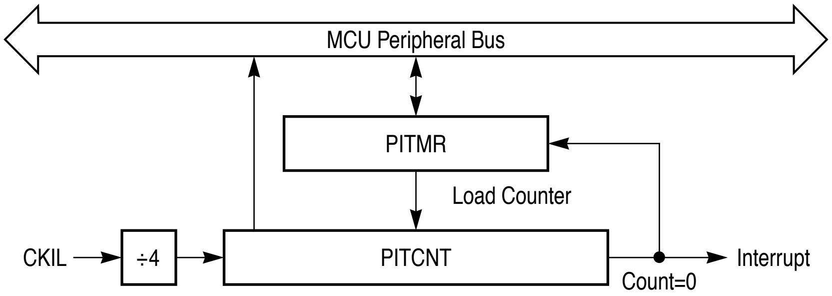


Figure 9-1. PIT Block Diagram

The PIT uses the following registers:

- PITCSR—The Periodic Interrupt Timer Control and Status Register determines whether the counter is loaded with $FFFF or the value in the Module Latch, controls operation in Debug mode, and contains the interrupt enable and flag bits.
- PITMR—The Periodic Interrupt Timer Module Latch contains the rollover value loaded into the counter.
- PITCNT—The Periodic Interrupt Timer Counter reflects the current timer count.

Each cycle of the PIT clock decrements the counter, PITCNT. When PITCNT reaches zero, the ITIF flag in the PITCSR is set. An interrupt is also generated if the ITIE bit in the PITCSR has been set by software. The next tick of the PIT clock loads either $FFFF or the value in the PITMR, depending on the state of the RLD bit in the PITCSR.

The PIT clock is a fixed rate of CKIL/4. Internal clock synchronization logic enables the MCU to read the counter value accurately. This logic requires that the frequency of MCU_CLK, which drives the MCU peripherals, be greater than or equal to CKIL. Therefore, when CKIL drives the MCU clock the division factor should be 1 (i.e., MCS[2:0] in the CKCTL register are cleared—see page 4-5).

Figure 9-2 is a timing diagram of PIT operation using the PITMR to reload the counter.

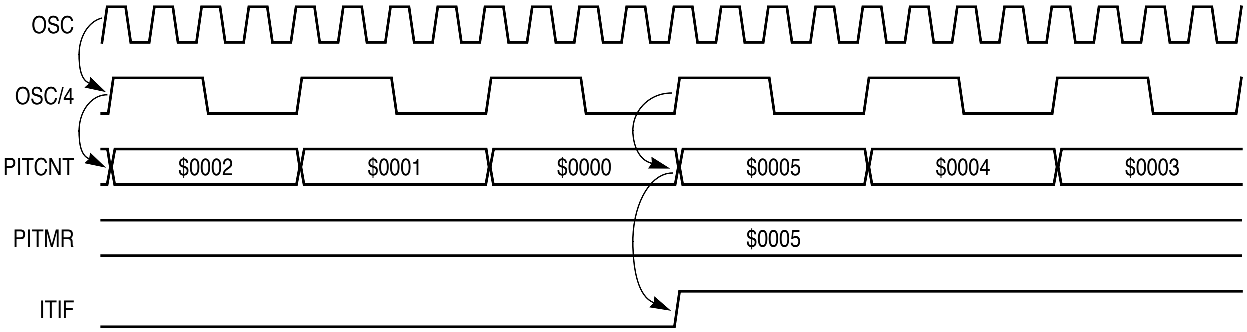


Figure 9-2. PIT Timing Using the PITMR

Setting the OVW bit in the ITCSR enables the counter to be updated at any time. A write to the PITMR register simultaneously writes the same value to PITCNT if OVW is set.

The PIT is not affected by the low power modes. It continues to operate in STOP, DOZE and WAIT modes.

PIT operation can be frozen when the MCU enters Debug mode if the DBG bit in the PITCSR is set. When Debug mode is exited, the timer resumes operation from its state prior to entering Debug mode. If the DBG bit is cleared, the PIT continues to run in Debug mode.

Note: The PIT has no enable control bit. It is always running except in debug mode.

9.1.2 PIT Registers

The following is a bit description of the three PIT registers.

PITCSR PIT Control/Status Register **$0020_7000**

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
											DBG	OVW	ITIE	ITIF	RLD	
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 9-1. ITCSR Description

Name	Type [1]	Description	Settings
DBG Bit 5	R/W	**Debug—**Controls PIT function in Debug mode.	0 = PIT runs normally (default). 1 = PIT is frozen.
OVW Bit 4	R/W	**Counter Overwrite Enable—**Determines if a write to PITMR is simultaneously passed through to PITCNT.	0 = PITMR write does not affect PITCNT (default). 1 = PITMR write immediately overwrites PITCNT.
ITIE Bit 3	R/W	**PIT Interrupt Enable—**Enables an interrupt when ITIF is set. **Note:** Either the EPIT bit in the NIER or the EFPIT bit in the FIER must also be set in order to generate this interrupt (see page 7-7).	0 = Interrupt disabled (default). 1 = Interrupt enabled.
ITIF Bit 2	R/1C	**PIT Interrupt Flag—**Set when the counter value reaches zero; cleared by writing it with 1 or writing to the PITMR.	0 = Counter has not reached zero (default). 1 = Counter has reached zero.
RLD Bit 1	R/W	**Counter Reload—**Determines the value loaded into the counter when it rolls over.	0 = $FFFF (default) 1 = Value in PITMR

[1] R/W = Read/write.
R/1C = Read, or write with 1 to clear (write with 0 ingored).

PITMR PIT Modulus Register **$0020_7002**

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
	PIT Modulus Value															
RESET	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

This register contains the value that is loaded into the PITCNT when it rolls over if the RLD bit in the PITCSR is set. The default value is $FFFF.

PITCNT PIT Counter **$0020_7004**

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
RESET	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

This read-only register provides access to the PIT counter value. The reset value is indeterminate.

9.2 Watchdog Timer

The watchdog timer protects against system failures by providing a means to escape from unexpected events or programming errors. Once the timer is enabled, it must be periodically serviced by software or it will time out and assert the Reset signal.

9.2.1 Watchdog Timer Operation

The watchdog timer uses the following registers:

- WCR—The Watchdog Control Register enables the timer, loads the watchdog counter, and controls operation in Debug and DOZE modes.
- WSR—The Watchdog Service Register is used to reinitialize the timer periodically to prevent it from timing out.

The watchdog timer is disabled at reset. Once it is enabled by setting the WDE bit in the WCR, it cannot be disabled again. The timer contains a 6-bit counter that is initialized to the value in the WT field in the WCR. This counter is decremented by each cycle of the watchdog clock, which runs at a fixed rate of CKIL$\div2^{14}$. Thus, for CKIL=32.768KHz, the watchdog timeout period can range from 0.5 seconds to 32 seconds.

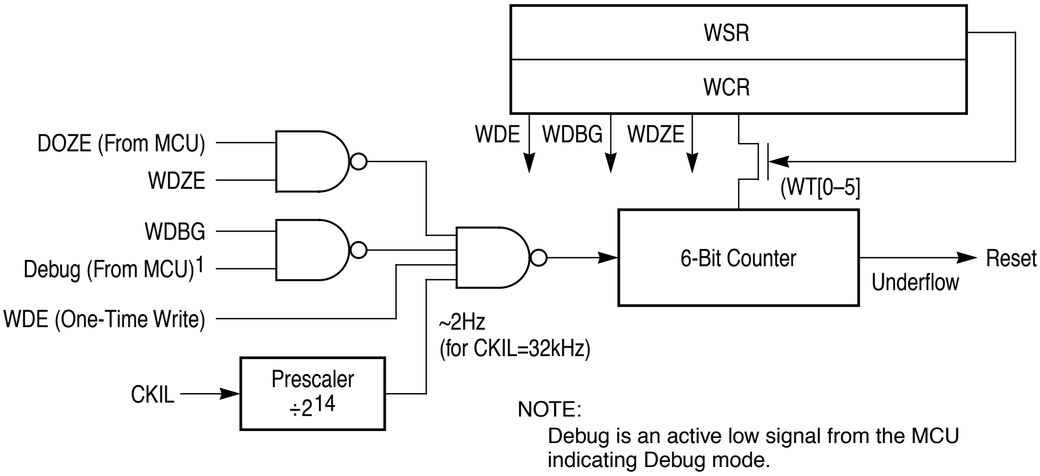
The counter is initialized to the value in the WT field when the watchdog timer is enabled and each time the timer is serviced. The timer must be serviced before the counter rolls over or it will reset the system. The timer can only be serviced by performing the following steps, in sequence:

1. Write $5555 to the WSR.
2. Write $AAAA to the WSR.

Any number of instructions can occur between these two steps. In fact, it is recommended that the steps be in different code sections and not in the same loop. This prevents the MCU from servicing the timer when it is erroneously bound in a loop or code section.

The watchdog timer is subject to the same synchronization logic restrictions as the PIT, i.e., MCU_CLK $\geq$ CKIL.

Figure 9-3 is a block diagram of the Watchdog Timer.



NOTE:
Debug is an active low signal from the MCU indicating Debug mode.

Figure 9-3. Watchdog Timer Block Diagram

The timer is unaffected by WAIT mode and halts in STOP mode. It can either halt or continue to run in DOZE mode, depending on the state of the WDZE bit in the WCR.

In Debug mode, the watchdog timer can either halt or continue to run, depending on the state of the WDBG bit in the WCR. If WDBG is set when the MCU enters Debug mode, the timer stops, register read and write accesses function normally, and the WDE bit one-time-write lock is disabled. If the WDE bit is cleared while in Debug mode, it will remain cleared when Debug mode is exited. If the WDE bit is not cleared while in Debug mode, the watchdog count will continue from its value before Debug mode was entered.

9.2.2 Watchdog Timer Registers

WCR Watchdog Control Register **$0020_8000**

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
	WT[5:0]													WDE	WDBG	WDZE
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 9-2. WCR Description

Name	Description	Settings
WT[5:0] Bits 15–10	**Watchdog Timer Field—**These bits determine the value loaded in the watchdog counter when it is initialized and after the timer is serviced.	
WDE Bit 2	**Watchdog Enable—**Setting this bit enables the watchdog timer. It can only be cleared in Debug mode or by Reset.	0 = Disabled (default). 1 = Enabled.
WDBG Bit 1	**Watchdog Debug Enable—**Determines timer operation in Debug mode.	0 = Continues to run in Debug mode (default) 1 = Halts in Debug mode.
WDZE Bit 0	**Watchdog Doze Enable—**Determines timer operation in DOZE mode.	0 = Continues to run in DOZE mode (default). 1 = Halts in DOZE mode.

WSR Watchdog Service Register **$0020_8002**

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
	WSR[15:0]															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register services the watchdog timer and prevents it from timing out. To service the timer, perform the following steps:

1. Write $5555 to the WSR.
2. Write $AAAA to the WSR.

9.3 GP Timer and PWM

This section describes the MCU GP timer and pulse width modulator (PWM). Although these are separate functions, they derive their clocks from a common 8-bit MCU_CLK divider, shown in Figure 9-4. They also share several control registers.

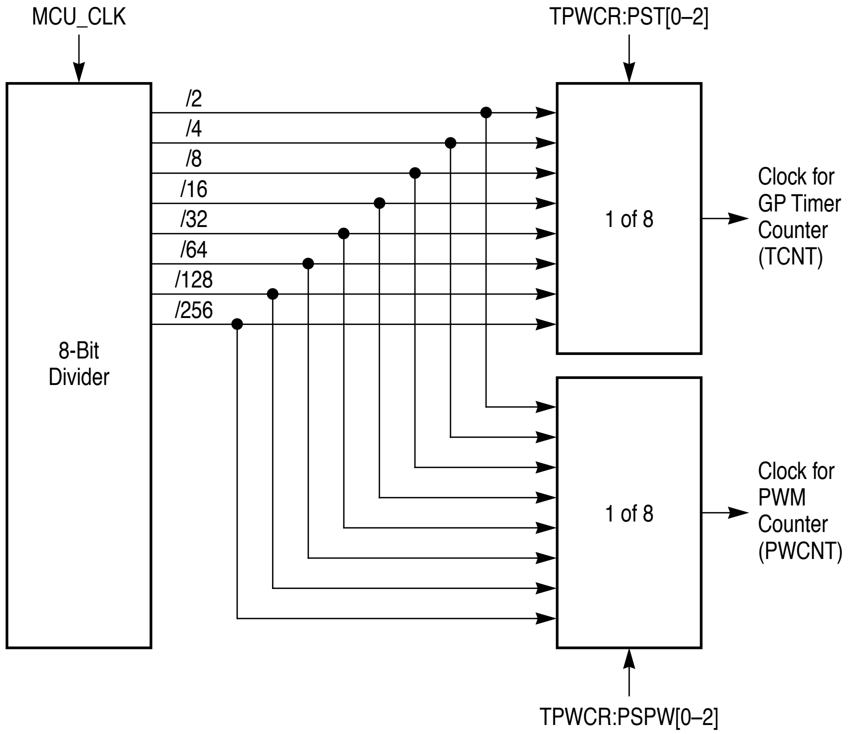


Figure 9-4. GP Timer/PWM Clocks

9.3.1 GP Timer

The GP timer provides two input capture (IC) channels and three output compare (OC) channels. The input capture channels use a 16-bit free-running up counter, TCNT, to record the time of external events indicated by signal transitions on the IC input pins. The output compare channels use the same counter to time the initiation of three different events.

9.3.1.1 GP Timer Operation

The GP timer uses the following registers:

- TPWCR[1]—The Timer Control Register enables the GP timer, selects the TCNT clock frequency, and determines GP timer operation in Debug and DOZE modes.
- TPWMR1—The Timer Mode Register selects the edges that trigger the IC functions, determines the action taken for the OC function, and can force an output compare on any of the OC channels.
- TPWSR1—The Timer Status Register contains flag bits for each IC and OC event and counter rollover.
- TPWIR1—The Timer Interrupt Register enables interrupts for each IC and OC event and counter rollover.
- TICR1,2—The Timer Input Capture Registers latch the TCNT value when the programmed edge occurs on the associated IC input.
- TOCR1,3,4—The Timer Output Compare Registers contain the TCNT values that trigger the programmed OC outputs.
- TCNT—The Timer Counter reflects the current TCNT value.

Figure 9-5 is a block diagram of the GP timer.

All GP timer functions are based on a 16-bit free-running counter, TCNT. The PST[2:0] bits in TPWCR select one of eight possible divisions of MCU_CLK as the clock for TCNT. PST[2:0] can be changed at any time to select a different frequency for the TCNT clock; the change does not take effect until the 8-bit divider rolls over to zero. TCNT begins counting when the TE bit in TPWCR is set. If TE is later cleared, the counter freezes at its current value, and resumes counting from that value when TE is set again. The MCU can read TCNT at any time to get the current value of TCNT.

TCNT is frozen when the MCU enters STOP mode, DOZE mode (if the TD bit in TPWCR is set) or Debug mode (if the TDBG bit in TPWCR is set). In each case, TCNT resumes counting from its frozen value when the respective mode is exited. If TD or TDBG are cleared, entering the associated mode does not affect GP timer operation.

---

[1] These registers also contain bits used by the pulse width modulator.

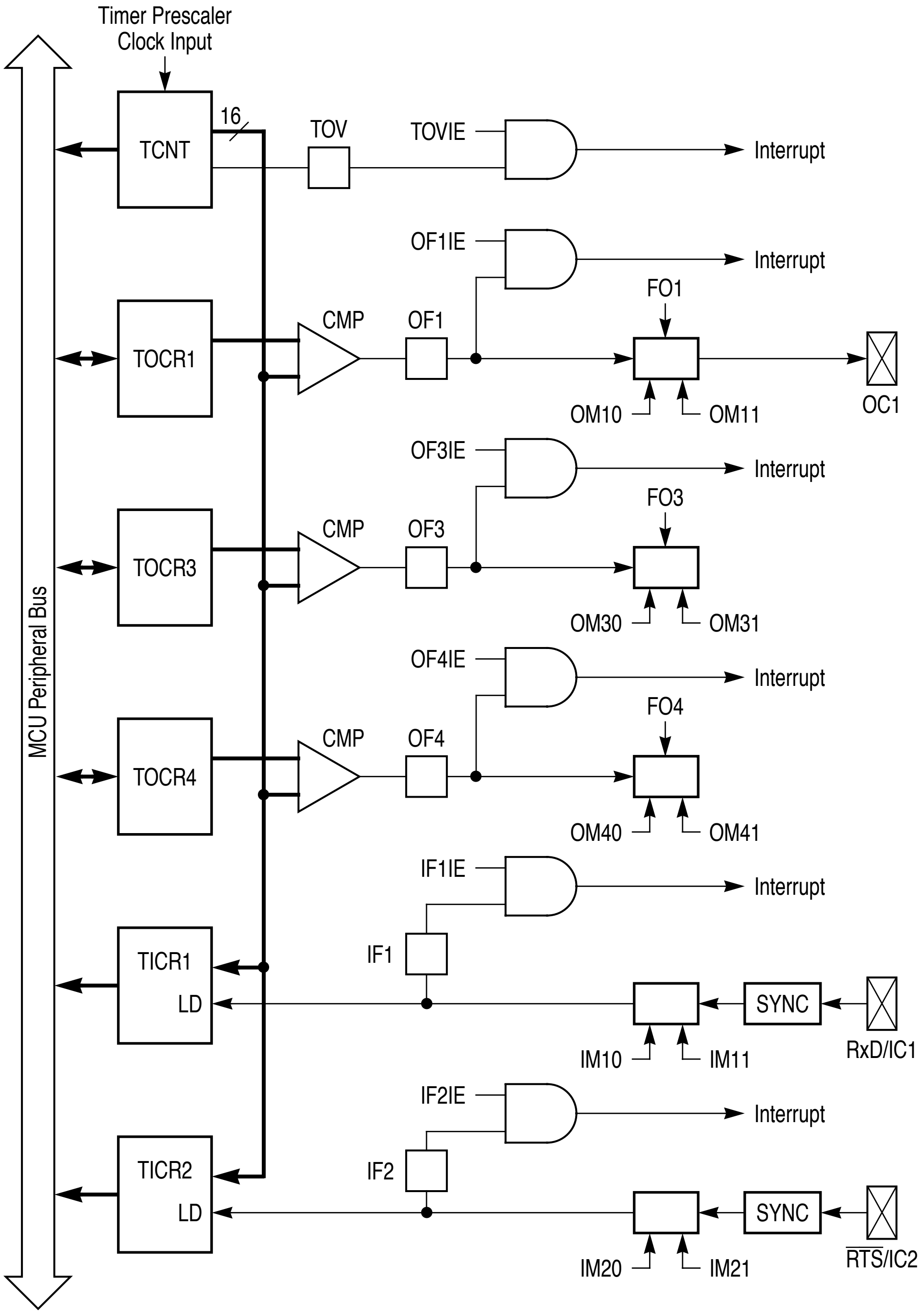


Figure 9-5. GP Timer Block Diagram

9.3.1.1.1 Input Capture

The inputs to IC1 and IC2 are UART pins RxD and $\overline{\text{RTS}}$ respectively. Each input capture pin has a dedicated 16-bit latch (TICR1,2) and input edge detection/selection logic. Each input capture function can be programmed to trigger on the rising edge, falling edge, or both edges of the associated IC pin through the associated IM[1:0] bits in the TPWMR. When the programmed edge transition occurs on an input capture pin, the associated TICR captures the content of TCNT and sets an associated flag bit (IF1,2) in the TPWSR. If the associated interrupt enable bit (IFIE1,2) in TPWIR) has been set, an interrupt request is also generated when the transition is detected. Input capture events are asynchronous to the GP timer counter, so they are conditioned by a synchronizer and a digital filter. The events are synchronized with MCU_CLK so that TCNT is latched on the opposite half-cycle of MCU_CLK from TCNT increment. An input transition shorter than one MCU_CLK period has no effect. A transition longer than two MCU_CLK periods is guaranteed to be captured, with a maximum uncertainty of one MCU_CLK cycle. TICR1 and 2 can be read at any time without affecting their values.

Both input capture registers retain their values during STOP and DOZE modes, and when the GP timer is disabled (TE bit cleared).

9.3.1.1.2 Output Compare

Each output compare channel has an associated compare register (TOCR1,3,4). When TCNT equals the 16-bit value in a compare register, a status flag (OCF1,3,4) in TPWSR is set. If the associated interrupt enable bit (OCIE1,3,4) in TPWIR has been set, an interrupt is generated. OC1 can also set, clear or toggle the OC1 output pin, depending on the state of OM1[1:0] in the TPWMR. OC3 and OC4 are not pinned out but their flags and interrupt enables can be used to time event generation.

The OC1 pin can be forced to its compare value at any time by setting FO1 in the TPWMR. The action taken as a result of a forced compare is the same as when an output compare match occurs, except that status flags are not set. OC3 and OC4 also have forcing bits, but they have no effect because the functions are not pinned out.

9.3.2 Pulse Width Modulator

The pulse width modulator (PWM) uses a 16-bit free-running counter, PWCNT, to generate an output pulse on the PWM pin with a specific period and frequency.

9.3.2.1 PWM Operation

The PWM uses the following registers:

- TPWCR[1]—The PWM Control Register enables the PWM, selects the PWCNT clock frequency, and determines PWM operation in Debug and DOZE modes.
- TPWMR[1]—The PWM Mode Register connects the PWM function to the PWM output pin and determines the output polarity.
- TPWSR[1]—The PWM Status Register contains flag bits indicating pulse assertion (PWCNT=PWOR) and deassertion (PWCNT rolls over).
- TPWIR[1]—The PWM Interrupt Register enables interrupts for each edge of the pulse.
- PWOR—The PWM Output Compare Register contains the PWCNT value that initiates the pulse.
- PWMR—The PWM Modulus Register contains the value loaded into PWCNT when it rolls over. This value determines the pulse period.
- PWCNT—The PWM Counter reflects the current PWCNT value.

Figure 9-6 is a block diagram of the pulse width modulator.

The pulse width modulator is based on a 16-bit free-running down counter, PWCNT. The PSPW[2:0] bits in TPWCR select one of eight possible divisions of MCU_CLK as the clock for PWCNT. PSPW[2:0] can be changed at any time to select a different frequency for the PWCNT clock; the change does not take effect until the 8-bit divider rolls over to zero. When the PWE bit in TPWCR is set, PWCNT is loaded with the value in PWMR and begins counting down. If PWE is later cleared, the counter freezes at its current value. If PWE is set again, PWCNT is reloaded with PWMR and begins counting down. The MCU can read PWCNT at any time to get the current value of PWCNT.

---

[1] These registers also contain bits used by the GP timer.

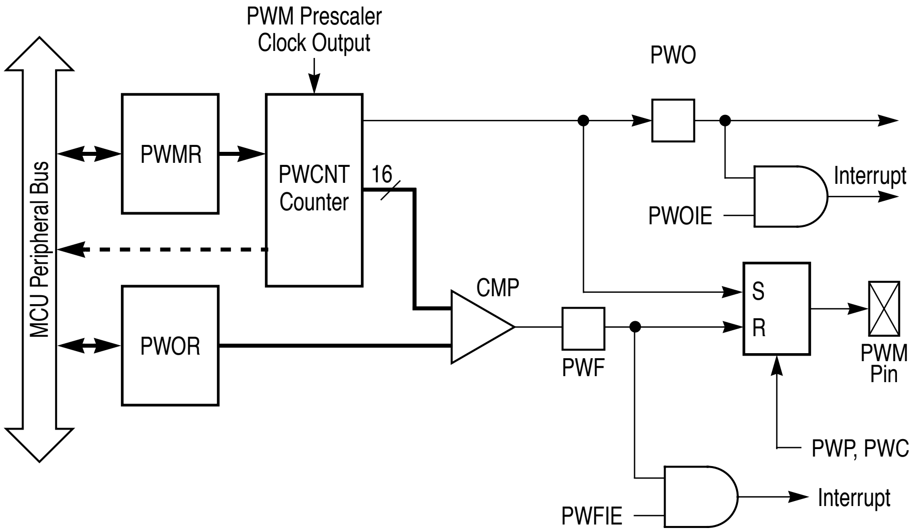


Figure 9-6. PWM Block Diagram

PWCNT is frozen when the MCU enters STOP mode, DOZE mode (if the PWD bit in TPWCR is set) or Debug mode (if the PWDBG bit in TPWCR is set). In each case, PWCNT resumes counting from its frozen value when the respective mode is exited. If PWD or PWDBG are cleared, entering the associated mode does not affect PWM operation.

When PWCNT counts down to the value preprogrammed in the PWOR, the pulse is asserted, and following events occur:

1. The PWF bit in TPWSR is set.
2. An interrupt is generated if the PWFIE bit in TPWCR has been set.
3. If the PWC bit in TPWMR is set, the PWM output pin is driven to its active state, which is determined by the PWP bit in TPWMR.

When PWCNT counts down to zero, the pulse is deasserted, generating the following events:

1. The PWO bit in TPWSR is set.
2. An interrupt is generated if the PWOIE bit in TPWCR has been set.
3. If the PWC bit in TPWMR is set, the PWM output pin is driven to its inactive state.
4. The PWMR value is reloaded to PWCNT.

The pulse duty cycle can range from 0 (PWOR=0) to 99.9985%=65535/65536*100 (PWOR=PWMR=$FFFF). The PWM period can vary between a minimum of 2 MCU_CLK cycles and a maximum of 65536*256 MCU_CLK cycles.

9.3.3 GP Timer and PWM Registers

TPWCR Timer and PWM Control Register **$0020_6000**

	Bits 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
	▨	▨	▨	▨	PWDBG	TDBG	PWD	PWE	TD	TE	PSPW[2:0]			PST[2:0]		
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 9-3. TPWCR Description

Name	Description	Settings
PWDBG Bit 11	**PWM Debug—**Enables PWM operation during Debug mode.	0 = PWM frozen in Debug mode (default). 1 = PWM runs in Debug mode.
TDBG Bit 10	**GP Timer DBG—**Enables IC and OC operation during Debug mode.	0 = GP timer frozen in Debug mode (default). 1 = GP timer runs in Debug mode.
PWD Bit 9	**PWM DOZE—**Enables PWM operation during DOZE mode.	0 = PWM enabled in DOZE mode (default). 1 = PWM disabled in DOZE mode.
PWE Bit 8	**PWM Enable—**Enables PWM operation. If PWE and TE are both cleared, the prescaler is stopped.	0 = PWM disabled; PWCNT stopped (default). 1 = PWM enabled; PWCNT is running.
TD Bit 7	**GP Timer DOZE—**Enables IC and OC operation during DOZE mode.	0 = GP timer enabled in DOZE mode (default). 1 = GP timer disabled in DOZE mode.
TE Bit 6	**GP Timer Enable—**Enables IC and OC operation. If PWE and TE are both cleared, the prescaler is stopped.	0 = IC and OC disabled; TCNT stopped (default). 1 = IC and OC enabled; TCNT is running.
PSPW[2:0] Bits 5–3	**Prescaler for PWM—**These bits select the MCU_CLK divisor for the clock that drives PWCNT.	
PST[2:0] Bits 2–0	**Prescaler for GP Timers—**These bits select the MCU_CLK divisor for the clock that drives TCNT.	

PSPW[2:0] PST[2:0]	PWCNT Prescaler TCNT Prescaler
000	2^1 (default)
001	2^2
010	2^3
011	2^4
100	2^5
101	2^6
110	2^7
111	2^8

TPWMR Timers and PWM Mode Register **$0020_6002**

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
		PWC	PWP	FO4	FO3	FO1	IM2[1:0]		IM1[1:0]						OM1[1:0]	
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 9-4. TPWMR Description

Name	Description	Settings
PWC Bit 14	**PWM Control—**Connects the PWM function to the PWM output pin.	0 = Disconnected (default). 1 = Connected.
PWP Bit 13	**PWM Pin Polarity—**Controls the polarity of the PWM output during the active time of the pulse, defined as time between output compare and PWCNT rollover.	0 = Active-high polarity (default). 1 = Active-low polarity.
FO4 Bit 12 **FO3** Bit 11 **FO1** Bit 10	**Forced Output Compare—**Writing 1 to FOC1 immediately forces the OC1 pin to the output compare state programmed in the associated OM1[1:0] bits. The OF1 flag in TPWSR is not affected. Setting FOC3 and FOCC4 have no effect because these functions are not pinned out. Each FOC bit is self-negating, i.e., always reads 0. Writing 0 to these bits has no effect.	
IM2[1:0] Bits 9–8 **IM1[1:0]** Bits 7–6	**Input Capture Operating Mode—**Each pair of bits determines the input signal edge that triggers the associated input compare response.	00 = Disabled (default). 01 = Rising edge. 10 = Falling edge. 11 = Both edges.
OM1[1:0] Bits 1–0	These bits determine the OC1 output response when the compare 1 function is triggered.	00 = Timer disconnected from pin (default). 01 = Toggle output. 10 = Clear output. 11 = Set output.

TPWSR Timers and PWM Status Register **$0020_6004**

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
									PWO	TOV	PWF	IF2	IF1	OF4	OF3	OF1
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Each of the bits in this register is cleared by writing it with 1. Writing zero to a bit has no effect.

Table 9-5. TPWSR Description

Name	Description	Settings
PWO Bit 7	**PWM Count Rollover—**Indicates if PWCNT has rolled over.	0 = PWCNT has not rolled over (default) 1 = PWCNT has rolled over since PWO was last cleared
TOV Bit 6	**Timer Count Overflow—**Indicates if TCNT has overflowed.	0 = TCNT has not overflowed (default) 1 = TCNT has overflowed since TOV was last cleared
PWF Bit 5	**PWM Output Compare Flag—**Indicates whether the PWM compare occurred.	0 = PWM compare has not occurred (default) 1 = PWM compare has occurred since PWF was last cleared
IF2 Bit 4 **IF1** Bit 3	**Input Capture Flags—**Each bit indicates that the associated input capture function has occurred	0 = Capture has not occurred (default) 1 = Capture has occurred since IF bit was last cleared
OF4 Bit 2 **OF3** Bit 1 **OF1** Bit 0	**Output Compare Flags—**Each bit indicates that the associated output compare function has occurred	0 = Compare has not occurred (default) 1 = Compare has occurred since OF bit was last cleared

TPWIR Timers and PWM Interrupt Enable Register **$0020_0006**

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
	▨	▨	▨	▨	▨	▨	▨	▨	PWOIE	TOVIE	PWFIE	IF2IE	IF1IE	OF4IE	OF3IE	OF1IE
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Note: Either the ETPW bit in the NIER or the EFTPW bit in the FIER must be set in order to generate any of the interrupts enabled in the TPWIR (see page 7-7).

Table 9-6. GNRC Description

Name	Description	Settings
PWOIE Bit 7	**PWM Count Rollover Interrupt Enable**	0 = Interrupt disabled (default) 1 = Interrupt generated when corresponding TPWSR flag bit is set
TOVIE Bit 6	**Timer Count Overflow Interrupt Enable**	
PWFIE Bit 5	**PWM Output Compare Flag Interrupt Enable**	
IF2IE Bit 4	**Input Capture 2 Interrupt Enable**	
IF1IE Bit 3	**Input Capture 1 Interrupt Enable**	
OF4IE Bit 2	**Output Compare 4 Interrupt Enable**	
OF3IE Bit 1	**Output Compare 3 Interrupt Enable**	
OF1IE Bit 0	**Output Compare 1 Interrupt Enable**	

TOCR1 Output Compare 1 Register **$0020_6008**
TOCR3 Output Compare 3 Register **$0020_600A**
TOCR4 Output Compare 4 Register **$0020_600C**

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
	TOCRn[15:0]															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

When TCNT equals the value stored in one of these registers, the corresponding output compare function is triggered.

TICR1 Input Capture 1 Register **$0020_600E**
TICR2 Input Capture 2 Register **$0020_6010**

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
	TICRn[15:0]															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

When TCNT equals the value stored in one of these registers, the corresponding input compare function is triggered.

PWOR PWM Output Compare Register $0020_6012

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
	PWOR[15:0]															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

When PWCNT equals the value written to this register, the pulse is initiated.

TCNT Timer Counter $0020_6014

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
	TCNT[15:0]															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This read-only register reflects the value of the GP timer counter, TCNT.

PWMR PWM Modulus Register $0020_6016

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
	PWMR[15:0]															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The value written to this register is loaded into the PWCNT when the PWM is enabled and each time PWCNT rolls over. The PWCNT roll-over period equals the value loaded + 1.

PWCNT PWM Counter $0020_6018

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
	PWCNT[15:0]															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This read-only register reflects the value of the PWM counter, PWCNT.

Chapter 10
Protocol Timer

The Protocol Timer (PT) serves as the control module for all radio channel timing. It relieves the MCU from the event scheduling associated with radio communication protocol so that software need only reprogram the PT once per frame or less. The events the PT can generate include the following:

- **QSPI triggers** can be used to program external devices that have SPI ports.
- **External events** driven on the PT pins TOUT[7–0] can be used to control external devices.
- **MCU and DSP interrupts** can be used in a variety of ways, for example to alert the cores to prepare for a change to a different channel or slot.
- **Transmit and Receive Macros** with programmable delays generate repeating event sequences with a single event call. A transmit and receive macro can run simultaneously.
- **Control events** governing PT operation and synchronization.

Each of these events can be represented by an event code in the protocol timer’s event table. Each entry that contains an event code is paired with a Time Interval Count (**TIC**) value. The entries are written in order of decreasing TIC value. As the value in a down counter matches each TIC value, an event represented by the corresponding event code is generated. The result is a series of events with specific timing and sequence.

10.1 Protocol Timer Architecture

This section describes the PT functional blocks, including the timing components, event table, and event generation hardware.

A block diagram of the PT is shown in Figure 10-1.

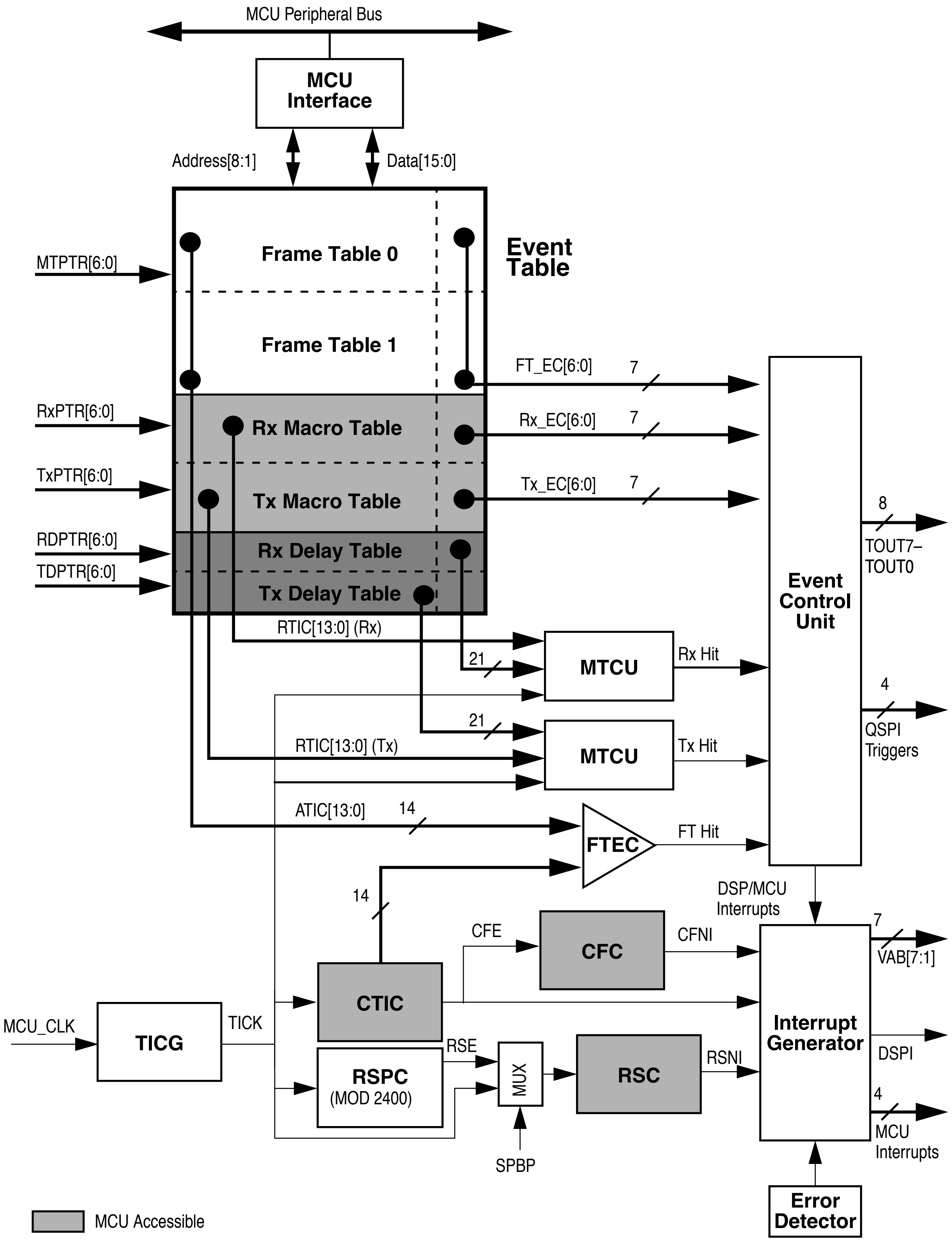


Figure 10-1. Protocol Timer Block Diagram

10.1.1 Timing Signals and Components

The Time Interval Clock Generator (TICG) generates the primary timing PT reference signal, the Time Interval Clock (TICK). This signal is related to symbol duration, and typically functions as a sub-symbol clock.

TICK drives two timing chains. The primary timing chain generates event timing. It contains a Channel Time Interval Counter (CTIC) which drives a Channel Frame Counter (**CFC**). The primary chain has a programmable modulus. The auxiliary chain, which has a fixed modulus, is used as a time slot reference. This chain contains a Reference Slot Prescale Counter (RSPC) which drives a Reference Slot Counter (RSC).

10.1.1.1 Time Interval Clock Generator

The TICG is a 9-bit programmable prescaler that divides MCU_CLK to generate the PT reference clock, TICK. The TICK frequency range is MCU_CLK/2 to MCU_CLK/512. The TICG modulus value is programmed in the Time Interval Modulus Register (TIMR), which is loaded into the TICG when it rolls over. Changing the TICG value “on the fly” is not supported.

10.1.1.2 Channel Time Interval Counter

The CTIC is a programmable read/write, free-running 14-bit modulo down counter decremented by the TICK signal. It is used to trigger frame table events and generate the frame reference signal Channel Frame Expire (CFE**).** An event is triggered each time the value in CTIC matches the TIC value pointed to in a Frame Table. CFE is asserted when the CTIC decrements to zero, which can trigger a Channel Frame Interrupt (CFI) to the MCU if the CFIE bit in the Protocol Timer Interrupt Enable Register (PTIER) is set. CTIC rolls over to a modulo value contained in the Channel Time Interval Modulus Register (CTIMR), which is usually the number of TICKs in a radio channel frame.

The PT can be synchronized to radio channel timing by reloading CTIC at a specific time. This can be done either by writing CTIC directly or writing a new value to CTIMR (if needed) and generating a reload_counter event.

10.1.1.3 Channel Frame Counter

The CFC is a programmable read/write, free-running 9-bit modulo down counter decremented by the CFE signal. It is used to count channel frames. If the CFNIE bit in the PTIER is set, the CFC generates a Channel Frame Number Interrupt (CFNI) when it decrements to zero The CFC rolls over to a modulo value contained in the Channel Frame Modulus Register (CFMR).

10.1.1.4 Reference Slot Prescale Counter

The RSPC is 12-bit free running modulo 2400 down counter decremented by TICK. The output of the counter is a slot reference signal, Reference Slot Expire (RSE), which drives the RSC. Systems that do not need this modulo 2400 divider can bypass the RSPC by setting the SPBP bit in the Protocol Timer Control Register (PTCR), so that TICK drives the RSC directly.

10.1.1.5 Reference Slot Counter

The RSC is a programmable 8-bit read/write free-running down counter decremented by RSE. It can be used, for example, to keep track of slot timing in an adjacent cell. If the RSNIE bit in the PTIER is set when the RSC decrements to zero, a Reference Slot Number Interrupt (RSNI) is generated. The RSC rolls over to a modulo value contained in the Reference Slot Modulus Register (RSMR).

10.1.2 Event Table

The event table is an 80-word dual-port RAM starting at the base of the protocol timer peripheral space, $0020_3000. Each entry contains a 14-bit field and a 7-bit field; all fields are halfword-aligned. The event table can be dynamically partitioned into two frame tables, two macro tables and two delay tables by initializing the base address registers FTBAR, MTBAR, and DTPTR respectively. A frame table, a receive macro, and a transmit macro can all be active simultaneously. Figure 10-2 shows the structure of the event table.

Note: The base address and pointer registers contain entry numbers. The actual address in MCU memory is equal to $0020_3000 plus 4 times the entry number.

The MCU can read and write the event table, whether or not the PT is enabled. PT control logic has read-only access to the event table. Arbitration logic ensures that the event table is accessed correctly, adding wait states to MCU cycles when necessary.

10.1.3 Event Generation

The components involved in generating events in the PT include a Frame Table Event Comparator, two Macro Timing Control Units, an Event Control Unit, and an Interrupt Generator.

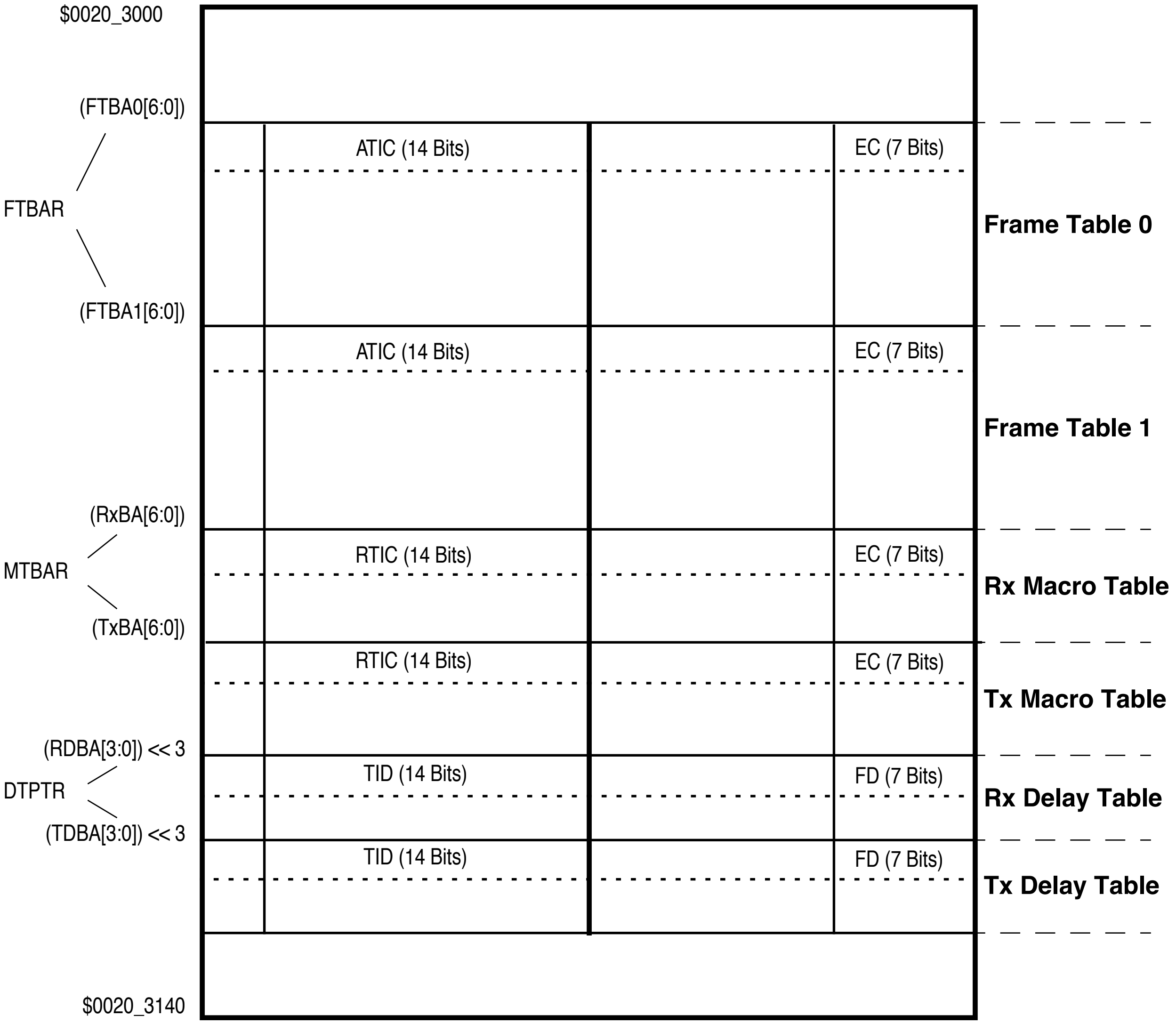


Figure 10-2. Event Table Structure

The Frame Table Event Comparator (FTEC) fetches the Absolute Time Interval Count (ATIC) in the Frame Table entry pointed to by the Frame Table Pointer Register (FTPTR). The FTEC compares its ATIC value with the current value of CTIC. When the values match, the FTEC generates an internal signal, FT Hit, initiating activity corresponding to the entry’s event code. The pointer is then incremented to the next entry in the table.

There are two Macro Timing Control Units (MTCUs), one each for the receive macro and the transmit macro. The MTCU for the receive macro loads a down counter with the relative time interval count (RTIC) in the entry in the Receive Macro Table pointed to by the Receive Macro Table Pointer (RxPTR) field in the Macro Table Pointer Register (MTPTR). When the counter reaches zero, the receive MTCU generates an internal signal, Rx Hit, initiating activity corresponding to the entry's event code. The pointer is then incremented to the next entry in the table. In similar fashion, the transmit macro uses the Transmit Macro Table Pointer (TxPTR) in MTPTR to generate Tx Hit.

The Event Control Unit (ECU) responds to FT Hit, Tx Hit, or Rx Hit by reading the event code (EC) associated with the table entry that generated the hit. The ECU decodes the EC and initiates one of the following events:

- Force one of the eight TOUT pins high or low.
- Issue one of the four QSPI triggers.
- Control event table sequencing.
- Alert the interrupt controller to generate one of these interrupts:
  - one of the three MCU interrupts.
  - DSP interrupt (DSP $\overline{\text{IRQD}}$.
  - one of the sixteen DSP vector interrupts.

In addition, the ECU can initiate a Transmit or Receive Macro. (A macro cannot initiate another macro.)

The Interrupt Controller receives inputs from the ECU, CFC, RSC, and Error Detector to generate the appropriate interrupt. Error detection is described in Section 10.2.4 on page 10-11. Interrupts are detailed in Section 10.2.5 on page 10-11.

10.2 PT Operation

This section describes all aspects of PT operation, including sequencing and generating events within a frame and in the transmit and receive macros, the various PT operating modes, error detection, and a summary of the interrupts generated by the PT.

10.2.1 Frame Events

The PT provides two frame tables to contain the primary lists of events to be triggered. The base addresses of these tables are stored in the Frame Table Base Address Register (FTBAR). Each entry in a frame table has a 14-bit Absolute TIC field and a 7-bit Event Code field, as shown in Figure 10-3.

Only one of the frame tables is active at a given time; the inactive table can be updated for later use. The active table can be switched by encoding an end_of_frame_switch or table_change command. If the active table is Frame Table 1, it can be switched to Frame Table 0 with the end_of_frame_halt command.

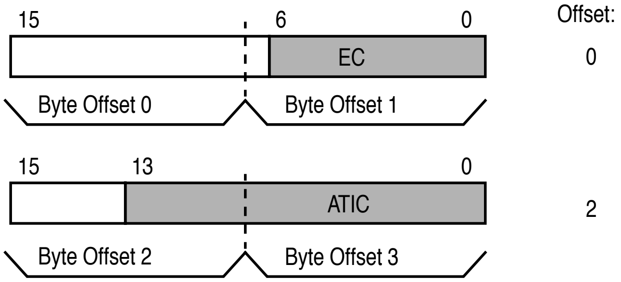


Figure 10-3. Frame Table Entry

Frame table entries are subject to the following restrictions:

1. All entries in each frame table must be in sequential order, i.e., with decreasing ATIC fields.
2. The ATIC value of each entry in a frame table must be less than the CTIC modulus, CTIMR.
3. Only one event can be scheduled per ATIC.
4. An end_of_frame command must be executed before CTIC rolls over.
5. The delay and end_of_macro events are for macros only.
6. Writing to a frame table entry that is currently being executed can generate erratic results. To guard against this possibility, MCU software can be written so as not to write to the active frame table.

When the protocol timer is enabled or exits the HALT state, FTPTR is initialized to the first entry in frame table 0 (the FTBA0 field in FTBAR). When the value in CTIC matches the ATIC field pointed to by FTPTR, the FTEC asserts an internal Frame Hit signal to the ECU, which generates the event specified by the EC field of the FTPTR entry. FTPTR is then incremented. The cycle repeats until one of the end_of_frame commands or the table_ change command is executed. Each of these commands reinitializes FTPTR to the first entry of one of the frame tables.

10.2.2 Macro Tables

The protocol timer can generate a separate, independent sequence of events for both a transmission burst and a receive burst. Both of these sequences, or macros, can run concurrently with the basic frame table sequence. Each of the macros occupies a partition in the event table referred to as a macro table. Each macro is called as an event from the frame table. In most cases, the transmit and receive macro tables only need to be written at initialization, providing a substantial reduction in MCU overhead.

Unlike frame table events, which are based on the absolute value in CTIC, macro events are timed relative to the previous macro event. Each entry in a macro table has a 14-bit Relative TIC field and a 7-bit Event Code field, as shown in Figure 10-4. The RTIC value represents the delay, in timer intervals, from the previous macro event (or from the macro call for the first macro event) to the event specified in the EC field. An RTIC value of 0 or 1 generates the event at the next time interval.

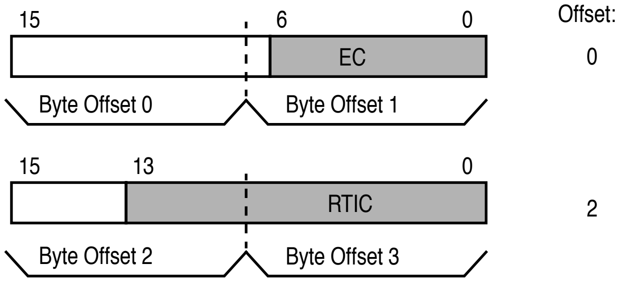


Figure 10-4. Macro Table Entry

When a receive macro is called, RxPTR is initialized to the first entry in the receive macro table. The address of this first entry is contained in the RxBAR field in the Macro Table Base Address Register (MTBAR). The RTIC value of this first entry is loaded into a 14-bit down counter in the receive MTCU. When this counter, decremented by the TICK signal, reaches zero, the MTCU asserts an internal Rx Hit signal to the ECU, which generates the event signal specified by the EC field of the macro pointer entry. The macro pointer is incremented, and the cycle repeats until an end_of_macro command is executed.

The transmit macro operates in similar fashion. The base address of the transmit macro table is stored in the TxBAR field in MTBAR. The TxPTR field in MTPTR is the address pointer. A transmit MTCU generates an internal Tx Hit signal to the ECU.

Macro table entries are subject to the following restrictions:

1. A macro cannot invoke another macro (i.e., macros cannot be nested).
2. Commands that affect frame table operation, which include all end_of_frame commands and the table_change command, are for frame tables only.
3. The last entry in a macro must be the end_of_macro command.

10.2.2.1 Delay Event

The delay event invokes a programmed delay of a specified number of frames and time intervals before the next command in the macro is executed. This event is only valid in macros, and cannot appear in the frame tables.

There are actually eight event codes for invoking the receive macro and eight for the transmit macro. Each of these event codes specifies a different entry in the receive or transmit delay table to be used when the macro calls a delay event. Each delay table entry contains a 7-bit frame delay (FD) and a 14-bit time interval delay (TID), as shown in Figure 10-5.

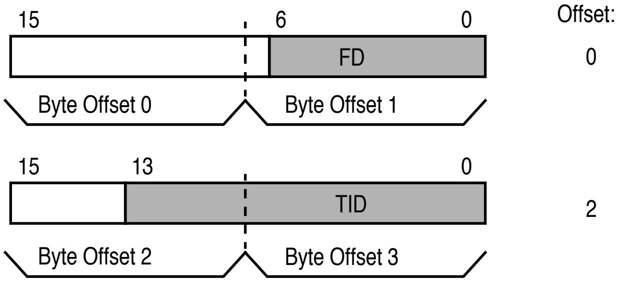


Figure 10-5. Delay Table Entry

When a receive macro is called, the delay index (0 through 7) determined by the particular event code used for the call is loaded into the Receive Delay Pointer (RDPTR) field in the Delay Table Pointer (DTPTR). This number represents the offset from the Receive Delay Table Base Address (RDBA), encoded in the DTPTR at initialization. Thus, DTPTR points to a specific number of frame delays and time interval delays invoked each time the macro uses the delay command. For example, if a frame table entry calls Rx_macro2, the TID and the FD are read from the third entry of the receive delay table. When this macro calls a delay, the event after it is delayed by a total of

$$[\ (FD * (\text{time intervals per frame})) + TID\ ] \quad \text{time intervals.}$$

The transmit macro works in similar fashion using the TDBA and TDPTR fields in DTPTR to point to an entry in the transmit delay table.

10.2.3 Operating Modes

The PT provides control bits to determine enable, halt, and low power operation. The various operating modes are summarized in Table 10-1.

Table 10-1. Protocol Timer Operation Mode Summary

Mode	Description	Activity		Entry to Mode	Exit from Mode
		Clocks and Counters	Event Execution		
Disabled	Timer disabled; GPIO activity only	disabled	disabled	TE=0	TE=1
Normal	Full PT operation	enabled	enabled	TE=1	TE=0
HALT	PT enters HALT state	enabled	disabled	Set HLTR bit or end_of_frame_ halt command	Clear THS and HLTR bits
DOZE, TDZD=0	MCU enters DOZE mode with peripheral active.	enabled	enabled	MCU enters DOZE mode	MCU exits DOZE mode
DOZE, TDZD=1	MCU enters DOZE mode with peripheral stop	disabled	disabled		
STOP	MCU in STOP mode	disabled	disabled	MCU enters STOP mode	MCU exits STOP mode

10.2.3.1 Enabling the PT

The PT is enabled by setting the TE bit in PTCR. If the TIME bit in PTCR is set, the PT is enabled immediately; if TIME is cleared, PT operation starts at the first CFE after TE is set. The TIME bit should only be changed while the PT is disabled (TE cleared).

10.2.3.2 Halting the PT

PT event execution can be halted in one of two ways:

1. Executing the end_of_frame_halt command at the end of a table. Frame table event execution stops immediately.
2. Setting the HLTR bit in PTCR . Frame table event execution continues until one of the end_of_frame commands (event codes $7A–$7C) is executed.

In either event, the THIP bit in the PTIER is set to indicate that the PT is in the process of halting.

Note: The PTCR should not be written while a halt is in process, or erratic behavior can result.

If the MTER bit in PTCR is set, macro activity stops immediately after the end_of_frame event is executed. If MTER is cleared, macro activity continues until the end_of_macro command. When all PT activity has finished, the THS bit in PTSR is set to indicate that the PT is in halt mode. A timer halt interrupt is asserted if the THIE in PTIER is set.

During halt mode, the PT counters and registers remain active. The PT remains in halt mode until the THS bit is cleared by writing it with 1. Event table execution resumes at the beginning of frame table 0.

10.2.3.3 PT Operation in Low Power Modes

The PT remains active in MCU WAIT mode, and also in DOZE mode if the TDZD bit in TCTR is cleared. When the MCU enters STOP mode (or DOZE mode if TDZD set), PT activity immediately stops, and all PT counters and registers are frozen.

For proper PT operation, the following steps should be taken before entering DOZE mode (when the TDZD bit in the PTCR is set) or STOP mode:

1. Halt the PT with an end_of_frame_halt command or by setting HLTR.
2. Wait for THS to be asserted.
3. Disable the PT by clearing TE.

When the MCU wakes up, software must reenable the PT by setting the TE bit.

10.2.4 Error Detection

The PT's error detector monitors for three types of error during PT activity. It sets a bit in the PTSR when an error is detected, and generates a Protocol Timer Error Interrupt (TERI) if the TERIE bit in PTIER is set. These errors include:

- End Of Frame Error. A CFE has occurred but the timer has not sequenced through one of the end of frame commands (EC = $7A–$7C). EOFE is set.
- Macro Being Used Error. A frame table calls a macro that is already active. MBUE is set.
- Pin Contention Error. Contradicting values drive a PT output pin during the same Time Interval. PCE is set.

10.2.5 Interrupts

Table 10-2 is a summary of the interrupts generated by the PT.

Table 10-2. Protocol Timer Interrupt Sources

Acronym	Name	Source
CFI	Channel Frame Interrupt	Channel Frame Expire (CFE) signal (CTIC output)
CFN	Channel Frame Number Interrupt	Channel Frame Counter (CFC) expires
RSNI	Reference Slot Number Interrupt	Reference Slot Counter (RSC) expires
MCUI0 MCUI1 MCUI2	MCU Interrupt 0 MCU Interrupt 1 MCU Interrupt 2	mcu_int0 event mcu_int1 event mcu_int2 event
DSPI	DSP Interrupt	dsp_int event
DVI0 DVI1 ...... DVI15	DSP Vector Interrupt 0 DSP Vector Interrupt 1 . . . . . DSP Vector Interrupt 15	CVR0 event CVR1 event . . . . . CVR15 event
TERI	Timer Error Interrupt	End of Frame Error (EOFE) Macro Being Used Error (MBUE) Pin Contention Error (PCE)
THI	Timer Halt Interrupt	end_of_frame_halt command HLTR bit in PTCR set

The PT interrupt generator provides four outputs to the MCU interrupt controller. Each of the first three is dedicated to a single interrupt source: MCUI0, MCUI1 and MCUI2. The fourth output is a logical OR combination of DVI, CFI, CFNI, RSNI, TERI and THI.

Note: To enable the reception of CFI, CFNI, and RSNI during a halt state, the THIE bit in the PTIER should be cleared after the PT is halted.

The PT provides for 16 DSP vectored interrupts (DVIs) through the CVR15–0 events, each of which specifies its own DSP vector addresses on VAB[7–0]. Another event, dsp_int, affects the DSP indirectly by generating DSP $\overline{\text{IRQD}}$ through the MDI. Refer to the description of the MTIR bit in the MSR on page 5-21. Dsp_irq differs from the CVR events in that it can wake the DSP from STOP mode.

10.2.6 General Purpose Input/Output (GPIO)

Any of the eight PT output pins TOUT7–0 can be configured as GPIO. GPIO functionality is determined by three registers:

- The Protocol Timer Port Control Register (PTPCR) determines which pins are GPIO and which function as PT pins.
- The Protocol Timer Direction Register (PTDDR) configures each GPIO pin as either an input or output

- The Protocol Timer Port Data Register (PTPDR) contains input data from GPI pins and data to be driven on GPO pins.

GPIO register functions are summarized in Table 10-3.

Table 10-3. PT Port Pin Assignment

PTPCR[i]	PTDDR[i]	Port Pin[i] Function
1	X	Protocol Timer
0	0	GP input
0	1	GP output

10.3 PT Event Codes

Table 10-4 lists the 128 possible PT events and their corresponding event codes.

Table 10-4. Protocol Timer Event List

Event Name	Event Code	Description
Tx_macro0[1]	$00	Start Tx macro with delay 0
Tx_macro1	$01	Start Tx macro with delay 1
Tx_macro2	$02	Start Tx macro with delay 2
Tx_macro3	$03	Start Tx macro with delay 3
Tx_macro4	$04	Start Tx macro with delay 4
Tx_macro5	$05	Start Tx macro with delay 5
Tx_macro6	$06	Start Tx macro with delay 6
Tx_macro7	$07	Start Tx macro with delay 7
Rx_macro0	$08	Start Rx macro with delay 0
Rx_macro1	$09	Start Rx macro with delay 1
Rx_macro2	$0A	Start Rx macro with delay 2
Rx_macro3	$0B	Start Rx macro with delay 3
Rx_macro4	$0C	Start Rx macro with delay 4
Rx_macro5	$0D	Start Rx macro with delay 5
Rx_macro6	$0E	Start Rx macro with delay 6
Rx_macro7	$0F	Start Rx macro with delay 7
Negate_Tout0[2]	$10	Tout0 = 0
Assert_Tout0	$11	Tout0 = 1
Negate_Tout	$12	Tout1 = 0
Assert_Tout1	$13	Tout1 = 1

Table 10-4. Protocol Timer Event List (Continued)

Event Name	Event Code	Description
Negate_Tout2	$14	Tout2 = 0
Assert_Tout2	$15	Tout2 = 1
Negate_Tout3	$16	Tout3 = 0
Assert_Tout3	$17	Tout3 = 1
Negate_Tout4	$18	Tout4 = 0
Assert_Tout4	$19	Tout4 = 1
Negate_Tout5	$1A	Tout5 = 0
Assert_Tout5	$1B	Tout5 = 1
Negate_Tout6	$1C	Tout6 = 0
Assert_Tout6	$1D	Tout6 = 1
Negate_Tout7	$1E	Tout7 = 0
Assert_Tout7	$1F	Tout7 = 1
reserved	$2F-$20	Reserved for future use
Trigger0	$30	Activate QSPI Trigger 0
Trigger1	$31	Activate QSPI Trigger 1
Trigger2	$32	Activate QSPI Trigger 2
Trigger3	$33	Activate QSPI Trigger 3
reserved	$3F-$34	Reserved for future use
CVR0	$40	DSP vector Interrupt 0
CVR1	$41	DSP vector Interrupt 1
CVR2	$42	DSP vector Interrupt 2
CVR3	$43	DSP vector Interrupt 3
CVR4	$44	DSP vector Interrupt 4
CVR5	$45	DSP vector Interrupt 5
CVR6	$46	DSP vector Interrupt 6
CVR7	$47	DSP vector Interrupt 7
CVR8	$48	DSP vector Interrupt 8
CVR9	$49	DSP vector Interrupt 9
CVR10	$4A	DSP vector Interrupt 10
CVR11	$4B	DSP vector Interrupt 11
CVR12	$4C	DSP vector Interrupt 12
CVR13	$4D	DSP vector Interrupt 13
CVR14	$4E	DSP vector Interrupt 14
CVR15	$4F	DSP vector Interrupt 15
reserved	$57-50	Reserved for future use

Table 10-4. Protocol Timer Event List (Continued)

Event Name	Event Code	Description
mcu_int0	$58	Assert MCUINT0 signal
mcu_int1	$59	Assert MCUINT1signal
mcu_int2	$5A	Assert MCUINT2 signal
reserved	$5B-$5F	Reserved for future use
dsp_int	$60	Assert DSPINT signal
reserved	$77-61	Reserved for future use
reload_counter	$78	Load CTIMR register to CTIC.
table_change[3]	$79	Load first opcode of non-active table.
end_of_frame_halt[3]	$7A	Last event of frame and PT halt.
end_of_frame_repeat[3]	$7B	Last event of frame and load first opcode of current table.
end_of_frame_switch[3]	$7C	Last event of frame and load first opcode of non-active table.
end_of_macro[4]	$7D	Last macro event.
delay[4]	$7E	Activate delay.
nop	$7F	No operation

1. Macros can only be called from the frame tables.
2. The negate/assert_Tout$_n$ events are the only events that affect external pins.
3. Can be activated only from frame table.
4. Can be activated only from macro table.

10.4 PT Registers

Table 10-5 is a summary of the 19 user-programmable PT control and GPIO registers, including the acronym, bit names, and address (least-significant halfword) of each register. The most-significant halfword of all register addresses is $0020.

Table 10-5. Protocol Timer Register Summary

PTCR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$3800							RSCE	CFCE			HLTR	SPBP	TDZD	MTER	TIME	TE
PTIER	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$3802				TERIE	THIE	DVIE	DSIE			MCIE[2:0]				RSNIE	CFNIE	CFIE
PTSR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$3804		PCE	MBUE	EOFE	THS	DVI	DSPI			MCUI[2:0]				RSNI	CFNI	CFI
PTEVR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$3806													THIP	TXMA	RXMA	ACT

TIMR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$3808								TIPV[8:0]								
CTIC	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$380A			CTIV[13:0]													
CTIMR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$380C			CTIPV[13:0]													
CFC	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$380E								CFCV[8:0]								
CFMR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$3810								CFPV[8:0]								
RSC	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$3812									RSCV[7:0]							
RSMR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$3814									RSPV[7:0]							
PTPCR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$3816									PTPC[7:0]							
PTDDR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$3818									PTDD[7:0]							
PTPDR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$381A									PTPD[7:0]							
FTPTR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$381C									FTPTR[7:0]							
MTPTR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$381E		TxPTR[6:0]								RxPTR[6:0]						
FTBAR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$3820		FTBA1[6:0]								FTBA0[6:0]						
MTBAR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$3822		TxBAR[6:0]								RxBAR[6:0]						
DTPTR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$3824		TDBA[3:0]				TDPTR[2:0]				RDBA[3:0]				RDPTR[2:0]		

10.4.1 PT Control Registers

PTCR Protocol Timer Control Register **$0020_3800**

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
							RSCE	CFCE			HLTR	SPBP	TDZD	MTER	TIME	TE
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 10-6. PTCR Description

Name	Description	Settings
RSCE Bit 9	**Reference Slot Counter Enable**	0 = Disabled (default). 1 = Enabled.
CFCE Bit 8	**Channel Frame Counter Enable**	0 = Disabled (default). 1 = Enabled.
HLTR Bit 5	**Halt Request—**Setting this bit halts PT operation at the next end_of_frame event. Macros may or may not complete depending on the state of the MTER bit.	0 = No halt request (default). 1 = Halt request.
SPBP Bit 4	**Slot Prescaler Bypass—**This bit determines if RSC is driven by the prescaler output (RSE) or the TICK signal	0 = Not bypassed—RSC input = TICK/2400(defalut). 1 = Bypassed—RSC input = TICK.
TDZD Bit 3	**Timer DOZE Disable**	0 = PT ignores DOZE mode (default). 1 = PT stops in DOZE mode.
MTER Bit 2	**Macro Termination—**This bit determines if macros are allowed to complete (i.e., continue to run until the end_of_macro command) when a halt event or halt request is issued.	0 = Macros run to completion (default). 1 = Macros halted immediately.
TIME Bit 1	**Timer Initiate Enable—**This bit determines if event execution begins immediately or waits for the next frame signal (CFE) after the PT is enabled (TE set) or the PT exits the halt state.	0 = Execution delayed until next CFE (default). 1 = Execution begins immediately after TE is set or halt state terminates, as soon as CTIC equal the first ATIC value in the event table.
TE Bit 0	**Timer Enable—**This bit is a “hard” enable/disable of PT activity. Clearing TE stops all PT activity immediately, regardless of the state of MTER.	0 = PT disabled (default). 1 = PT enabled.

PTIER Protocol Timer Interrupt Enable Register **$0020_3802**

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
				TERIE	THIE	DVIE	DSIE			MCIE2	MCIE1	MCIE0		RSNIE	CFNIE	CFIE
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Note: The conditions in Table 10-7 must be met in addition to setting the individual interrupt enable bits in the PTIER.

Table 10-7. Additional Conditions for Generating PT Interrupts

PTIER Bit	Additional Conditions
MCIE2	Set EPT2 bit in the NIER or EFPT2 bit in the FIER.
MCIE1	Set EPT1 bit in the NIER or EFPT1 bit in the FIER.
MCIE0	Set EPT0 bit in the NIER or EFPT0 bit in the FIER.
TERIE THIE DVIE RSNIE CFNIE CFIE	Set EPTM bit in the NIER or EFPTM bit in the FIER.
DSIE	Write the IDPL field in the IPRC with a non-zero value.

The NIER and FIER registers are described on page 7-7.
The IPRC register is described on page 7-14.

Table 10-8. PTIER Description

Name	Description	Settings
TERIE Bit 12	**Timer Error Interrupt Enable—**Enables an MCU interrupt when a timer error has been detected (see Section 10.2.4 on page 10-11).	0 = Interrupt disabled (default). 1 = Interrupt enabled.
THIE Bit 11	**Timer HALT Interrupt Enable—**Enables an MCU interrupt when the PT enters the halt state either from a frame table command or setting the HLTR bit in PTCR.	
DVIE Bit 10	**DSP Vector Interrupt Enable—**Enables an MCU interrupt when a CVR command is executed.	
DSIE Bit 9	**DSP Interrupt Enable—**Enables a DSP $\overline{\text{IRQD}}$ interrupt to the DSP through the MDI when a dsp_int command is executed.	
MCIE2 Bit 6	**MCU Interrupt 2 Enable—**Enables an MCU interrupt when an mcu_int2 command is executed.	
MCIE1 Bit 5	**MCU Interrupt 1 Enable—**Enables an MCU interrupt when an mcu_int1 command is executed.	
MCIE0 Bit 4	**MCU Interrupt 0 Enable—**Enables an MCU interrupt when an mcu_int0 command is executed.	
RSNIE Bit 2	**Reference Slot Number Interrupt Enable—**enables an MCU interrupt when the RSC decrements to zero.	
CFNIE Bit 1	**Channel Frame Number Interrupt Enable—**Enables an MCU interrupt when the CFC decrements to zero.	
CFIE Bit 0	**Channel Frame Interrupt Enable—**Enables an MCU interrupt when the CTIC decrements to zero.	

PTSR Protocol Timer Status Register $0020_3804

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
		PCE	MBUE	EOFE	THS	DVI	DSPI			MCU2	MCU1	MCU0		RSNI	CFNI	CFI
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Each of these bits is cleared by writing it with 1.Writing zero to a bit has no effect.

Table 10-9. PTSR Description

Name	Description	Settings
PCE Bit 14	**Pin Contention Error—**Set when two events attempt to drive opposite values to a PT pin simultaneously.	0 = PCE has not occurred (default). 1 = PCE has occurred.
MBUE Bit 13	**Macro Being Used Error—**Set when a frame table command calls a macro that is already active.	0 = MBUE has not occurred (default). 1 = MBUE has occurred
EOFE Bit 12	**End of Frame Error—**Set when CFE occurs before an end_of_frame command.	0 = EOFE has not occurred (default). 1 = EOFE has occurred.
THS Bit 11	**Timer Halt State—**Indicates if the PT is in halt state. Operation resumes from the beginning of frame table 0 when THS is cleared.	0 = Normal mode (default). 1 = Halt mode.
DVI Bit 10	**DSP Vector Interrupt—**Set by a CVR event.	0 = DVI has not occurred (default). 1 = DVI has occurred.
DSPI Bit 9	**DSP Interrupt—**Set by a dsp_int event.	0 = DSPI has not occurred (default). 1 = DSPI has occurred.
MCUI2 Bit 6	**MCU2 Interrupt—**Set by an mcu_int2 event.	0 = MCUI2 has not occurred (default). 1 = MCUI2 has occurred.
MCUI1 Bit 5	**MCU1 Interrupt—**Set by an mcu_int1 event.	0 = MCUI1 has not occurred (default). 1 = MCUI1 has occurred.
MCUI0 Bit 4	**MCU0 Interrupt—**Set by an mcu_int0 event.	0 = MCUI0 has not occurred (default). 1 = MCUI0 has occurred.
RSNI Bit 2	**Reference Slot Number Interrupt—**Set when the RSC decrements to zero.	0 = RSNI has not occurred (default). 1 = RSNI has occurred.
CFNI Bit 1	**Channel Frame Number Interrupt—**Set when the CFC decrements to zero.	0 = CFNI has not occurred (default). 1 = CFNI has occurred.
CFI Bit 0	**Channel Frame Interrupt—**Set when the CTIC decrements to zero.	0 = CFI has not occurred (default). 1 = CFI has occurred.

PTEVR Protocol Timer Event Register $0020_3806

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
													THIP	TXMA	RXMA	ACT
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

PTEVR is a read-only register.

Table 10-10. PTEVR Description

Name	Description	Settings
THIP Bit 3	**Timer Halt in Process—**Indicates if the PT is in the process of halting.	0 = Normal mode (default). 1 = Halt mode in progress.
TxMA Bit 2	**Transmit Macro Active**	0 = Not active(default). 1 = Active.
RxMA Bit 1	**Receive Macro Active**	0 = Not active(default). 1 = Active.
ACT Bit 0	**Active Frame Table**	0 = Frame table 0 active (default). 1 = Frame table 1 active.

TIMR Time Interval Modulus Register $0020_3808

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
								TIMV[8:0]								
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 10-11. TIMR Description

Name	Description
TIMV Bits 8–0	**Time Interval Modulus Value—**This field contains the value loaded into CTIG when it rolls over. When TIMV = n, the PT reference clock TICK frequency is MCU_CLK/(n+1). This register should be written before the PT is enabled. **Note:** In normal operation, TIMR must be greater than 5 to ensure reliable PT event generation. However, TIMR values of 2 to 5 are sufficient for tracking channel activity when the PT does not execute events, such as in low power modes. TIMR values of 0 and 1 are not supported.

CTIC Channel Time Interval Counter $0020_380A

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
	▨	▨	CTIV[13:0]													
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 10-12. CTIC Description

Name	Description
CTIV[13:0] Bits 13–0	**Channel Time Interval Value—**This field contains the current CTIC value. CTIC is described on page 10-3.

CTIMR Channel Time Interval Modulus Register $0020_380C

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
	▨	▨	CTIMV[13:0]													
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 10-13. CTIMR Description

Name	Description
CTIMV Bits 13–0	**Time Interval Modulus Value—**This field contains the value loaded into CTIC when it rolls over or when a reload_counter command. The actual CTIC modulus is equal to CTIMV + 1. For example, to obtain a CTIC modulus value of 2400, this field should be written with 2399 (=$95F).

CFC Channel Frame Counter $0020_380E

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
	▨	▨	▨	▨	▨	▨	▨	CFCV[8:0]								
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 10-14. CFC Description

Name	Description
CFCV[8:0] Bits 8–0	**Channel Time Interval Value—**This field contains the current CFC value. CFC is described on page 10-3. **Note:** Writing CFC with zero when it is enabled sets the CFNI bit in PTSR and generates an interrupt if the CFNIE bit in PTIER is set.

CFMR Channel Frame Modulus Register $0020_3810

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
								CFMV[8:0]								
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 10-15. CFMR Description

Name	Description
CFMV Bits 8–0	**Channel Frame Modulus Value—**This field contains the value loaded into CFC when it is enabled and when it rolls over. A CFMV value of 0 is not supported. This register should be written before the CFC is enabled.

RSC Reference Slot Counter $0020_3812

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
									RSCV[7:0]							
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 10-16. RSC Description

Name	Description
RSCV[7:0] Bits 7–0	**Reference Slot Count Value—**This field contains the current RSC value. RSC is described on page 10-4. **Note:** Writing RSC with zero when it is enabled sets the RSNI bit in PTSR and generates an interrupt if the RSNIE bit in PTIER is set.

RSMR Reference Slot Modulus Register $0020_3814

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
									RSMV[7:0]							
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 10-17. RSMR Description

Name	Description
RSMV Bits 7–0	**Reference Slot Modulus Value—**This field contains the value loaded into RSC when it is enabled and when it rolls over. An RSMV value of 0 is not supported. This register should be written before the RSC is enabled.

FTPTR Frame Table Pointer **$0020_381C**

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
										FTPTR[6:0]						
RESET	0	0	0	0	0	0	0	0	0	—	—	—	—	—	—	—

Table 10-18. FTPTR Description

Name	Description
FTPTR[6:0] Bits 6–0	**Frame Table Pointer[6:0]—**These read-only bits contain a pointer to the next frame table entry.

MTPTR Macro Table Pointer **$0020_381E**

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
		TxPTR[6:0]								RxPTR[6:0]						
RESET	0	—	—	—	—	—	—	—	0	—	—	—	—	—	—	—

Table 10-19. MTPTR Description

Name	Description
TxPTR[6:0] Bits 14–8	**Transmit Macro Pointer[6:0]—**These read-only bits contain a pointer to the next transmit macro table entry.
RxPTR[6:0] Bits 6–0	**Receive Macro Pointer[6:0]—**These read-only bits contain a pointer to the next receive macro table entry.

FTBAR Frame Table Base Address Register **$0020_3820**

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
		FTBA1[6:0]								FTBA0[6:0]						
RESET	0	—	—	—	—	—	—	—	0	—	—	—	—	—	—	—

Table 10-20. FTBAR Description

Name	Description
FTBA1[6:0] Bits 14–8	**Frame Table 1 Base Address[6:0]—**These bits specify the offset from the beginning of PT RAM ($0020_3000) of the first entry in Frame Table 1. They should be initialized before the PT is enabled.
FTBA0[6:0] Bits 6–0	**Frame Table 0 Base Address[6:0]—**These bits specify the offset from the beginning of PT RAM of the first entry in Frame Table 0. They should be initialized before the PT is enabled.

MTBAR Macro Table Base Address Register $0020_3822

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
		TxBA1[6:0]								RxBA0[6:0]						
RESET	0	—	—	—	—	—	—	—	0	—	—	—	—	—	—	—

Table 10-21. MTBAR Description

Name	Description
TxBA1[6:0] Bits 14–8	**Transmit Macro Base Address[6:0]—**These bits specify the offset from the beginning of PT RAM of the first entry in the transmit macro. They should be initialized before the first transmit macro is activated.
RxBA0[6:0] Bits 6–0	**Receive Macro Base Address[6:0]—**These bits specify the offset from the beginning of PT RAM of the first entry in the receive macro. They should be initialized before the first receive macro is activated.

DTPTR Delay Table Pointer $0020_3824

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
		TDBA[3:0]				TDPTR[2:0]				RDBA[3:0]				RDPTR[2:0]		
RESET	0	—	—	—	—	—	—	—	0	—	—	—	—	—	—	—

Table 10-22. DTPTR Description

Name	Description
TDBA[3:0] Bits 14–11	**Transmit Macro Delay Table Base Address[3:0]—**These bits determine the location in memory of the first entry in the transmit macro delay table. They contain the four most significant bits of the 7-bit offset from the beginning of PT RAM. TDBA should be initialized before the first transmit macro is activated.
TDPTR[2:0] Bits 10–8	**Transmit Macro Delay Pointer[2:0]—**These read-only bits are the three-bit offset from TDBA that point to the delay table entry of the active transmit macro. They are specified by the particular event code that called the macro.
RDBA[3:0] Bits 6–3	**Receive Macro Delay Table Base Address[3:0]—**These bits determine the location in memory of the first entry in the receive macro delay table. They contain the four most significant bits of the 7-bit offset from the beginning of PT RAM. RDBA should be initialized before the first receive macro is activated.
RDPTR[2:0] Bits 2–0	**Receive Macro Delay Pointer[2:0]—**These read-only bits are the three-bit offset from TDBA that point to the delay table entry of the active receive macro. They are specified by the particular event code that called the macro.

10.4.2 GPIO Registers

PTPCR PT Port Control Register **$0020_3816**

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
									PTPC[7:0]							
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 10-23. PTPCR Description

Name	Description	Settings
PTPC[7:0] Bits 7–0	**PT Port Control**—Each of these bits determines if the corresponding TOUT pin functions as a PT TOUT pin or GPIO.	0 = GPIO (default). 1 = Protocol timer pin (TOUT)

PTDDR PT Data Direction Register **$0020_3818**

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
									PTDD[7:0]							
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 10-24. PTDDR Description

Name	Description	Settings
PTDD[7:0] Bits 7–0	**PT Data Direction**—For each PT pin that is configured as GPIO, the corresponding PTDD pin determines if it is an input or output.	0 = Input (default). 1 = Output

PTPDR PT Port Data Register **$0020_381A**

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
									PTPD[7:0]							
RESET	0	0	0	0	0	0	0	0	—	—	—	—	—	—	—	—

Table 10-25. PTPDR Description

Name	Description
PTPD[7:0] Bits 7–0	**PT Port Data**—The function of each of these bits depends on how the corresponding TOUT pin is configured. • PT Reading PTPDn reflects the internal latch. Writing PTPDn writes the data latch. If the PT is disabled (TE = 0), the PTPDn is the initial state of the TOUT driver. • GPI Reading PTPDn reflects the pin value. Writing PTPDn writes the data latch. • GPO Reading PTPDn reflects the data latch, which equals the pin value. Writing PTPDn writes the data latch which drives the pin value.

10.5 Protocol Timer Programming Example

The following lines illustrate a typical series of entries in the event table.

Frame Table No. 0

<abs TIC>	trigger QSPI_0	
<abs TIC>	Rx_macro0Start	Rx burst timing macro
<abs TIC>	Tx_macro1start	Tx burst timing macro
<abs TIC>	trigger CVR5	
<abs TIC>	Rx_macro2start	Rx burst timing macro
<abs TIC>	Table_change	

Frame Table No. 1

<abs TIC>	Rx_macro2start	Rx burst timing macro
<abs TIC>	DSP_int	DSP interrupt
<abs TIC>	MCU_int_0	MCU interrupt
<abs TIC>	trigger QSPI_2	
<abs TIC>	Tx_macro3start	Tx burst timing macro
<abs TIC>	End_of_frame_repeat	

Receive Macro Table

<rel TIC>	Assert_Tout3	
<rel TIC>	delay	activate delay
<rel TIC>	trigger QSPI_1	
<rel TIC>	Negate_Tout3	
<rel TIC>	End_of_macro	

Transmit Macro Table

<rel TIC>	Assert_Tout6	
<rel TIC>	Assert_Tout7	
<rel TIC>	trigger CVR2	
<rel TIC>	delay	activate delay
<rel TIC>	MCU_int_0	
<rel TIC>	Negate_Tout6	
<rel TIC>	End_of_macro	

Chapter 11
UART

The Universal Asynchronous Receiver/Transmitter (UART) module provides communication with external devices such as modems and other serial devices. Key features of the UART include the following:

- Full duplex operation.
- Full 8-wire serial interface.[1]
- Direct support of the Infrared Data Association (IrDA) mechanism.
- Robust receiver data sampling with noise filtering.
- 16-word FIFOs for transmit and receive, block-addressable with the LDM and STM instructions.
- 7- or 8-bit characters with optional even or odd parity and one or two stop bits.
- BREAK signal generation and detection.
- 16x bit clock generator providing bit rates from 300 bps to 525 Kbps.
- Four maskable interrupts.
- $\overline{\text{RTS}}$ interrupt providing wake from STOP mode.
- Low power modes.
- Internal or external 16x clock.
- Far-end baud rate can be automatically determined (autobaud).[2]

The UART performs all normal operations associated with “start-stop” asynchronous communication. Serial data is transmitted and received at standard bit rates in either NRZ or IrDA format.

11.1 UART Definitions

The following definitions apply to both transmitter and receiver operation:

---

[1] Using GPIO pins for $\overline{\text{DSR}}$, $\overline{\text{DCD}}$, $\overline{\text{DTR}}$, and $\overline{\text{RI}}$.

[2] Using the GP timer.

Bit Time—The time allotted to transmit or receive one bit of data.

Start Bit—One bit-time of logic zero that indicates the beginning of a data frame. A start bit must begin with a one-to-zero transition.

Stop Bit—One bit-time of logic one that indicates the end of a data frame.

Frame—A series of bits consisting of the following sequence:

1. A start bit
2. 7 or 8 data bits
3. optional parity bit
4. one or two stop bits

BREAK—A frame in which all bits, including the stop bit, are logic zero. This frame is normally sent to signal the end of a message or the beginning of a new message.

Framing Error—An error condition in which the expected stop bit is a logic zero. This can be caused by a misaligned frame, noise, a BREAK frame, or differing numbers of data and/or stop bits between the two devices. Note that if a UART is configured for two stop bits and only one stop bit is received, this condition is not considered a frame error.

Parity Error—An error condition in which the calculated parity of the received data bits in a frame differs from the frame's parity bit. Parity error is only calculated after an entire frame is received.

Overrun Error—An error condition in which the receive FIFO is full when another character is received. The received character is ignored to prevent overwriting the existing data. An overrun error indicates that the software reading the FIFO is not keeping up with character reception on the RxD line.

11.2 UART Architecture

This section provides a brief description of the UART transmitter, receiver, clock generator, infrared interface, pins, and frame configuration. A block diagram of the UART is presented in Figure 11-1.

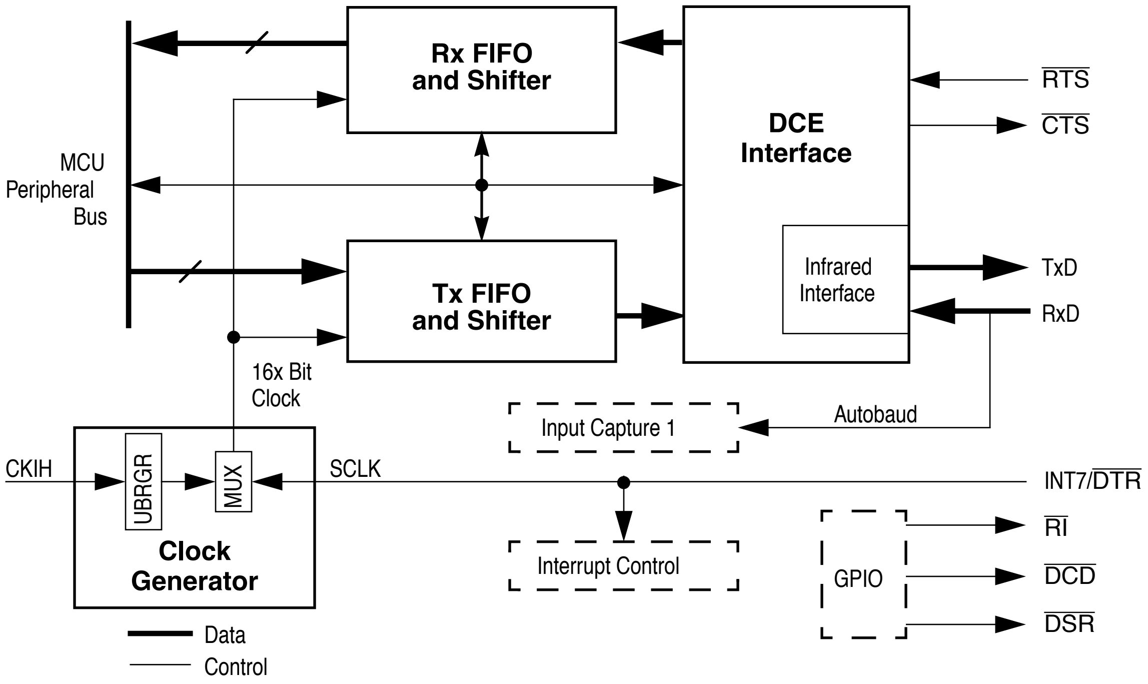


Figure 11-1. UART Block Diagram

11.2.1 Transmitter

The UART transmitter contains a 16-word FIFO (UTX) with one character (byte) per word. Word-aligned characters enable the MCU to perform block writes using the Store Multiple (STM) command. The transmitter adds start, stop and optional parity bits to each character to generate a transmit frame. It then shifts the frame out serially on the UART transmission pin, TxD. One bit is shifted on each cycle of a “1x” transmit clock. It derives the 1x clock from the “16x” clock produced by the clock generator. Transmission can begin as soon as UTX is written, or can be delayed until the far-end receiver asserts the $\overline{\text{RTS}}$ signal. Interrupts can be generated when $\overline{\text{RTS}}$ changes state and when UTX is empty or the number of untransmitted words falls below a programmed threshold. The transmitter is enabled by setting the TxEN bit in UART Control Register 1 (UCR1).

11.2.2 Receiver

The UART receiver contains a 16-word FIFO (URX), one character per word, enabling the MCU to perform block reads using the Load Multiple (LDM) command. It receives bits serially from the UART receive pin, RxD, strips the start, stop, and parity (if present) bits and stores the characters in URX. To provide jitter and noise tolerance, the receiver samples each bit 16 times and applies a voting technique to determine the bit’s value. The receiver monitors data for proper frame construction, BREAK characters (all zeros), parity errors, and receiver overrun. Each of the 16 URX words contains a received character data field, error flags and a ‘character ready’ flag to indicate when the character is ready to be read. Errors flags include those for frame, parity, BREAK, and receiver overrun, as well as a general error flag. In the event of a receiver overrun, the receiver can deassert the $\overline{\text{CTS}}$ signal to turn off the far-end transmitter. The receiver is enabled by setting the RxEN bit in UCR1.

11.2.3 Clock Generator

The clock generator provides a 16x clock signal for the transmitter and receiver. The input can be either CKIH or an external clock, SCLK, applied to the INT7/$\overline{\text{DTR}}$ pin, depending on the state of the CLKSRC bit in UART Control Register 2 (UCR2). CKIH is divided by a 12-bit value written in the UART Bit Rate Generator Register (UBRGR).

11.2.4 Infrared Interface

The Infrared Interface converts data to be transmitted or received as specified in the IrDA Serial Infrared Physical Layer Specification. Each “zero” driven on the TxD pin is a narrow logic high pulse, 3/16 of a bit time in duration; each “one” is a full logic low. The receiver in kind interprets a narrow pulse on RxD as a “zero” and no pulse as a “one”. External circuitry is required to drive an infrared LED with TxD and to convert received infrared signals to electrical signals for RxD. The Infrared Interface is enabled by setting the IREN bit in UCR1.

11.2.5 UART Pins

The DSP56652 provides pins for $\overline{\text{RTS}}$, $\overline{\text{CTS}}$, TxD, and RxD. The remaining UART signals can be implemented with GPIO pins. Suggested GPIO pin allocations are listed in Table 11-1, but any GPIO pins can be used.

Table 11-1. Suggested GPIO Pins for UART Signals

UART Signal	Suggested Pin	Peripheral
$\overline{\text{DCD}}$ (Data Carrier Detect)	ROW6	Keypad Port—see Section 13.2 on page 13-4
$\overline{\text{RI}}$ (Ring Indicator)	ROW7	
$\overline{\text{DSR}}$ (Data Set Ready)	INT6	Edge Port—see Section 7.3 on page 7-15
$\overline{\text{DTR}}$ (Data Terminal Ready)	INT7	

In addition, any unused UART pins can be configured for GPIO.

11.2.6 Frame Configuration

The DSP56652 UART configuration must match that of the external device. The most common frame format consists of one start bit, eight data bits (LSB first), no parity bit, and one stop bit, for a total of 10 bit times per frame. All elements of the frame—the number of data and stop bits, parity enabling and odd/even parity—are determined by bits in UCR2.

11.3 UART Operation

This section describes UART transmission and reception, clock generation, and operation in low power and Debug modes.

The UART is enabled by setting the UEN bit in UCR1.

11.3.1 Transmission

The MCU writes data for UART transmission to UTX. Normally, the UART waits for $\overline{\text{RTS}}$ to be asserted before beginning transmission. The $\overline{\text{RTS}}$ pin can be monitored by reading the RTSS bit in the UART Status Register (USR). When $\overline{\text{RTS}}$ changes state, the RTSD bit in USR is set. If the RTSDIE bit in UCR1 has been set, an interrupt is generated as well. This interrupt can wake the MCU from STOP mode. If $\overline{\text{RTS}}$ is deasserted in mid-character, the UART completes transmission of the character before shutting off the transmitter. Transmitter operation can also proceed without $\overline{\text{RTS}}$ by setting the IRTS bit in UCR2. In this case, $\overline{\text{RTS}}$ has no effect on the transmitter or RTSD and cannot generate an interrupt.

A BREAK character can be sent by setting the SNDBRK bit in UCR1. When the MCU sets SNDBRK, the transmitter completes any frame in progress and transmits zeros, sampling SNDBRK after every bit is sent. UTX can be written with more transmit data while SNDBRK is set. When it samples SNDBRK cleared, the transmitter sends two marks before transmitting data (if any) in UTX. Care must be taken to ensure that SNDBRK is set for a sufficient length of time to generate a valid BREAK.

When all data in UTX has been sent and the FIFO and shifter are empty, the TXE bit in USR is set. If the amount of untransmitted data falls below a programmed threshold, the TRDY bit in USR is set. The threshold can be set for one, four, eight, or fourteen characters by writing TxFL[1:0] in UCR1. Both TXE and TRDY can trigger an interrupt if the TXEIE and TRDYIE bits respectively in UCR1 are set. The two interrupts are internally wire-or'd to the interrupt controller.

11.3.2 Reception

The RxD line is at a logic one when it is idle. If the pin goes to a logic low, and the receiver detects a qualified start bit, it proceeds to decode the succeeding transitions on the RxD pin, monitoring for the correct number of data and stop bits and checking for parity according to the configuration in UCR2. When a complete character is decoded, the data is written to the data field in a URX register and the CHARRDY bit in that register is set. If a valid stop bit is not detected a frame error is flagged by setting the URX FRMERR bit. A parity error is flagged by setting the PRERR bit. If a BREAK frame is detected the BRK and FRMERR flags are set. If the URX is about to overflow (i.e., the FIFO is full as another character is being received), the OVRRUN flag is set. If any of these four flags is set, the ERR bit is also set. If the number of unread words exceeds a threshold programmed by the RxFL[1:0] bits in UCR1, the RRDY bit in USR is set, and an interrupt is generated if the RRDYIE bit in UCR1 has been set. Adjusting the threshold to a value of one can effectively generate an interrupt every time a character is ready. Reading the URX clears the interrupt and all the flags.

The $\overline{\text{CTS}}$ pin can be asserted to enable the far-end transmitter, and deasserted to prevent receiver overflow. $\overline{\text{CTS}}$ is driven by receiver hardware if the CTSC bit in UCR2 is set. The pin is driven by software via the CTSD bit in UCR2 if CTSC is cleared.

11.3.3 UART Clocks

The clock generator provides a 16x bit clock for the transmitter and receiver. Software can select either an external or internally-generated clock through the CLKSRC bit in UCR2. Clearing CLKSRC selects the internal clock which is derived by dividing CKIH by a number between 1 and 4096, determined by UBRGR. This provides sufficient flexibility to generate standard baud rates from a variety of clock sources. Clock error calculation is straightforward, as shown in Example 11-1.

Example 11-1. UART Baud Error Calculation

Desired baud rate = 115.2 kbps
Input clock = 16.8 MHz
Divide ratio = 9 (UBRGR[11:0] = 8)
Actual baud rate = 16.8 MHz / 9 / 16 = 116.67 kHz
Actual/required ratio = 116.67 / 115.2 = 1.0127
Error per bit = 1.27%
Error per 12-bit frame = 15%

Setting the CLKSRC bit selects an external clock, SCLK, which is input on the INT7/$\overline{\text{DTR}}$ pin.

11.3.4 Baud Rate Detection (Autobaud)

The baud rate from the far-end transmitter can be determined in software by observing the duration of the logic one and logic zero states of the Input Capture 1 (IC1) module, which is internally connected to RxD for this purpose.

11.3.5 Low-Power Modes

The UART serial interface operates as long as the 16x bit clock generator is provided with a clock and the UART is enabled (the UARTEN bit in UCR1 is set). The internal bus interface is operational if the system clock is running. The RXEN, TXEN, and UARTEN bits enable low-power control through software. UART functions in the various hardware-controlled low power modes is shown in Table 11-2.

Table 11-2. UART Low Power Mode Operation

	Normal Mode	WAIT Mode	DOZE Mode		STOP Mode
			DOZE = 0	DOZE = 1	
System Clock	ON	ON	ON	ON	OFF
UART Serial I/F	ON	ON	ON	OFF	OFF
Internal Bus	ON	ON	ON	OFF	OFF

If DOZE mode is entered with the DOZE bit asserted while the UART serial interface is receiving or transmitting data, the UART completes the receive or transmit of the current character, then signals to the far-end transmitter or receiver to stop sending or receiving. Control, status, and data registers do not change when entering or exiting low-power modes.

11.3.6 Debug Mode

In Debug mode, URX reads do not advance the internal RX FIFO pointer, so repeated URX reads do not cause the URX to change once it contains a valid character.

11.4 UART Registers

Table 11-3 is a summary of the UART control and GPIO registers, including the acronym, bit names, and address (least-significant halfword) of each register. The most-significant halfword of all register addresses is $0020.

Table 11-3. UART Register Summary

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
URX $4000–403C	CHARRDY	ERR	OVRRUN	FRMERR	BRK	PRERR			Rx DATA							
UTX $4040–407C									Tx DATA							
UCR1 $4080	TxFL[1:0]		TRDIE	TXEN	RxFL[1:0]		RRDYIE	RxEN	IREN	TXEIE	RTSDIE	SNDBRK			DOZE	UEN
UCR2 $4082		IRTS	CTSC	CTSD				PREN	PROE	STPB	WS	CLKSRC				
UBRGR $4084					CD[11:0]											
USR $4086	TXE	RTSS	TRDY				RRDY				RTSD					
UTS $4088			FRCPERR	LOOP		LOOPIR										
UPCR $408A													PC[3:0]			
UDDR $408C													PDC[3:0]			
UPDR $408E													PD[3:0]			

11.4.1 UART Control Registers

URX UART Receive Register **$0020_4000**

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
	CHARRDY	ERR	OVRRUN	FRMERR	BRK	PRERR	▨	▨	Rx DATA							
RESET	0	0	0	0	0	0	0	0	—	—	—	—	—	—	—	—

The 16-entry receive buffer FIFO is accessed through the URX register at address $0020_4000. This register is actually mapped to 16 word addresses from $0020_4000 to $0020_403C to support LDM instructions. At reset, the flag bits in the most significant byte are cleared, and the least significant byte, which holds the received character, contains random data.

Table 11-4. URX Description

Name	Description	Settings
CHARRDY Bit 15	**Character Ready—**Set when the complete character has been received and error conditions have been evaluated. Cleared when the register is read.	0 = Character not ready (default). 1 = Character ready.
ERR Bit 14	**Error Detected—**Set when any of the error conditions indicated in bits 13–10 is present. Cleared when the register is read.	0 = No error detected (default). 1 = Error detected.
OVRRUN Bit 13	**Receiver Overrun—**Set when incoming data is ignored because the URX FIFO is full. An overrun error indicates that MCU software is not keeping up with the receiver. Under normal conditions, this bit should never be set. Cleared when the register is read.	0 = No overrun (default). 1 = Overrun error.
FRMERR Bit 12	**Frame Error—**Set when a received character is missing a stop bit, indicating that the data may be corrupted. Cleared when the register is read.	0 = No framing error detected (default). 1 = Framing error detected for this character.
BRK Bit 11	**BREAK Detect—**Set when all bits in the frame, including stop bits, are zero, indicating that the current character is a BREAK. FRMERR is also set. If odd parity is employed, PRERR is also set. BRK is cleared when the register is read.	0 = BREAK not detected (default). 1 = BREAK detected for this character.
PRERR Bit 10	**Parity Error—**Set when parity is enabled and the calculated parity in the received character does not match the received parity bit, indicating that the data may be corrupted. PRERR is never set when parity is disabled. Cleared when the register is read.	0 = No parity error (default). 1 = Parity error detected for this character.
Rx DATA Bits 7–0	**Received Data—**This field contains the character in a received frame. In 7-bit mode, bit 7 is always zero.	

UTX UART Transmit Register **$0020_4040**

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
	▨	▨	▨	▨	▨	▨	▨	▨	Tx DATA							
RESET	0	0	0	0	0	0	0	0	—	—	—	—	—	—	—	—

The 16-entry transmit buffer FIFO is accessed through the UTX register at address $0020_4040. This register is actually mapped to 16 word addresses from $0020_4040 to $0020_407C to support STM instructions. Reading one of these registers returns zeros in bits 15–8 and random data in bits 7–0.

Table 11-5. UTX Description

Name	Description	Settings
Tx DATA Bits 7–0	**Transmit Data**—This field contains data to be transmitted to the far-end device. Writing to this register initiates transmission of a new character. Data is transmitted LSB first. In 7-bit mode, bit 7 is ignored. Tx DATA should only be written when the TRDY bit in USR is set, indicating that UTX can accept more data.	

UCR1 UART Control Register 1 $0020_4080

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
	TxFL[1:0]		TRDYIE	TXEN	RxFL[1:0]		RRDYIE	RxEN	IREN	TXEIE	RTSDIE	SNDBRK			DOZE	UEN
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 11-6. UCR1 Description

Name	Description	Settings
TXFL[1:0] Bits 15–14	**Transmit FIFO Interrupt Trigger Level—**These bits determine the number of available registers in UTX required to indicate to the MCU that space is available to write data to be transmitted. When the number of available registers rises above this threshold, the TRDY bit in USR is set and a maskable interrupt can be generated	00 = One FIFO slot (default). 01 = Four FIFO slots. 10 = Eight FIFO slots. 11 = Fourteen FIFO slots.
TRDYIE Bit 13	**Transmitter Ready Interrupt Enable—**Setting this bit enables an interrupt when the space available in UTX reaches the threshold determined by TxFL[1:0]. **Note:** Either the EUTX bit in the NIER or the EFUTX bit in the FIER must also be set in order to generate this interrupt (see page 7-7).	0 = Interrupt disabled (default). 1 = Interrupt enabled.
TXEN Bit 12	**Transmitter Enable—**Setting this bit enables the UART transmitter. If TXEN is cleared during a transmission, the transmitter is immediately disabled and the TxD pin is pulled high. The UTX cannot be written while TXEN is cleared.	0 = Transmitter disabled (default). 1 = Transmitter enabled.
RXFL[1:0] Bits 11–10	**Receive FIFO Interrupt Trigger Level—**These bits determine the number of received characters in URX required to indicate to the MCU that the URX should be read. When the number of registers containing received data rises above this threshold, the RRDY bit in USR is set and a maskable interrupt can be generated.	00 = One FIFO slot (default). 01 = Four FIFO slots. 10 = Eight FIFO slots. 11 = Fourteen FIFO slots.
RRDYIE Bit 9	**Receiver Ready Interrupt Enable—**Setting this bit enables an interrupt when the number of received characters in UTX reaches the threshold determined by RxFL[1:0]. **Note:** Either the EURX bit in the NIER or the EFURX bit in the FIER must also be set in order to generate this interrupt (see page 7-7).	0 = Interrupt disabled (default). 1 = Interrupt enabled.

Table 11-6. UCR1 Description (Continued)

Name	Description	Settings
RXEN Bit 8	**Receiver Enable—**Setting this bit enables the UART transmitter. **Note:** The receiver requires a valid one-to-zero transition to accept a valid character, and will not recognize BREAK characters if the RxD line is at a logic low when the receiver is enabled.	0 = Receiver disabled (default). 1 = Receiver enabled.
IREN Bit 7	**Infrared Interface Enable—**Setting this bit enables the IrDA infrared interface, configuring the RxD and TxD pins to operate as described in Section 11.2.4 on page 11-4.	0 = Normal NRZ (default). 1 = IrDA.
TXEIE Bit 6	**Transmitter Empty Interrupt Enable—**Setting this bit enables an interrupt when all data in UTX has been transmitted. **Note:** Either the EUTX bit in the NIER or the EFUTX bit in the FIER must also be set in order to generate this interrupt (see page 7-7).	0 = Interrupt disabled (default). 1 = Interrupt enabled.
RTSDIE Bit 5	**RTS Delta Interrupt Enable—**Setting this bit enables an interrupt when the $\overline{\text{RTS}}$ pin changes state. **Note:** Either the EURTS bit in the NIER or the EFRTS bit in the FIER must also be set in order to generate this interrupt (see page 7-7).	0 = Interrupt disabled (default). 1 = Interrupt enabled.
SNDBRK Bit 4	**Send BREAK—**Setting this forces the transmitter to send BREAK characters, effectively pulling the TxD pin low until SNDBRK is cleared. SNDBRK cannot be set unless TXEN and UEN are both set.	0 = Normal transmission (default). 1 = BREAK characters transmitted.
DOZE Bit 1	**UART DOZE Mode**	0 = UART ignores DOZE mode (default). 1 = UART stops in DOZE mode.
UEN Bit 0	**UART Enable—**This bit must be set to enable the UART. If UEN is cleared during a transmission, the transmitter stops immediately and pulls TxD to logic one.	0 = UART disabled (default). 1 = UART enabled.

UCR2 UART Control Register 2 $0020_4082

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
		IRTS	CTSC	CTSD				PREN	PROE	STPB	CHSZ	CLKSRC				
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 11-7. UCR2 Description

Name	Description	Settings
IRTS Bit 14	**Ignore $\overline{\text{RTS}}$ Pin—**Setting this bit configures the UART to ignore the $\overline{\text{RTS}}$ pin, enabling it to transmit at any time. When IRTS is cleared, the UART must wait for $\overline{\text{RTS}}$ to assert before it can transmit.	0 = $\overline{\text{RTS}}$ qualifies data transmission (default). 1 = $\overline{\text{RTS}}$ ignored.
CTSC Bit 13	**$\overline{\text{CTS}}$ Pin Control—**This bit determines whether hardware or software controls the $\overline{\text{CTS}}$ pin. When CTSC is set, the receiver controls $\overline{\text{CTS}}$, automatically deasserting it when URX is full. When CTSC is cleared, the $\overline{\text{CTS}}$ pin is driven by the CTSD bit.	0 = CTSD bit controls $\overline{\text{CTS}}$ (default). 1 = Receiver control $\overline{\text{CTS}}$.
CTSD Bit 12	**$\overline{\text{CTS}}$ Driver—**This bit drives the $\overline{\text{CTS}}$ pin when CTSC is cleared. Setting this bit asserts $\overline{\text{CTS}}$, meaning that it is driven low; clearing CTSD deasserts (pulls high) $\overline{\text{CTS}}$. When CTSC is set this bit has no effect.	0 = $\overline{\text{CTS}}$ driven high (default). 1 = $\overline{\text{CTS}}$ driven low.
PREN Bit 8	**Parity Enable—**Controls the parity generator in the transmitter and the parity checker in the receiver.	0 = Parity disabled (default). 1 = Parity enabled.
PROE Bit 7	**Parity Odd/Even—**Determines the functionality of the parity generator and checker. This bit has no effect if PREN is cleared.	0 = Even parity (default). 1 = Odd parity.
STPB Bit 6	**Stop Bits—**Determines the number of stop bits transmitted. The STPB bit has no effect on the receiver, which expects one or more stop bits.	0 = One stop bit (default). 1 = Two stop bits.
CHSZ Bit 5	**Character Size—**Determines the number of character bits transmitted and expected.	0 = 8 character bits (default). 1 = 7 character bits.
CLKSRC Bit 4	**Clock Source—**Determines the source of the 16x transmit and receive clock. This bit should not be changed during a transmission.	0 = CKIH divided by UBRGR (default). 1 = IRQ7/$\overline{\text{DTR}}$ pin.

UBRGR UART Bit Rate Generator Register $0020_4084

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
					CD[11:0]											
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 11-8. UBRGR Description

Name	Description	Settings
CD[11:0] Bits 11–0	**Clock Divider—**If the CLKSRC bit in UCR2 is cleared, CKIH is divided by a number determined by this field to generate the 16x bit clock. The actual divisor is equal to the value in CD[11:0] plus one, i.e., a value of $000 yields a divisor of 1, and $FFF yields a divisor of 4096.	

USR UART Status Register $0020_4086

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
	TXE	RTSS	TRDY				RRDY				RTSD					
RESET	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

All bits are read-only, and writes have no effect, with the exception of RTSD, which is cleared by writing it with one.

Table 11-9. USR Description

Name	Description	Settings
TXE Bit 15	**Transmitter Empty—**Set when all data in UTX FIFO has been sent. Cleared by a write to UTX.	0 = UTX or transmit buffer contains unsent data (default). 1 = UTX and transmit buffer empty.
RTSS Bit 14	**$\overline{\text{RTS}}$ Pin Status—**Indicates the current status of the $\overline{\text{RTS}}$ pin. When the USR is read, a "snapshot" of the $\overline{\text{RTS}}$ pin is taken immediately before this bit is presented to the data bus.	
TRDY Bit 13	**Transmitter Ready—**Set when the number of unsent characters in UTX FIFO falls below the threshold determined by the TxFL bits in UCR1. Cleared when the MCU writes enough data to fill the UTX above the threshold.	0 = Number of unsent characters is above the threshold (default). 1 = Number of unsent characters is below the threshold.
RRDY Bit 9	**Receiver Ready—**Set when the number of characters in URX FIFO exceeds the threshold determined by the RxFL bits in UCR1. Cleared when the MCU reads enough data to bring the number of unread characters in URX below the threshold.	0 = Number of unread characters is below the threshold (default). 1 = Number of unread characters is above the threshold.
RTSD Bit 15	**RTS Delta—**Set when the $\overline{\text{RTS}}$ pin changes state. Cleared by writing the bit with one.	0 = $\overline{\text{RTS}}$ has not changed state since RTSD was last cleared (default). 1 = $\overline{\text{RTS}}$ has changed state.

UTS UART Test Register $0020_4088

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
			FRCPERR	LOOP		LOOPIR										
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register is provided for test purposes and is not intended for use in normal operation.

Table 11-10. UTS Description

Name	Description	Settings
FRCPERR Bit 13	**Force Parity Error—**If parity is enabled, the transmitter is forced to generate a parity error as long as this bit is set.	0 = No intentional parity errors generated (default). 1 = Parity errors generated.
LOOP Bit 12	**Loop Tx and Rx—**Setting this bit connects the receiver to the transmitter. The RxD pin is ignored.	0 = Normal operation (default). 1 = Receiver connected to transmitter.
LOOPIR Bit 10	**Loop Tx and Rx for Infrared Interface—** Setting this bit connects the infrared receiver to the infrared transmitter.	0 = Normal IR operation (default). 1 = IR Receiver connected to IR transmitter.

11.4.2 GPIO Registers

Four of the UART pins can function as GPIO, governed by the following control registers.

UPCR UART Port Control Register **$0020_408A**

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
													UPC3 ($\overline{\text{CTS}}$)	UPC2 ($\overline{\text{RTS}}$)	UPC1 (TxD)	UPC0 (RxD)
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 11-11. UPCR Description

Name	Description	Settings
UPC[3:0] Bits 3–0	**Pin Configuration—**Each bit determines whether its associated pin functions as UART or GPIO.	0 = GPIO (default). 1 = UART

UDDR UART Data Direction Register **$0020_408C**

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
													UDD3 ($\overline{\text{CTS}}$)	UDD2 ($\overline{\text{RTS}}$)	UDD1 (TxD)	UDD0 (RxD)
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 11-12. UDDR Description

Name	Description	Settings
UDD[3:0] Bits 3–0	**UART Data Direction—**Each of these bits determines the data direction of the associated pin if it is configured as GPIO.	0 = Input (default). 1 = Output.

UPDR UART Port Data Register **$0020_408E**

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
													UPD3 ($\overline{\text{CTS}}$)	UPD2 ($\overline{\text{RTS}}$)	UPD1 (TxD)	UPD0 (RxD)
RESET	0	0	0	0	0	0	0	0	0	0	0	0	—	—	—	—

Table 11-13. UPDR Description

Name	Description
UPD[3:0] Bits 3–0	**UART Port GPIO Data [3:0]—**Each of these bits contains data for the corresponding UART pin if it is configured as GPIO. Writes to UPDR are stored in an internal latch, and driven on any port pin configured as an output. Reads of this register return the value sensed on input pins and the latched data driven on outputs

Chapter 12
Smart Card Port

The Smart Card Port (SCP) is a serial communication channel designed to obtain user information such as identification. It is a customized UART with additional features for the SCP interface, as specified by ISO 7816-3 and GSM 11.11. Typically, a DSP56652 application uses this port to obtain subscriber information, and a smart card containing this information is referred to as a Subscriber Interface Module (SIM). Figure 12-1 presents a block diagram of the SCP.

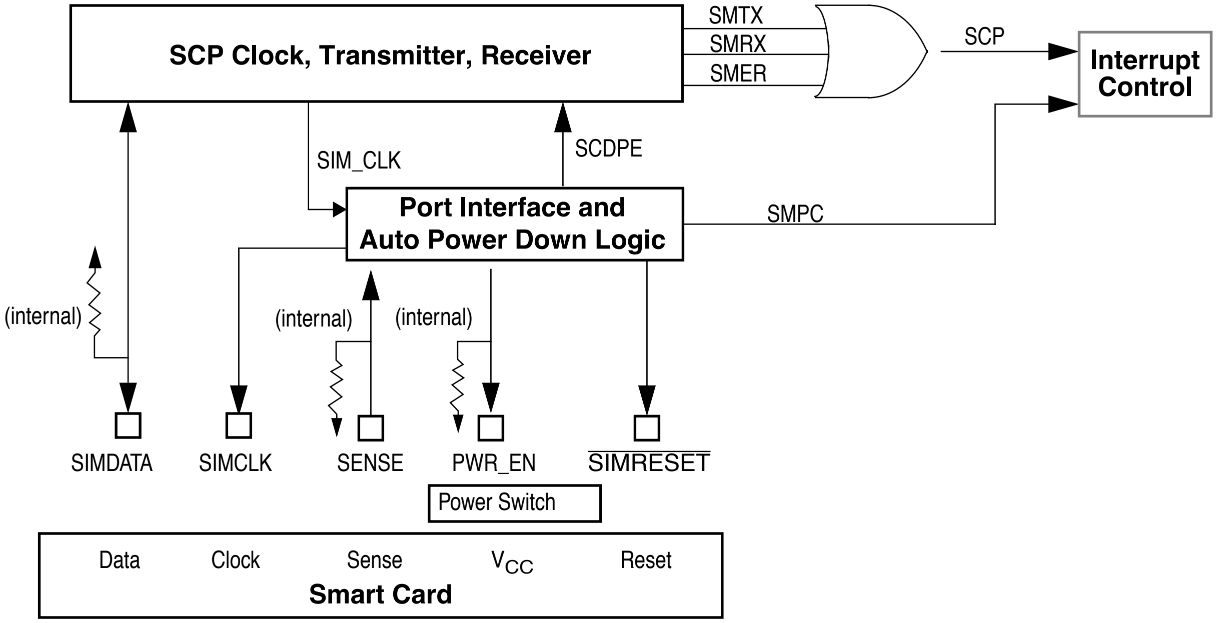


Figure 12-1. Smart Card Port Interface

Systems that do not require the SCP can configure the port as GPIO.

12.1 SCP Architecture

This section gives an overview of the SCP pins, data communication, and auto power-down circuitry.

12.1.1 SCP Pins

The SCP provides the following five pins to connect to a smart card:

- SIMDATA—a bidirectional pin on which transmit and receive data are multiplexed.
- SIMCLK—an output providing the clock signal to the smart card.
- SENSE—an input indicating if a smart card is inserted in the interface.
- $\overline{\text{SIMRESET}}$—an output that resets the smart card logic.
- PWR_EN—an output that enables an external power supply for the smart card.

The five pins can function as GPIO if the SCP function is not required. Because SCP operation requires all five pins, they cannot be configured for GPIO individually.

12.1.2 Data Communication

The SCP contains a quad-buffered receiver FIFO and a double-buffered transmitter. A single register serves both as a write buffer for transmitted data and a read buffer for received data. Reading the register clears an entry in the receive FIFO, and writing the register enters a new character to be transmitted. Three flags and optional interrupts are provided for FIFO not empty, FIFO, full, and FIFO overflow. The transmitter provides two flags and optional interrupts for character transmitted and TX buffer empty.

The SCP employs an asynchronous serial protocol containing one start bit, eight data bits, a parity bit and two stop bits. The polarity of the parity bit can established either by programming a register or, in the initial Character mode, by hardware at the beginning of each communication session. Both the card and the port can indicate receiving a corrupted frame (no stop bit) by issuing a NACK signal (pulling the SIMDATA pin low during the stop bit period). The SCP can also issue a NACK to the card when its receive buffer overflows to avoid losing further data, when it receives incorrect parity, and when it receives incorrect protocol data in Initial Character mode. Flags and optional interrupts are provided for the three NACK signals. The receiver also has flags and optional interrupts to indicate parity error, frame error, and receiver overrun.

The SCP generates a primary data clock, SIM_CLK, which is further divided to generate the bit rate. SIM_CLK also drives the SIMCLK pin, which can be synchronously pulled low by software.

12.1.3 Power Up/Down

A transition on the SENSE pin triggers both power up and power down sequences. Power up is done under software control, while power down can be controlled either by software or hardware.

12.2 SCP Operation

This section describes SCP activation and deactivation, clock generation, data transactions, and low power mode operation. A summary of the various SCP interrupts is also provided.

12.2.1 Activation/Deactivation Control

The smart card power up and power down sequences are specified in ISO 7816-3 and GSM 11.11. The signals and control bits provided by the DSP56652 to implement these sequences are illustrated in Figure 12-2 and described below.

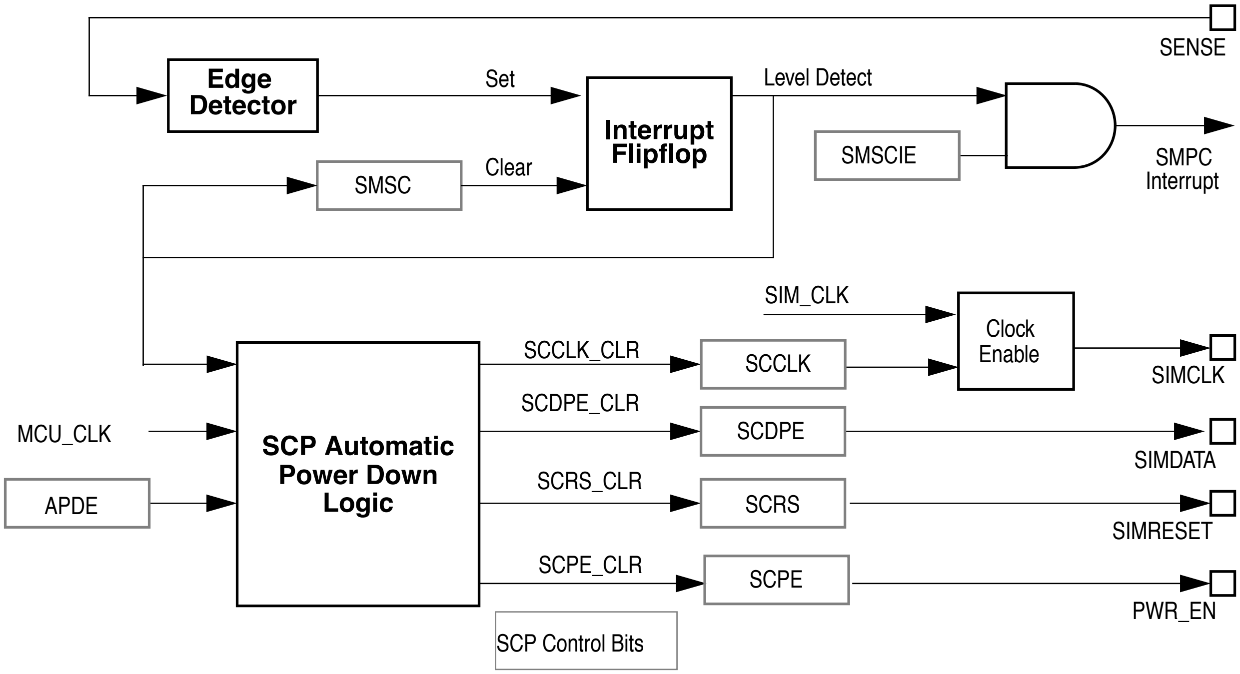


Figure 12-2. SCP: Port Interface and Auto Power Down Logic

When the port is enabled, the SENSE input detects insertion and removal of the smart card, initiating SCP activation and deactivation. Inserting the card pulls the SENSE pin low, and removing the card pulls the pin high. The SENSE pin state is reflected in the SCSP bit in the SCP Status Register (SCPSR). A rising or falling edge on the SENSE pin sets the SMSC flag in the SCPSR, and can generate an interrupt if the SMSCIE bit in the SCP Interrupt Enable Register (SCPIER) is set.

The power up sequence specified in ISO 7816 is implemented by the DSP56652 as follows:

1. The $\overline{\text{SIMRESET}}$ pin is asserted (pulled low) by clearing the SCRS bit in the Smart Card Activation Control. Register (SCACR).
2. The smart card is powered up. The SCPE bit in the SCACR can be set to turn on an external power supply for the card.
3. The SIMDATA pin is put in the reception mode (tri-stated) by setting the SCDPE bit in the SCACR.
4. The SCP drives a stable, glitch-free clock (SIM_CLK) on the SIMCLK pin by setting the SCCLK bit in the SCACR.
5. $\overline{\text{SIMRESET}}$ is deasserted by clearing the SCSR bit.

The power down sequence specified in ISO 7816 is implemented by the DSP56652 as follows:

1. $\overline{\text{SIMRESET}}$ is asserted by setting the SCRS bit.
2. SIMCLK is turned off (pulled low) by clearing the SCCLK bit.
3. SIMDATA transitions from tristate to low by clearing the SCDPE bit.
4. SIM V_{CC} is powered off. The SCPE bit is cleared if it was used to activate an external power supply.

The power down sequence can be performed in hardware by setting the APDE bit in the SCACR. The deactivation sequence is initiated by a rising edge on the SENSE pin so that the sequence can be completed before the card has moved far enough to lose connections with the contacts. The SCACR control bits in the above power down sequence are adjusted automatically.

12.2.2 Clock Generation

SCP clock operation is illustrated in Figure 12-3 on page 12-5. The SCP generates its primary data clock, SIM_CLK, by dividing CKIH by four or five, depending on the state of the CKSEL bit in the Smart Card Port Control Register (SCPCR). To determine the bit rate, SIM_CLK is further divided by 372 (normal mode) or 64 (speed enhancement mode), controlled by the SIBR bit in the SCPCR.

SIM_CLK is also gated to the smart card through the SIMCLK pin. The pin can be pulled low to save power by clearing the SCCLK bit in the SCACR.

12.2.3 Data Transactions

This section describes the SCP data format, reception, and transmission. A summary of NACK timing is also included. Data paths are shown in Figure 12-3.

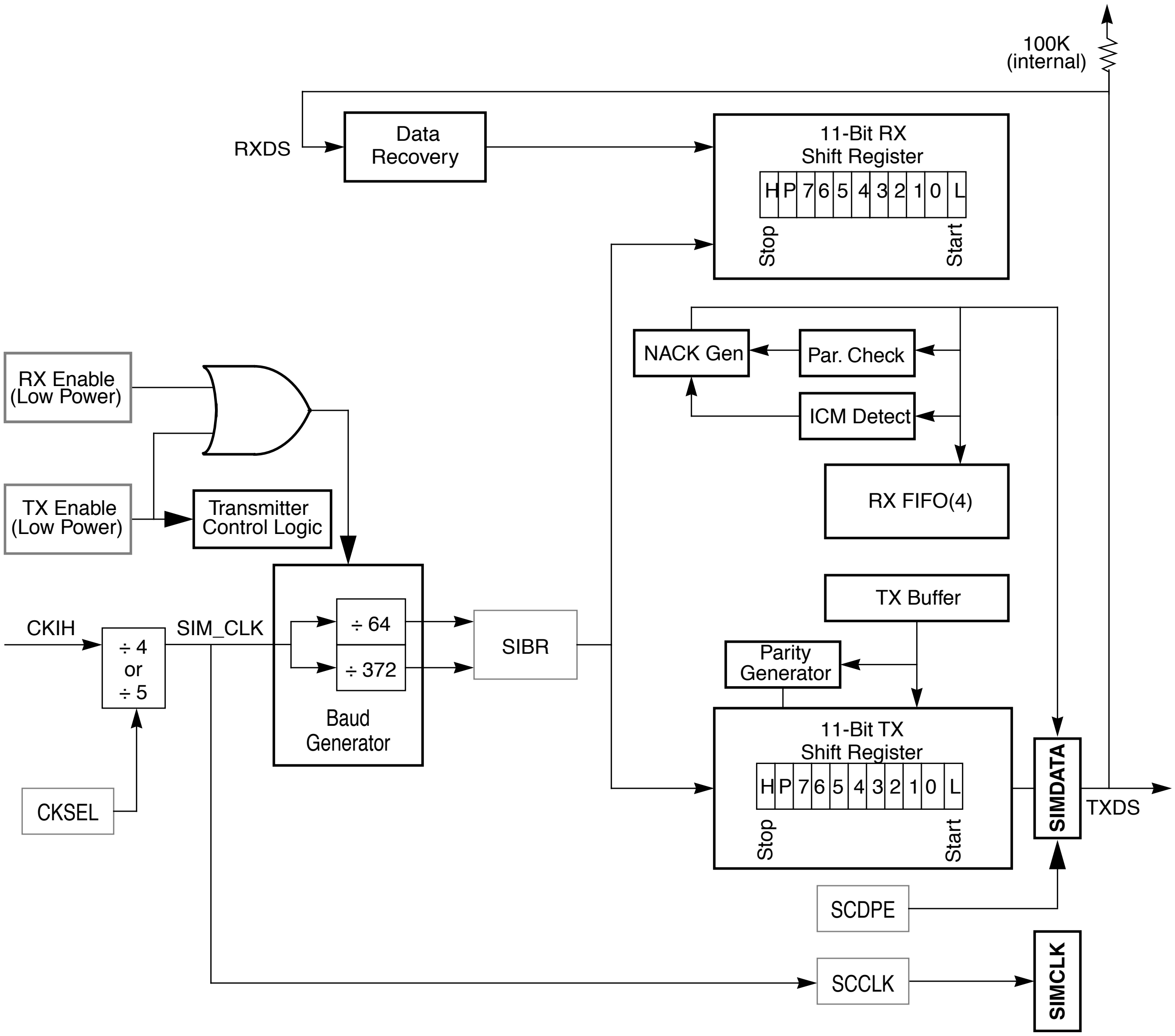


Figure 12-3. SCP: Clocks and Data

12.2.3.1 Data Format

The SCP data format and protocol are compatible with ISO 7816. The data format is fixed at one start bit, eight data bits, one parity bit, and two stop bits. Either receiver can overlay a NACK during the stop bit period to indicate an error by pulling the SIMDATA pin low. The SCP generates the NACK in hardware to save software overhead.

Odd/even parity is determined by the SCPT bit in the SCPCR. This bit can be explicitly written or adjusted automatically by the first smart card transmission after the card is inserted. In the latter mode, referred to as the initial character mode, the first character sent by the smart card is either $03 to indicate odd parity, or $3B to indicate even parity, and the parity bit in the frame is set. The initial character mode is selected by setting the SCIC bit in the SCPCR.

12.2.3.2 SIMDATA Pin

The SIMDATA pin serves as both transmitter and receiver for the SCP. The transmitter and receiver are enabled by the SCTE and SCRE pins respectively in the SCPCR. To avoid contention on the pin, only one of these bits should be set at a given time. If both bits are cleared, the clock input to the baud generator is disabled. The first transaction after the smart card is inserted is always from card to SCP, so it is recommended that SCRE be set and SCTE cleared as part of initialization and after the card is removed.

12.2.3.3 Data Reception

When the smart card is inserted and the power up sequence is complete, the SCP puts the SIMDATA pin in tristate mode to receive the first transmission from the card. The pin is initially at a logic one. If the pin goes to a logic low, and the receiver detects a qualified start bit, it proceeds to decode the succeeding transitions on the SIMDATA pin, monitoring for eight data bits, two stop bits, and correct parity. When a complete character is decoded, the data is written to the next available space in the four-character receive FIFO. The MCU reads the data at the top of the FIFO by reading the SCP Data Register (SCPDR), and the FIFO location is cleared.

Two receive conditions can be flagged in the SCPSR:

1. If the FIFO is empty when the first character is received, the SCFN bit is set. An interrupt is generated if the SCFNIE bit in the SCPIER is set.
2. If the received character fills the FIFO, the SCFF bit is set. An interrupt is generated if the SCFFIE bit in the SCPIER is set.

Three receive error conditions can also be flagged in the SCPSR:

1. A parity error is flagged by setting the SCPE bit. If the NKPE bit in the SCPCR is set, a NACK is sent to the smart card.
2. A frame error is flagged if the stop bit is not received by setting the SCFE bit.
3. If the FIFO is full when another character is received, the SCOE flag is set to indicate an overrun. If the NKOVR bit in the SCPCR is set, a NACK is sent to the smart card. The new character is not transferred to the FIFO and is overwritten if another character is received before the FIFO is read.

Any of the three error conditions generates an interrupt if the SCREIE bit in the SCPIER is set.

12.2.3.4 Data Transmission

To send a character, the MCU should clear the SCRE bit and set the SCTE bit to enable transmission. The MCU then writes to the SCPDR, the data is stored in a transmit buffer, and the SCP transmits the data to the card over the SIMDATA pin. The SCP outputs a start bit, eight character bits (least significant bit first), a parity bit, and two stop bits. If the smart card detects a parity error in the transmission it sends a NACK back to the SCP, the SCP alerts the MCU of the failure by setting the TXNK bit in the SCPSR, and the MCU must retry the transmission by writing the same data to SCPDR. When a frame has been transmitted, the transmit buffer is cleared and the SCTC flag in the SCPSR is set; if the SCTCIE bit in the SCPIER has been set, an interrupt is generated. Although a transmission in progress will complete if the transmitter is disabled, it is recommended that software waits until SCTC is set before clearing the SCTE bit.

12.2.3.5 NACK Timing

The following is a summary of the timing for NACK signals as specified in ISO 7816. The unit of time used by the specification is the Elementary Time Unit, or etu, which is defined as one bit time.

A NACK pulse is generated at 10.5 etu’s after the start bit. The width of the NACK pulse is 1 to 2 etu’s. The NACK should be sampled at 11 etu’s after the start bit. If a NACK pulse is received, the character should be retransmitted a minimum of 2 etu’s after the detection of the error. The start of the repeated character should be a minimum of 2 etu’s after the detection of the error bit. Figure 12-4 shows timing of the data format.

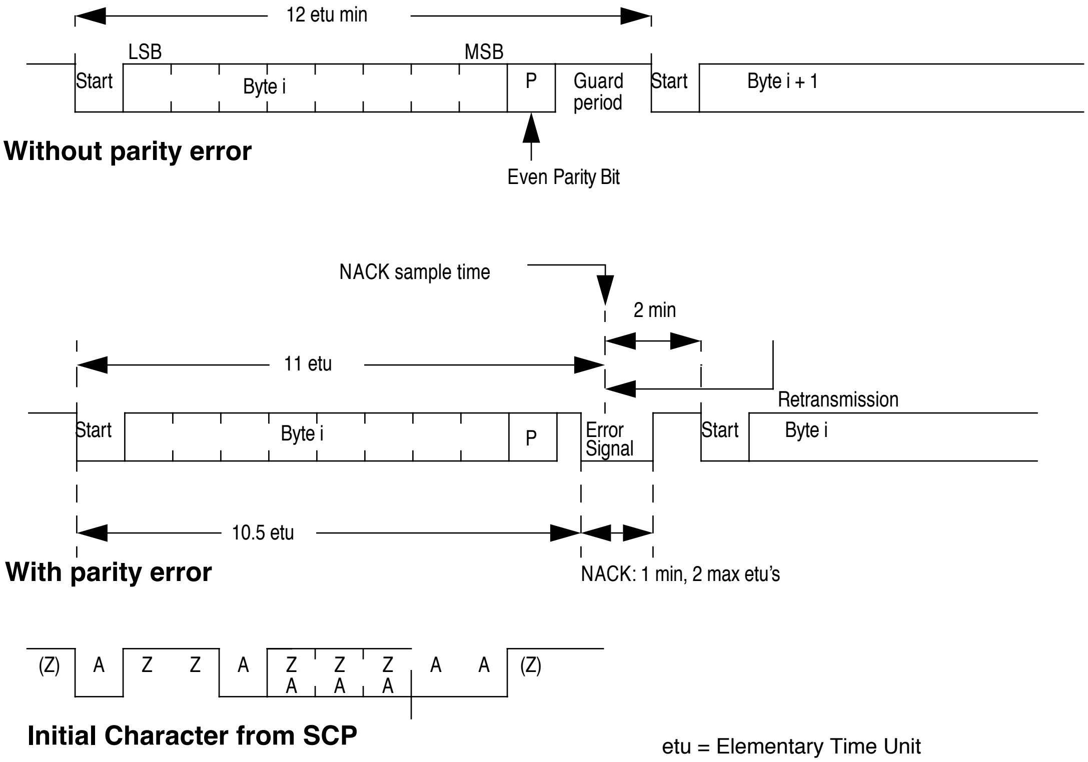


Figure 12-4. SCP Data Formats

12.2.4 Low Power Modes

If the DOZE bit in the SCPCR register is set when the MCU enters DOZE mode, the SCP completes the current transmit/receive transaction, then gates off the receive and transmit clocks. However, the clocks to the Automatic Power Down and the SENSE debouncer circuits remain enabled to allow the SCP to initiate an automatic power down if the smart card is removed during DOZE mode. All state machines and registers retain their current values. When exiting DOZE mode the SCP reenables all its clocks, and resumes operation with its previously retained state. If the DOZE bit in the SCPCR is cleared, the SCP continues full operation in DOZE mode.

When the MCU enters STOP mode, the SCP gates off the CKIH clock and freezes all state machines and registers. Software must complete all transmit/receive transactions and power down the SCP before entering STOP mode. When exiting the Stop mode, the SCP reenables all its clocks, and resumes operation with its previously retained state.

12.2.5 Interrupts

The SCP generates two interrupts to the MCU:

- SMPC is generated when the smart card position is changed (i.e., inserted or removed).
- SCP indicates all other SCP interrupt conditions generated by transmission, reception and errors. Error interrupts have priority over transmit and receive interrupts. Receive interrupts are not cleared until the SCPDR is read.

Figure 12-5 illustrates the sources and conditions that generate the various SCP interrupts.

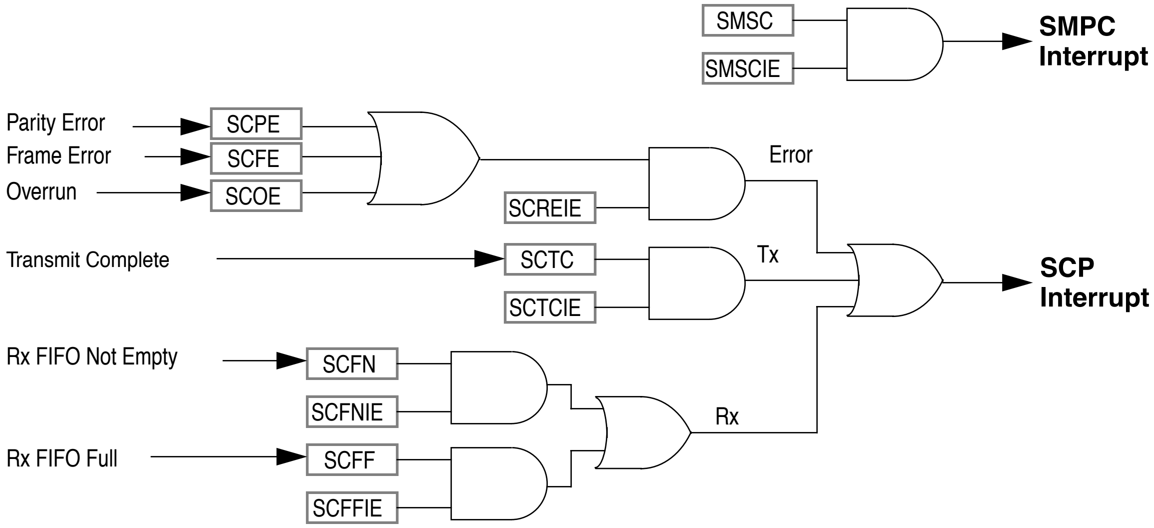


Figure 12-5. SCP Interrupts

12.3 SCP Registers

Table 12-1 is a summary of the SCP control and GPIO registers, including the acronym, bit names, and address (least-significant halfword) of each register. The most-significant halfword of all register addresses is $0020.

Table 12-1. SCP Register Summary

SCPCR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$B000							CKSEL	NKOVR	DOZE	SIBR	SCSR	SCPT	SCIC	NKPE	SCTE	SCRE

SCACR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$B002												SCCLK	SCRS	SCDPE	SCPE	APDE

SCPIER	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$B004												SCTCIE	SCFNIE	SCFFIE	SCREIE	SMSCIE

SCPSR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$B006							SCFF	SCFN	SCTY	SCTC	TXNK	SCPE	SCFE	SCOE	SMSC	SCSP

SCPDR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$B008									SCPD[7:0]							

SCPPCR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$B00A	SMEN						PDIR[4:0]					PDAT[4:0]				

12.3.1 SCP Control Registers

SCPCR SCP Control Register **$0020_B000**

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
							CKSEL	NKOVR	DOZE	SIBR	SCSR	SCPT	SCIC	NKPE	SCTE	SCRE
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 12-2. SCPCR Description

Name	Description	Settings
CKSEL Bit 9	**Clock Select—**Determines if the CKIH divisor that generates SIM_CLK is 4 or 5.	0 = CKIH divided by 5 (default). 1 = CKIH divided by 4.
NKOVR Bit 8	**NACK on Receiver Overrun—**Enables overrun checking and reporting.	0 = NACK not generated 1 = NACK is generated on overrun error.
DOZE Bit 7	**DOZE Mode—**Controls SCP operation in DOZE mode.	0 = SCP ignores DOZE mode (default). 1 = SCP stops in DOZE mode.
SIBR Bit 6	**SIM Baud Rate—**Determines the SIM_CLK divisor to generate the SIM baud clock.	0 = Baud rate = SIM_CLK ÷ 372 (default). 1 = Baud rate = SIM_CLK ÷ 64.
SCSR Bit 5	**SCP System Reset—**Setting this bit resets the SCPSR and state machines. Other registers are not affected. If the SCSR bit is set while a character is being sent or received, the transmission is completed before reset occurs.	
SCPT Bit 4	**SCP Parity Type—**Selects odd or even parity. In initial character mode, hardware adjusts this bit automatically	0 = Even parity (default). 1 = Odd parity.
SCIC Bit 3	**SCP Initial Character Mode—**Setting this bit implements initial character mode, in which parity is determined by the first character sent by the card after it is inserted (see page 12-6).	0 = Parity determined by writing SCPT bit (default). 1 = Parity determined by initial character from card.
NKPE Bit 2	**NACK on Parity Error—**Determines if a NACK signal is sent (SIMDATA pin pulled low) if a parity error is detected. This affects both the SCP and the smart card.	0 = No NACK sent (default). 1 = NACK sent on parity error.
SCTE Bit 1	**SCP Transmit Enable—**Setting this bit allows data written to the transmit buffer to be loaded to the transmit shift register and shifted out on the SIMDATA pin. A transmission in progress when SCTE is cleared is completed before the transmitter is disabled.	0 = Disabled (default). 1 = Enabled.
SCRE Bit 0	**SCP Receive Enable—**Setting this bit allows data received on the SIMDATA pin to be shifted into the receive shift register and loaded to the receive FIFO. A reception in progress when SCRE is cleared is completed before the receiver is disabled.	0 = Disabled (default). 1 = Enabled.

SCACR Smart Card Activation Control Register **$0020_B002**

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
												SCCLK	SCRS	SCDPE	SCPE	APDE
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 12-3. SCACR Description

Name	Description	Settings
SCCLK Bit 4	**Smart Card Clock—**Setting this bit drives SIM_CLK to the smart card on the SIMCLK pin. Cleared by software or automatically after the card is removed if the APDE bit is set.	0 = SIMCLK pulled low (default). 1 = SIMCLK driven by the SIM_CLK signal.
SCRS Bit 3	**Smart Card Reset—**This bit drives the $\overline{\text{SIMRESET}}$ pin. It is controlled automatically after the card is removed if the APDE bit is set.	0 = $\overline{\text{SIMRESET}}$ pulled low (default). 1 = $\overline{\text{SIMRESET}}$ driven high.
SCDPE Bit 2	**Smart Card Data Pin Enable—**Setting this bit allows the SIMDATA pin to function as a receiver or transmitter. It is cleared automatically after the card is removed if the APDE bit is set.	0 = SIMDATA pulled low (default). 1 = SIMDATA functions as SCP transmit or receive pin.
SCPE Bit 1	**Smart Card Power Enable—**This bit drives the PWR_EN pin, which can switch on an external power supply to power the smart card. It is cleared automatically after the card is removed if the APDE bit is set.	0 = PWR_EN pulled low (default). 1 = PWR_EN driven high.
APDE Bit 0	**Auto Power Down Enable—**Setting this bit allows hardware to control the SCP pins to perform the power down sequence automatically after the smart card is removed.	0 = Software performs power down sequence (default). 1 = Hardware automatically performs power down sequence.

SCPIER SCP Interrupt Enable Register **$0020_B004**

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
												SCTCIE	SCFNIE	SCFFIE	SCREIE	SMSCIE
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Note: In addition to the individual interrupt enable bits in the SCPIER, the following bits must also be set in order to generate the respective interrupts (see page 7-7):

SIM Sense Change—either ESMPD in the NIER or EFSMPD in the FIER
All other interrupts—either ESCP in the NIER or EFSCP in the FIER.

Table 12-4. SCPIER Description

Name	Description	Settings
SCTCIE Bit 4	**SCP Transmit Complete Interrupt Enable—**Allows an interrupt to be generated when the SCTC bit in the SCPSR is set.	0 = Interrupt disabled (default). 1 = Interrupt enabled.
SCFNIE Bit 3	**SCP Receive FIFO Not Empty Interrupt Enable—**Allows an interrupt to be generated when the SCFN bit in the SCPSR is set.	
SCFFIE Bit 2	**SCP Receive FIFO Full Interrupt Enable—**Allows an interrupt to be generated when the SCFF bit in the SCPSR is set.	
SCREIE Bit 1	**SCP Receive Error Interrupt Enable—**Allows an interrupt to be generated when the SCPE, SCFE, or SCOE bit in the SCPSR is set.	
SMSCIE Bit 0	**SIM SENSE Change Interrupt Enable—**Allows an interrupt to be generated when the SMSC bit in the SCPSR is set.	

SCPSR SCP Status Register **$0020_B006**

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
							SCFF	SCFN	SCTY	SCTC	TXNK	SCPE	SCFE	SCOE	SMSC	SCSP
RESET	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	—

Table 12-5. SCPSR Description

Name	Type[1]	Description	Settings
SCFF Bit 9	R/RDC	**SCP Receive FIFO Full—**Set when all four receive FIFO characters are filled. Cleared by reading the SCPDR.	0 = FIFO can receive more data (default). 1 = FIFO full.
SCFN Bit 8	R/RDC	**SCP Receive FIFO Not Empty—**Set when the FIFO contains at least one character. Cleared by reading the SCPDR.	0 = FIFO empty (default). 1 = FIFO not empty.
SCTY Bit 7	R/WDC	**SCP Transmit Register Empty—**Set when the transmit data register is empty, signalling the MCU that a character can be written to SCPDR. Cleared by reading the SCPDR. Normally, the MCU uses the SCTC bit rather than SCTY to determine when the next character can be sent.	0 = Transmit register not empty. 1 = Transmit register empty (default).
SCTC Bit 6	R/WDC	**SCP Transmit Complete—**Set after transmitting the second stop bit of a frame (one additional bit time later if a NACK is received). Cleared by writing the SCPDR.	0 = Next transmission not complete. 1 = Transmission complete (default).
TXNK Bit 5	R/WDC	**NACK Received for Transmitted Word—**Set when a NACK is detected while transmitting a character. Cleared by writing the SCPDR or by hardware reset. TXNK is set simultaneously with SCTC.	0 = No NACK (default). 1 = NACK received.
SCPE Bit 4	R/1C	**SCP Parity Error—**Set when an incorrect parity bit has been detected in a received character. Cleared by writing with 1.	0 = No parity error (default). 1 = Parity error detected.
SCFE Bit 3	R/1C	**SCP Frame Error—**Set when an expected stop bit in a received frame is sampled as a 0. Cleared by writing with 1.	0 = No frame error (default). 1 = Frame error detected.
SCOE Bit 2	R/1C	**SCP Overrun Error—**Set when a new character has been shifted in to the receive buffer and the RX FIFO is full. Cleared by writing with 1.	0 = No overrun error (default). 1 = Overrun error detected.

Table 12-5. SCPSR Description (Continued)

Name	Type[1]	Description	Settings
SMSC Bit 1	R/1C	**SIM Sense Change—**Set simultaneously with the SMPC interrupt when the smart card (SIM) is inserted or removed, generating a falling or rising edge on the SENSE pin. Cleared by writing with 1.	0 = No change on SENSE pin (default). 1 = Edge on SENSE pin detected.
SCSP Bit 0	R	**SCP SENSE Pin—**Reflects the current state of the SCP SENSE pin.	

1. R = Read only
R/RDC = Read/Read SCPDR to clear
R/WDC = Read/Write SCPDR to clear
R/1C = Read; write with 1 to clear (write with 0 ignored).

SCPDR SCP Data Register **$0020_B008**

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
	▨	▨	▨	▨	▨	▨	▨	▨	SCPD[7:0]							
RESET	0	0	0	0	0	0	0	0	—	—	—	—	—	—	—	—

Table 12-6. SCPDR Description

Name	Description
SCPD[7:0] Bits 7–0	**SCP Data Buffer—**This field is used both to transmit and receive SCP data. Writing to the SCPDR enters a new character in the transmit buffer; reading the SCPDR register reads the character at the top of the RX FIFO.

12.3.2 GPIO

The five SCP pins can function as GPIO. GPIO functions are governed by the SCPPCR register. The data direction and port GPIO data fields correspond to the SCP pins as shown in Table 12-7.

Table 12-7. SCP Pin GPIO Bit Assignments

GPIO Bit #	SCP Pin
9, 4	PWR_EN
8, 3	$\overline{\text{SIMRESET}}$
7, 2	SIMDATA
6, 1	SENSE
5, 0	SIMCLK

SCPPCR SCP Port Control Register **$0020_B00A**

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
	SMEN	▨	▨	▨	▨	▨	SCPDD[4:0]					SCPPD[4:0]				
RESET	0	0	0	0	0	0	0	0	0	0	0	—	—	—	—	—

Table 12-8. SCPPCR Description

Name	Description	Settings
SMEN Bit 15	**SCP Port Enable—**Determines if all five smart card pins function as SCP pins or GPIO.	0 = GPIO (default). 1 = SCP.
SCPDD[4:0] Bits 9–5	**SCP Data Direction—**Each of these bits determines the data direction of the associated pin if it is configured as GPIO.	0 = Input (default). 1 = Output.
SCPPD[4:0] Bits 4–0	**SCP Port GPIO Data [4:0]—**Each of these bits contains data for the corresponding SCP pin if it is configured as GPIO. Writes to these bits are stored in an internal latch, and driven on any port pin configured as an output. Reads of these bits return the value sensed on input pins and the latched data driven on outputs	

Chapter 13
Keypad Port

The keypad port (KP) is a 16-bit peripheral designed to ease the software burden of scanning a keypad matrix. It works with any sized matrix up to eight rows by eight columns. With appropriate software support, keypad logic can detect, debounce, and decode one or two keys pressed simultaneously. A key press generates an interrupt that can bring the MCU out of low power modes.

The KP is designed for a keypad matrix that shorts intersecting row and column lines when a key is depressed. It is not intended for use with other switch configurations.

13.1 Keypad Operation

This section describes KP pin configuration, software polling required to determine a valid keypress, low power operation, and noise suppression circuitry. Figure 13-1 is a block diagram of the keypad port.

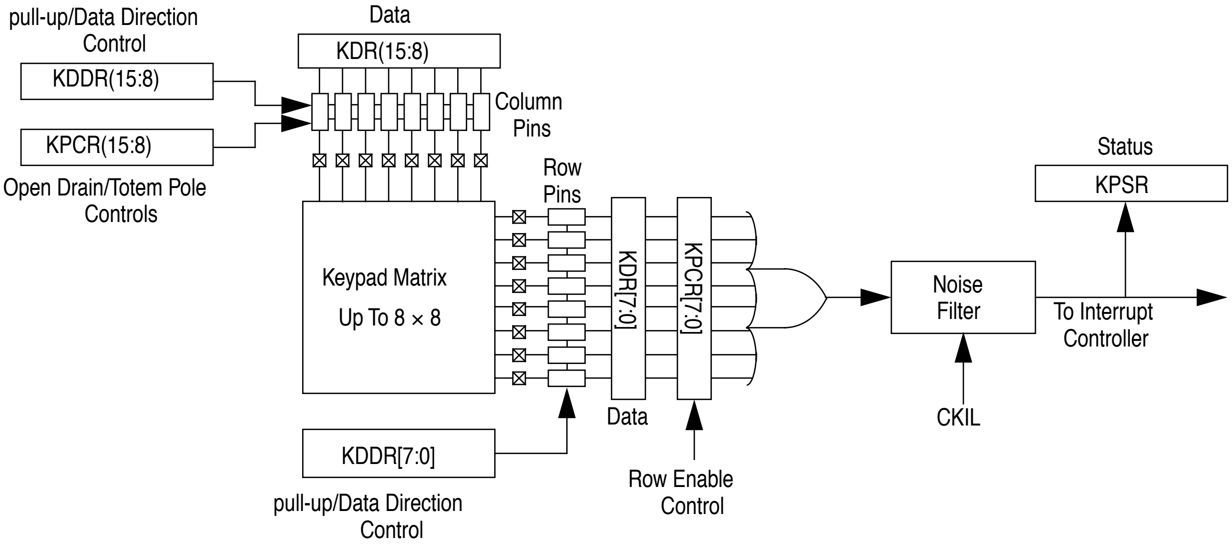


Figure 13-1. Keypad Port Block Diagram

13.1.1 Pin Configuration

The KP provides 16 pins to support any keypad configuration up to eight rows and eight columns. Five of these pins, ROW7–ROW 5 and COL7–COL6, are multiplexed with other functions and require specific settings in the General-Purpose Configuration Register for keypad operation. (Refer to Table 4-13 on page 4-18.) Any pins not used for the keypad are available as GPIO pins.

13.1.1.1 Column Pins

Each column pin intended for keypad operation must be configured as an output by setting the corresponding KCD bit in the Keypad Port Data Direction Register (KDDR), and for keypad rather than GPIO operation by setting the corresponding KCO bit in the Keypad Port Control Register (KPCR). Column pins configured for keypad operation are open drain with on-board pull-up resistors; column pins configured as GPIO outputs have totem pole drivers with the pull-up resistors disabled. These configurations are summarized in Table 13-1.

Table 13-1. Keypad Port pull-up Resistor Control

KDDR[15:8]	KPCR[15:8]	Pin Function	pull-up Resistors
0	x	Input	Enabled
1	0	Output—totem pole	Disabled
1	1	Output—open drain	Enabled

13.1.1.2 Row Pins

Row pins intended for keypad operation must be configured as inputs by clearing the corresponding KRD bits in the KDDR, and for keypad operation (rather than GPIO) by setting the corresponding KRE bits in the KPCR. When pulled low, each row pin configured for keypad operation sets the KPKD bit in the Keypad Status Register (KPSR) and generates an interrupt. Row pins configured as GPIO do not set the status flag or generate an interrupt when they are pulled low. The KPKD bit is cleared by reading the KPSR, then writing the KPKD bit with 1.

A discrete switch can be connected to any row input pin that is not part of the keypad matrix. The second terminal of the discrete switch is connected to ground. If the pin is configured as an input and for keypad operation, hardware detects closure of the switch and generates an interrupt if the corresponding row pin is configured for keypad operation.

Care should be taken not to configure a row pin for both KP operation and as an output. In this configuration a keypad interrupt is generated if the associated data bit in the Keypad Data Register (KPDR) is written with zero, pulling the pin low.

13.1.2 Keypad Matrix Polling

The keypad interrupt service routine typically includes a keypad polling loop to determine which key is pressed. This loop walks a 0 across each of the keypad columns by clearing the corresponding KCO bit, and reads the row values in the KPDR at each step. The process is repeated several times in succession, and the results of each pass compared with those from the previous pass. When several consecutive scans yield the same key closures, a valid key press has been detected. Software can then determine which switch is pressed and pass the value up to the next higher software layer.

13.1.3 Standby and Low Power Operation

The keypad does not require software intervention until a keypress is detected. Software can put the keypad in a standby state between keypresses to conserve power by clearing the KCO bits in the KPCR. Clearing the KCO bits turns off the open-drain mode in the corresponding column outputs, converting them to totem pole drivers, and disconnects the pull-up resistors, reducing standby current. The outputs are forced low by clearing the corresponding bits in the KPDR. Row inputs are left enabled. The MCU can then attend to other tasks or enter a low power mode.

The keypad port interrupts the MCU when a key is pressed, waking it up if it is in a low power mode. The MCU re-enables the open drain drivers, sets all the column strobes high, and runs the keypad polling routine to determine which key is pressed. Care should be taken to enable the open drain drivers before driving the columns high to avoid shorting power to ground through two or more switches.

13.1.4 Noise Suppression on Keypad Inputs

The noise suppression circuit illustrated in Figure 13-2 qualifies keypad closure signals to prevent false keypad interrupts. The circuit is a four-state synchronizer driven by CKIL. A KP interrupt is not generated until all four synchronizer stages have latched a valid key assertion, effectively filtering out any noise less than four clock cycles in duration. The interrupt signal is an S-R latch output that remains asserted until cleared by software. Once cleared, the interrupt and its reflection in the KPSR cannot be set again until a period of no key closure is detected. In this way, the hardware prevents multiple interrupts for the same key press with no software intervention.

Because the keypad interrupt signal is driven by the noise suppression circuit, CKIL must remain powered in low power modes for which the keypad is a wake-up source.

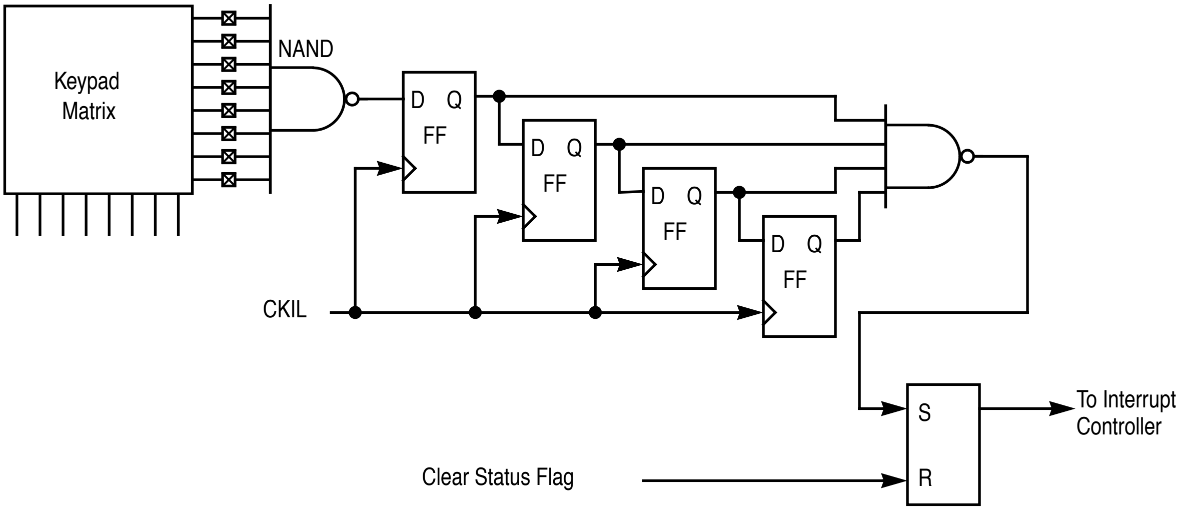


Figure 13-2. Glitch Suppressor Functional Diagram

13.2 Keypad Port Registers

Table 13-2 is a summary of the KP control and GPIO registers, including the acronym, bit names, and address (least-significant halfword) of each register. The most-significant halfword of all register addresses is $0020. All registers except KPSR are byte-addressable, with column bits in the most significant byte, and row bits in the least significant byte.

Table 13-2. Keypad Port Register Summary

KPCR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$A000	KCO[7:0]								KRE[7:0]							
KPSR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$A002																KPKD
KDDR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$A004	KCDD[7:0]								KRDD[7:0]							
KPDR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$A006	KCD[7:0]								KRD[7:0]							

KPCR Keypad Port Control Register $0020_A000

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
	KCO[7:0]								KRE[7:0]							
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 13-3. KPCR Description

Name	Description	Settings
KCO[7:0] Bits 15–8	**Keypad Column Strobe Open Drain Enable—**Each bit determines if the corresponding pin functions as a keypad column pin (strobe operation—open-drain output in normal operation, totem pole output in low power and standby modes) or GPIO (totem pole output only).	0 = GPIO (default). 1 = KP—open-drain output in normal operation.
KRE[7:0] Bits 7–0	**Keypad Row Interrupt Enable—**Each bit determines if the corresponding row pin functions as KP (generates an interrupt if pulled low) or GPIO (no interrupt). **Note:** Either the EKPD bit in the NIER or the EFKPD bit in the FIER must also be set in order to generate the keypad interrupts (see page 7-7).	0 = GPIO—interrupt disabled (default). 1 = KP—interrupt enabled.

KPSR Keypad Status Register $0020_A002

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
																KPKD
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 13-4. Generic Description

Name	Type[1]	Description	Settings
KPKD Bit 0	R/1C	**Keypad Keypress Detect—**This bit reflects the keypad interrupt status. It is set when a valid key closure has been detected, and cleared by reading the KPSR, then writing KPKD with 1.	0 = No valid keypress detected (default). 1 = Valid keypress detected.

1. R/1C = Read, or write with 1 to clear (write with 0 ignored).

KDDR Keypad Data Direction Register $0020_A004

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
	KCDD[7:0]								KRDD[7:0]							
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 13-5. KDDR Description

Name	Description	Settings
KCDD[7:0] Bits 15–8	**Keypad Column Pin Data Direction**	0 = Input (default). 1 = Output
KRDD[7:0] Bits 7–0	**Keypad Row Pin Data Direction** Each of these bits determines the data direction of the associated pin. Valid data should be written to the KPDR before any of these bits are configured as outputs.	

KPDR Keypad Port Data Register $0020_A006

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
	KCD[7:0]								KRD[7:0]							
RESET	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Table 13-6. KPDR Description

Name	Description	
KCD[7:0] Bits 15–8	**Keypad Column Data**	Each of these bits contains data for the corresponding keypad pin. Writes to KPDR are stored in an internal latch, and driven on any port pin configured as an output. Reads of this register return the value sensed on input pins and the latched data driven on outputs.
KRD[7:0] Bits 7–0	**Keypad Row Data**	

Chapter 14
Serial Audio and Baseband Ports

The Serial Audio Port (SAP) and the Baseband Port (BBP) are both DSP peripherals based on the synchronous serial interface (SSI) included in several other Motorola DSP devices. Each port supports full-duplex serial communication with a variety of serial devices, including one or more industry-standard codecs, other DSPs, microprocessors, and peripherals that implement the Motorola SSI. Features common to both the SAP and the BBP include the following:

- Independent transmit and receive sections that can operate with separate (asynchronous) or shared (synchronous) internal/external clocks and frame syncs
- TDM operation with either one slot per frame (normal mode) or up to 32 time slots per frame (network mode).
- Programmable word length (8, 12, or 16 bits).
- Program options for frame synchronization and clock generation

Features unique to one port include the following:

- The SAP contains a bit rate multiplier (BRM) to convert a 16.8 MHz input to a 16.834 MHz clock that can generate standard codec clock rates.
- The SAP includes a general-purpose timer.
- The BBP contains transmit and receive frame counters.

In addition, any or all of the pins in each port can be configured as GPIO.

Figure 14-1 and Figure 14-2 are block diagrams of the SAP and BBP respectively.

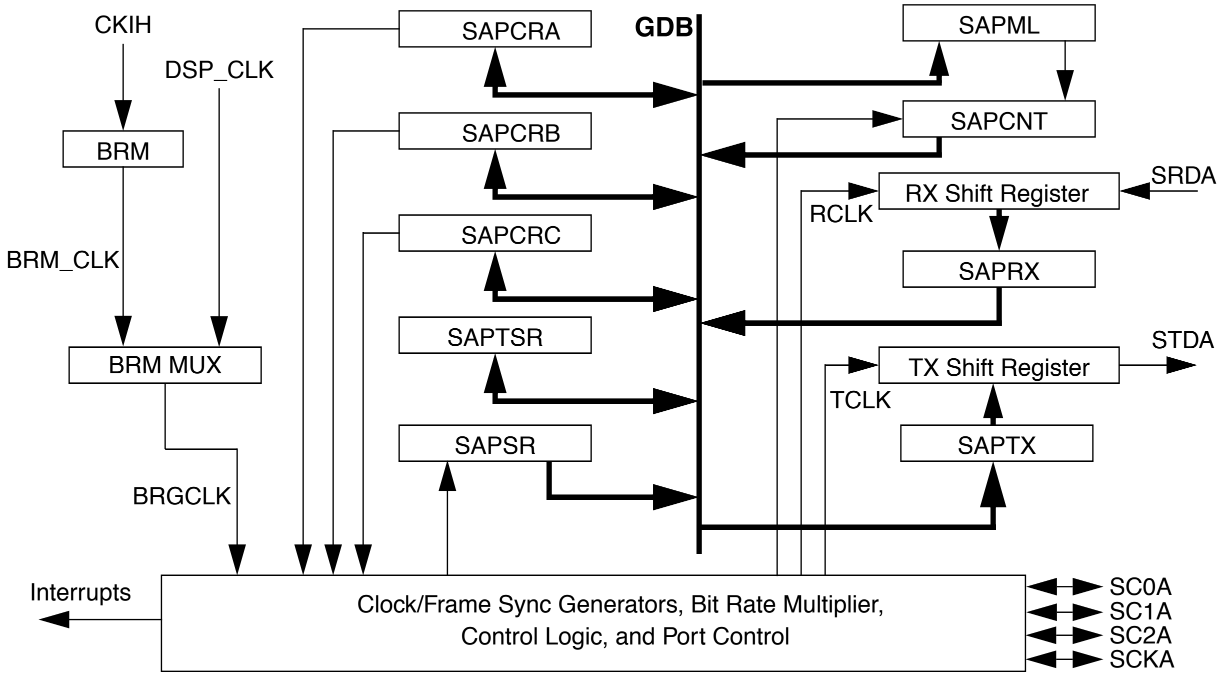


Figure 14-1. SAP Block Diagram

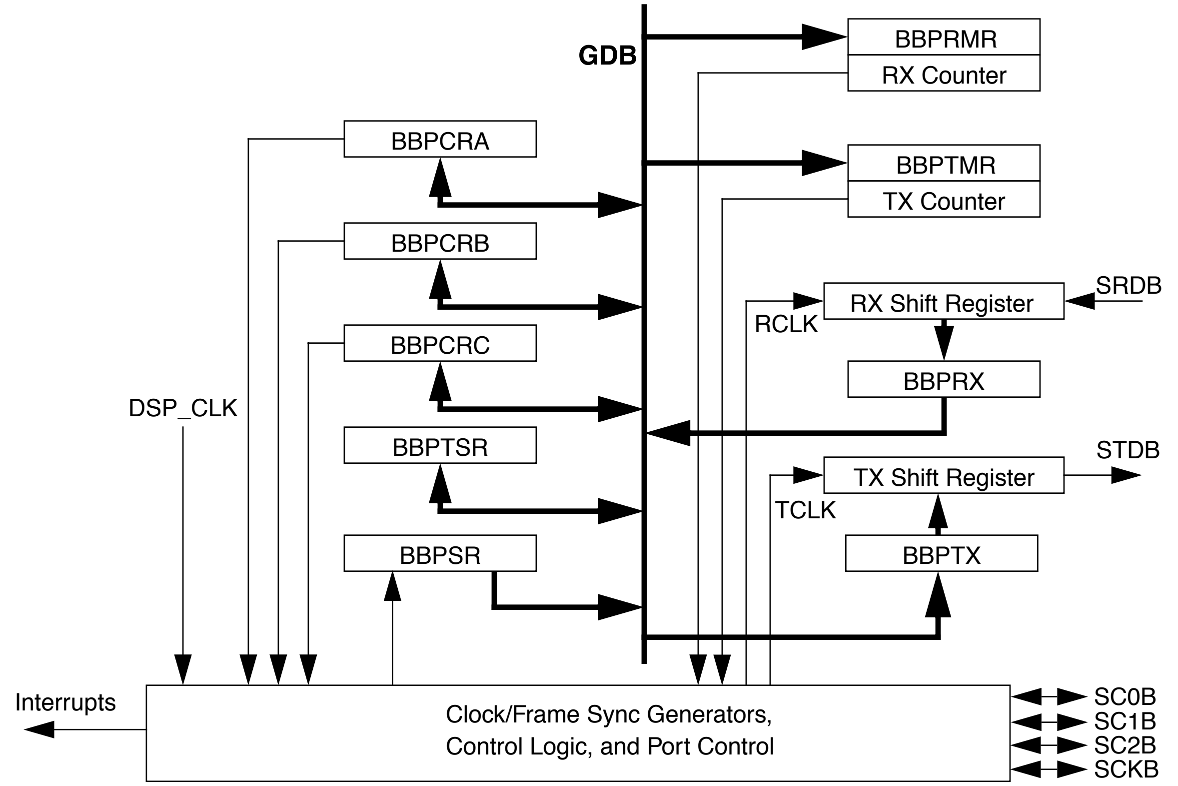


Figure 14-2. BBP Block Diagram

14.1 Data and Control Pins

Each of the ports contains six pins. The names and functions of these pins are summarized in Table 14-1.

Table 14-1. SAP and BBP Pins

SAP Pin	BBP Pin	Function	
		Asynchronous Mode	**Synchronous Mode**
SC0A	SC0B	Receiver Clock	Serial Flag 0
SC1A	SC1B	Receiver Frame Sync	Serial Flag 1
SC2A	SC2B	Transmitter Frame Sync	Tx and Rx Frame Sync
SCKA	SCKB	Transmitter Clock	Tx and Rx Clock
SRDA	SRDB	Serial Receive Data Pin	
STDA	STDB	Serial Transmit Data Pin	

The functions of the serial clock pin (SCK) and serial control pins (SC0–2) for each port depend on whether the port clock and frame sync signals operate independently (asynchronous mode) or are common (synchronous mode). Signal directions (input or output) for these four pins are determined by the Serial Control Pin Direction SCD[2:0] and Serial Clock Pin Direction (SCKD) bits in Control Register C for each port (SAPCRC and BBPCRC). Pins that are not used for SAP or BBP operation can be configured as GPIO. Pin functions are further described in the following sections:

- Receive and transmit clocks—Section 14.2 on page 14-3.
- Data transmission and reception—Section 14.4 on page 14-9.
- Serial flags—Section 14.3.4 on page 14-8.

14.2 Transmit and Receive Clocks

Several options are provided to configure the SAP and BBP transmit and receive bit clocks, including clock sources (internal or external), frequency, polarity, and BRM (SAP only).

14.2.1 Clock Sources

The transmit and receive clock source(s) can be either external or internal. For an external clock source, the pins functioning as clocks are configured as inputs. For an internal clock source, clock pins are configured as outputs. The BBP internal clock is derived from

DSP_CLK; the SAP internal clock is derived from either DSP_CLK or the Bit Rate Multiplier clock (BRM_CLK), as determined by the BRM bit in the SAP Control Register C (SAPCRC). Clock sources and pins are governed by SAPCRC/BBPCRC control bits SYN (which selects synchronous or asynchronous mode), SCKD, and SCD0, as shown in Table 14-2.

Table 14-2. SAP/BBP Clock Sources

SYN	SCKD	SCD0	Receive Clock Source	Receive Clock Out	Transmit Clock Source	Transmit Clock Out
Asynchronous Mode						
0	0	0	External, SC0A/B	—	External, SCKA/B	—
0	0	1	Internal	SC0A/B	External, SCKA/B	—
0	1	0	External, SC0A/B	—	Internal	SCKA/B
0	1	1	Internal	SC0A/B	Internal	SCKA/B
Synchronous Mode						
1	0	x	External, SCKA/B	—	External, SCKA/B	—
1	1	x	Internal	SCKA/B	Internal	SCKA/B

Note: Although an external serial clock can be independent of and asynchronous to the DSP system clock, its frequency must be less than or equal to one-third the DSP_CLK frequency.

14.2.2 Clock Frequency

The frequency of the internally-generated bit clock is determined by the source clock, an optional divide-by-8 prescaler, and a programmable prescale modulus, as shown in the following equation:

$$\text{Bit clock frequency} = \frac{\text{BRGCLK}}{2 \times (2\text{–PSR})^3 \times (\text{PM}+1)}$$

where BRGCLK =DSP_CLK (BBP)
DSP_CLK or BRM_CLK (SAP)
PSR = Prescaler (PSR) bit in Control Register A (SAPCRA or BBPCRA)
PM = Value of the Prescale Modulus (PM[7:0]) bits in SAPCRA or BBPCRA.

The minimum frequency is generated with the prescaler on (PSR=0) and the maximum prescale modulus (PM[7:0]=255), yielding

$$\text{Bit clock frequency} = \frac{\text{BRGCLK}}{2 \times (2)^3 \times (256)} = \frac{\text{BRGCLK}}{4096}$$

The combination of PSR=1 and PM[7:0]=0 is reserved, so the maximum frequency is generated with PSR=1 and PM[7:0]=1, yielding

$$\text{Bit clock frequency} = \frac{\text{BRGCLK}}{2 \times (1)^3 \times (2)} = \frac{\text{BRGCLK}}{4}$$

If the bit clock is supplied externally, the maximum allowed frequency is DSP_CLK ÷ 3.

14.2.3 Clock Polarity

The Clock Polarity (CKP) bit in the SAPCRC or BBPCRC determines the clock edge on which data and frame sync are clocked out and latched in. When the CKP bit is cleared, data and frame sync are clocked out on the rising edge of the transmit bit clock and latched in on the falling edge of the receive bit clock. When the CKP bit is set, data and frame sync are clocked out on the falling edge of the transmit bit clock and latched in on the rising edge of the receive bit clock.

14.2.4 Bit Rate Multiplier (SAP Only)

The BRM provides a way for systems with a CKIH of 16.8 MHz to generate a SAP bit clock with the standard codec frequency of 2.048 MHz. The BRM applies a 512/525 multiplier to DSP_CLK to generate a 16.384 MHz BRM_CLK from a 16.8 MHz input. To generate a 2.048 MHz bit clock, perform the following steps:

1. Set the BRM bit in the SAPCRC to select BRM_CLK rather than DSP_CLK as the bit rate clock source BRGCLK.
2. Set the PSR bit in SAPCRA to disable the prescaler.
3. Write $03 to the PM[7:0] bits in the SAPCRA to divide the 16.384 MHz BRGCLK by four.

14.3 TDM Options

Several facets of SAP and BBP TDM operation can be controlled, including synchronous or asynchronous mode, frame configuration, frame sync parameters, serial I/O flags, and interrupts.

14.3.1 Synchronous and Asynchronous Modes

The transmit and receive sections for each port can operate either synchronously or asynchronously, as determined by the Synchronous Mode (SYN) bit in the SAPCRC or BBPCRC. In asynchronous mode, there are separate, independent signals and pins for the transmit clock, receive clock, transmit frame sync (TFS) and receive frame sync (RFS). The synchronous mode has a common transmit and receive clock and a common transmit and receive frame syncs. Pin assignments for these signals are listed in Table 14-1 on page 14-3.

14.3.2 Frame Configuration

Each port can be configured for one time slot per frame (normal mode) or multiple time slots per frame (network mode). Each of these modes is periodic. A non-periodic on-demand mode is also provided. The mode is determined by Operation Mode (MOD) bit in the SAPCRC or BBPCRC and the Frame Rate Divider Control (DC[4:0]) bits in the SAPCRA or BBPCRA, as shown in Table 14-3

Table 14-3. Frame Configuration

MOD	DC[4:0] Value	Mode	DC[4:0] Meaning
0	0–31	Normal	(Word transfer rate) – 1
1	1–31	Network	(Number of time slots) – 1
1	0	On-Demand	—

14.3.2.1 Normal Mode

Normal mode is typically used to transfer data to or from a single device. There can be multiple (up to 32) “time slots” per frame, according to the DC[4:0] bits, but data is transferred and received only in the first time slot. Thus, in normal mode, DC[4:0] effectively determine the word transfer rate.

14.3.2.2 Network Mode

Network mode is typically used in TDM systems employing multiple devices. Two to 32 time slots can be selected with the DC[4:0] bits, and data is transferred and received in each time slot.

14.3.2.3 On-Demand Mode

On-demand mode is selected by adjusting the MOD bit for network mode and clearing DC[4:0]. In this mode, frame sync is not periodic but is generated only when data is available to transmit. The TFS must be internal (output), and the RFS must be external (input). Therefore, either synchronous or asynchronous mode can be used in simplex operation, but full-duplex operation requires asynchronous mode. On-demand mode is useful for interfacing to a codec that requires a continuous clock.

14.3.3 Frame Sync

The frame sync frequency for each port is

$$\text{Frame sync frequency} = \frac{\text{bit clock frequency}}{\text{WL} \times (\text{DC} + 1)}$$

where bit clock = the transmit or receive bit clock frequency derived in Section 14.2.2 on page 14-4

WL = Binary value of the word length (8, 12, or 16) as specified by the WL[1:0]) bits in SAPCRA or BBPCRA.

DC = Binary value of the DC[4:0] bits in SAPCRA or BBPCRA.

The following RFS and TFS parameters can be adjusted by bits in SAPCRC or BBPCRC:

- Duration—The sync signals can be either one bit long or one word long by adjusting the Frame Sync Length (FSL[1:0]) bits. In asynchronous mode, the sync signals can be the same or different lengths.
- Direction—The signals can be outputs or inputs according to SCD[2:1]
- Timing—Word-length frame syncs can be asserted at the start of a frame or on the last bit of the previous frame by adjusting the Frame Sync Relative timing (FSR) bit.
- Polarity—The sync signals can be active-high or active-low based on the Frame Sync Polarity (FSP) bit.

14.3.4 Serial I/O Flags

In synchronous mode, the SC0x and SC1x pins are available as Serial I/O Flags. Flag I/O is typically used in codec systems to select among multiple devices for addressing. Flag values can change state for each transmitted or received word. The DSP56652 provides double-buffered control and status bits for the flags to keep them synchronized with the transmit and receive registers. Each flag can be configured as an input or output according to the corresponding SCD bit in the SAPCRC or BBPCRC.

If a flag pin is configured as an input, its state is reflected in the Input Flag (IF0 or IF1) bit in the port Status Register (SAPSR or BBPSR). The pin is latched during reception of the first received bit after an RFS, and the corresponding IF bit is set when the contents of the port's receive shift register are transferred to the SAPRX or BBPRX. Latching the flag input pin allows the signal to change state without affecting the flag state until the first bit of the next received word.

When configured as an output, the flag pin reflects the state of the Output Flag (OF0 or OF1) bit in Control Register B (SAPCRB or BBPCRB). When one of these bits is changed, the value is latched the next time the contents of SAPTX or BBPTX are transferred to the port's transmit shift register. The corresponding flag pin changes state at the start of the following frame (normal mode) or time slot (network mode), and remains stable until the first bit of the following word is transmitted. Use the following sequence for setting output flags when transmitting data:

1. Wait for the TDE bit to be set, indicating the TXB register is empty.
2. Write the OF0 and OF1 bits flags.
3. Write the transmit data to the TXB register.

For each port, the two flags operate independently but can be used together for multiple serial device selection. They can be used unencoded to select one or two codecs, or can be decoded externally to select up to four codecs.

14.3.5 TDM Interrupts

In network mode, interrupts can be generated at the end of the last slot in a transmit or receive frame. The interrupts are enabled by the Receive Last Slot Interrupt (RLIE) and Transmit Last Slot Interrupt (TLIE) bits in the SAPCRB or BBPCRB.

The other four TDM interrupts—Receive, Receive Error, Transmit, and Transmit Error—can occur in any TDM mode. These interrupts are described in Section 14.4.

14.4 Data Transmission and Reception

Each port provides configuration options for data transmission and reception, as well as data format.

14.4.1 Data Transmission

The transmission sequence varies somewhat between normal, network, and on-demand modes.

14.4.1.1 Normal Mode Transmission

The following steps illustrate a typical transmission sequence in normal mode:

1. Write the first transmit data word to the port’s Transmit Register (SAPTX or BBPTX). This clears the Transmit Data Register Empty (TDE) bit in the SAPSR or BBPSR.
2. Set the Transmit Enable (TE) bit in the SAPCRB or BBPCRB.
3. At the next TFS, the Transmit Register data is copied to the Transmit Shift Register, the transmitter is enabled, and the TDE bit is set. The Transmit Register retains the current data until it is written again. If the Transmit Interrupt Enable (TIE) bit in the SAPCRB or BBPCRB is set, an interrupt is generated. At this point, a new value is normally written to the Transmit Register, clearing TDE.
4. Data is shifted out from the shift register to the STDx pin, clocked by the transmit bit clock.
5. The cycle repeats from step 3.

If the TDE bit is set when step 3 occurs, indicating that new data has not been written to the Transmit Register, the Transmit Underflow Error (TUE) bit in the SAPSR or BBPSR is set. If the Transmit Error Interrupt Enable (TEIE) bit in the SAPCRB or BBPCRB is set, an interrupt is generated. The previously sent data, which has remained in the Transmit Register, is again copied to the shift register and transmitted out.

Note: If the TE bit is cleared during a transmission, the SAP or BBP completes the transmission of the current data in the transmit shift register before disabling the transmitter. TE should not be cleared until the TDE bit is set, indicating that the current data has been transferred from the transmit register to the transmit shift register. When the transmitter is disabled, the STDx pin is tri-stated, and any data present in the SAPTX or BBPTX is not transmitted. Data can be written to a Transmit Register when the TE bit is cleared, but is not copied to the shift register until the TE bit is set.

14.4.1.2 Network Mode Transmission

The following steps illustrate a typical transmission sequence in network mode:

1. Write the Transmit Register with the first transmit data word. If no data is to be sent for the first time slot, write to the Time Slot register (SAPTSR or BBPTSR) instead to avoid an underrun error. The content written to the Time Slot Register is irrelevant and ignored.
2. Set the TE bit.
3. If the Transmit Register has been written, the data is copied to the transmit shift register at the next TFS for the first time slot in a frame. For other time slots, the copy takes place at the beginning of the next time slot. The Transmit Register retains the current data until it is written again.
4. The TDE bit is set. If the TIE bit is set, an interrupt is generated. At this point, the Transmit Register or Time Slot Register is written, depending on the following circumstances:
   f. If data is to be transmitted in the next time slot, that data is written to the Transmit Register.
   g. If the next time slot is idle but subsequent time slots are to be used, the Time Slot Register is written to avoid a transmit underrun error.

   Either of these writes clears TDE.
5. If the shift register contains data, the data is shifted out to the STDx pin, clocked by the transmit bit clock. If the shift register is empty (data was written to the Time Slot Register rather than the Transmit Register), the STDx pin is tri-stated for that time slot.
6. If data is to be sent for any subsequent time slots in the frame, or if this is the last time slot in the frame, the cycle repeats from step 3.
7. If no further data is to be sent in this frame, the first time slot of the next frame can be set up by writing either the Transmit Register (with data for the first time slot) or the Time Slot Register. After transmission of the last data word is completed, the TE bit can be toggled (cleared and then reset) . This action disables the transmitter (after the last bit has been shifted out of the transmit shift register) and the STDx pin remains in the high-impedance state until the beginning of the next frame. At the next frame sync, the next frame begins at step 3.

At step 3, if neither the Transmit Register nor the Time Slot Register have been written since step 3 of the previous cycle, the TUE bit is set and an interrupt is generated if enabled as described in Normal mode.

In addition to interrupts for receive and transmit, special network mode interrupts are provided to indicate the last slot.

14.4.1.3 On-Demand Mode

A typical transmission sequence in on-demand mode is as follows:

1. Set the TE bit in the SAPCRB or BBPCRB.
2. Write transmit data to the port’s Transmit Register.
3. The Transmit Register data is copied to the Transmit Shift Register. The Transmit Register retains the current data until it is written again.
4. The TDE bit is set, and an interrupt is generated if the TIE bit in is set.
5. Data is immediately shifted out from the shift register to the STDx pin, clocked by the transmit bit clock.
6. The cycle repeats from step 2, but not at any particular time. If the Transmit Register is written before the current time slot has expired, step 5 will not occur (and the Transmit Register will not accept another word) until the current time slot expires.

Although the SAP transmitter is double-buffered, only one word can be written to the Transmit Register, even when the transmit shift register is empty. Transmit underruns are impossible for on-demand transmission and are disabled.

14.4.2 Data Reception

Data reception is enabled by setting the Receive Enable (RE) pin in SAPCRB or BBPCRB, which allows or inhibits transfer from the shift register to the Receive Register. Data is received on the SRDA or SRDB pin, clocked into the receive shift register by the receive transmit clock. When the number of bits received equals the expected word length (as selected by the WL bits in SAPCRA or BBPCRA), the shift register contents are transferred to the Receive Register (SAPRX or BBPRX), and the Receive Data Register Full (RDF) bit in the SAPSR or BBPSR is set. If the Receive Interrupt Enable (RIE) bit in SAPCRB or BBPCRB is set, an interrupt is generated. Reading the receive register clears the RDF bit. If the received word is the first word in a frame, the Receive Frame Sync (RFS) bit in the SAPSR or BBPSR is set.

If RDF is set when the shift register is full, indicating that the previous received word has not been read, the Receive Overrun Error (ROE) bit in the SAPSR or BBPSR is set, and an interrupt is generated if the Receive Error Interrupt Enable (REIE) bit in the SAPCRB or BBPCRB has been set. The newer data is lost.

14.4.3 Data Formats

Data words can be 8, 12, or 16 bits long. Word length is determined by the WL[1:0] bits in the SAPCRA or BBPCRA.

The shift registers in the SAP and BBP are bidirectional to accommodate data formats that specify MSB first (such as those used by codecs) and LSB first (such as those used by AES-EBU digital audio). Selection of MSB or LSB first is determined by the SHFD bit in the SAPCRC or BBPCRC.

14.5 Software Reset

Either port can be reset without disturbing the rest of the system by clearing the PC[5:0] bits in the Port Control Register (SAPPCR or BBPPCR). This action stops all serial activity and resets the status bits; the contents of SAPCRA, SAPCRB, and SAPCRC are not affected. The port remains in reset while all pins are programmed as GPIO, and becomes active (i.e., functions as the SAP or BBP) only if at least one of the pins is programmed as a SAP or BBP pin.

Note: To ensure proper operation of the interface, the DSP program must reset the SAP or BBP before changing any of its control registers except for the SAPCRB or BBPCRB.

14.6 General-Purpose Timer (SAP Only)

The SAP provides a general-purpose timer that can be used for debugging. The timer is enabled by the TCE bit in the SAPCRB. The following two registers control timer operation:

- The SAP Timer Counter (SAPCNT) is a counter that is decremented by a clock running at a frequency of (DSP_CLK ÷ 2048). When it decrements to zero, a timer counter rollover interrupt is issued.
- The SAP Timer Modulus Register (SAPMR) contains a modulus value that is loaded into the SAPCNT register when TCE is set and each time the counter rolls over.

Note: Although this timer is technically not involved in SAP operation, the SAP must be enabled by setting the PEN bit *and* at least one of the PC[5:0] bits in the SAP Port Control Register (SAPPCR) to enable the timer.

14.7 Frame Counters (BBP Only)

The BBP provides two counters that can be used to count transmit and receive frames.

Setting the TCE bit in BBPCRB enables the transmit frame counter and loads it with the value in the BBP Transmit Counter Modulus Register (BBPTMR). The counter is decremented by transmit frame sync. When the counter rolls over, it is again loaded with BBPTMR, and an interrupt is generated if the TCIE bit in BBPCRB is set.

Setting the RCE bit in BBPCRB enables the receive frame counter and loads it with the value in the BBP Receive Counter Modulus Register (BBPRMR). The counter is decremented by receive frame sync. When the counter rolls over, it is again loaded with BBPRMR, and an interrupt is generated if the RCIE bit in BBPCRB is set.

Note: Although these counters are technically not involved in BBP operation, the BBP must be enabled by setting the PEN bit *and* at least one of the PC[5:0] bits in the BBP Port Control Register (BBPPCR) to enable the counters.

14.8 Interrupts

Table 14-4 presents a summary of the possible interrupts the DSP can generate for each port, ordered from highest to lowest priority (assuming they are all assigned the same interrupt priority level), along with their corresponding status and interrupt enable bits, if any.

Table 14-4. SAP and BBP Interrupts

Interrupt	SAPCRB Interrupt Enable Bit	SAPSR Status Bit
SAP Receive Data with Overrun Error	REIE	ROE
SAP Receive Data	RIE	RDF
SAP Receive Last Slot	RLIE	—
SAP Transmit Data with Underrun Error	TEIE	TUE
SAP Transmit Last Slot	TLIE	—
SAP Transmit Data	TIE	TDE
SAP Timer Counter Rollover	TCIE	—
	BBPCRB Interrupt Enable Bit	**BBPSR Status Bit**
BBP Receive Data with Overrun Error	REIE	ROE
BBP Receive Data	RIE	RDF
BBP Receive Last Slot	RLIE	—
BBP Receive Frame Counter	RCIE	—
BBP Transmit Data with Underrun Error	TEIE	TUE
BBP Transmit Last Slot	TLIE	—
BBP Transmit Data	TIE	TDE
BBP Transmit Frame Counter	TCIE	—

14.9 SAP and BBP Control Registers

Table 14-5 and Table 14-6 are summaries of the SAP and BBP control registers respectively, including the acronym, bit names, and address of each register.

Table 14-5. Serial Audio Port Register Summary

SAPCNT	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X:$FFB4	LV[15:0]															
SAPMR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X:$FFB5	LV[15:0]															
SAPCRA	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X:$FFB6	PSR	WL[1:0]		DC[4:0]					PM[7:0]							
SAPCRB	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X:$FFB7	REIE	TEIE	RLIE	TLIE	RIE	TIE	RE	TE						TCE	OF[1:0]	
SAPCRC	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X:$FFB8	FSP	FSR	FSL[1:0]					BRM	SHFD	CKP	SCKD	SCD[2:0]			MOD	SYN
SAPSR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X:$FFB9									RDF	TDE	ROE	TUE	RFS	TFS	IF[1:0]	
SAPRX	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X:$FFBA	Receive Word															
SAPTSR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X:$FFBB	(Dummy)															
SAPTX	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X:$FFBC	Transmit Word															
SAPPDR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X:$FFBD											PD[5:0]					
SAPDDR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X:$FFBE											PDC[5:0]					
SAPPCR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X:$FFBF									PEN		PC[5:0]					

Table 14-6. Baseband Port Register Summary

BBPRMR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X:$FFA4	LV[15:0]															
BBPTMR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X:$FFA5	LV[15:0]															
BBPCRA	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X:$FFA6	PSR	WL[1:0]		DC[4:0]					PM[7:0]							
BBPCRB	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X:$FFA7	REIE	TEIE	RLIE	TLIE	RIE	TIE	RE	TE	RCIE	TCIE	RCE	TCE			OF[1:0]	
BBPCRC	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X:$FFA8	FSP	FSR	FSL[1:0]						SHFD	CKP	SCKD	SCD[2:0]			MOD	SYN
BBPSR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X:$FFA9									RDF	TDE	ROE	TUE	RFS	TFS	IF[1:0]	
BBPRX	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X:$FFAA	Receive Word															
BBPTSR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X:$FFAB	(Dummy)															
BBPTX	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X:$FFAC	Transmit Word															
BBPPDR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X:$FFAD											PD[5:0]					
BBPDDR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X:$FFAE											PDC[5:0]					
BBPPCR	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X:$FFAF									PEN		PC[5:0]					

14.9.1 SAP and BBP Control Registers

SAPCNT SAP Timer Counter **X:$FFB4**

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
	SAP Timer Count															
RESET	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

This read-only register holds the value of the SAP timer.

BBPRMR BBP Receive Counter Modulus Register **X:$FFA4**

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
	BBP Receive Counter Load Value															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register contains the value that is loaded in the BBP receive frame counter register when the counter is enabled and when the counter rolls over.

SAPMR SAP Timer Modulus Register **X:$FFB5**

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
	SAP Timer Load Value															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register contains the value that is loaded in the SAPCNT register when the timer is enabled and when the timer rolls over.

BBPTMR BBP Transmit Counter Modulus Register **X:$FFA5**

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
	BBP Transmit Counter Load Value															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register contains the value that is loaded in the BBP transmit frame counter register when the counter is enabled and when the counter rolls over.

SAPCRA SAP Control Register A **X:$FFB6**
BBPCRA BBP Control Register A **X:$FFA6**

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
	PSR	WL[1:0]		DC[4:0]					PM[7:0]							
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 14-7. SAP/BBP CRA Description

Name	Description	Settings
PSR Bit 15	**Bit Clock Prescaler—**Setting this bit bypasses the divide-by-eight prescaler to the bit rate generator. **Note:** The combination of PSR = 1 and PM[7:0] = $00 is reserved and may cause synchronization problems if used.	0 = Prescale applied (default). 1 = No prescale.
WL[1:0] Bits 14–13	**Word Length—**These bits select the word length for transmitted and received data.	00 = 8 bits per word (default). 01 = 12 bits per word. 10 = 16 bits per word. 11 = Reserved.
DC[4:0] Bits 12–8	**Frame Rate Divider Control—**These bits in conjunction with the MOD bit in the SAPCRC or BBPCRC configure the transmit and receive frames. Refer to Table 14-3 on page 14-6. In network mode, value of this field plus one equals the number of slots per frame. In normal mode, the value of this field is the number of dummy “time slots”, effectively determining the word transfer rate.	
PM[7:0] Bits 7–0	**Prescale Modulus—**These bits along with the PSR bit determine the bit clock frequency. Refer to Section 14.2.2 on page 14-4. **Note:** The combination of PSR = 1 and PM[7:0] = $00 is reserved and may cause synchronization problems if used.	

SAPCRB SAP Control Register B **X:$FFB7**

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
	REIE	TEIE	RLIE	TLIE	RIE	TIE	RE	TE						TCE	OF1	OF0
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

BBPCRB BBP Control Register B **X:$FFA7**

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
	REIE	TEIE	RLIE	TLIE	RIE	TIE	RE	TE	RCIE	TCIE	RCE	TCE			OF1	OF0
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Note: In addition to setting the interrupt enable bits in the SAPCRB or BBPCRB, the SAPPL or BBPPL field respectively in the IPRP must be written with a non-zero value to generate the respective interrupts (see page 7-14).

Table 14-8. SAP/BBP CRB Description

Name	Description	Settings
REIE Bit 15	**Receive Error Interrupt Enable—**Setting this bit enables an interrupt when a receive overflow error occurs.	0 = Interrupt disabled (default). 1 = Interrupt enabled.
TEIE Bit 14	**Transmit Error Interrupt Enable—**Setting this bit enables an interrupt when a transmit underflow error occurs.	0 = Interrupt disabled (default). 1 = Interrupt enabled.
RLIE Bit 13	**Receive Last Slot Interrupt Enable—**In network mode, setting this bit enables an interrupt at the end of the last receive time slot in a frame. RLIE has no effect in other modes.	0 = Interrupt disabled (default). 1 = Interrupt enabled.
TLIE Bit 12	**Transmit Last Slot Interrupt Enable—**In network mode, setting this bit enables an interrupt at the beginning of the last transmit time slot in a frame. TLIE has no effect in other modes.	0 = Interrupt disabled (default). 1 = Interrupt enabled.
RIE Bit 11	**Receive Interrupt Enable—**Setting this bit enables an interrupt when the receive register receives the last bit of a word and transfers the contents to the Receive Register.	0 = Interrupt disabled (default). 1 = Interrupt enabled.
TIE Bit 10	**Transmit Interrupt Enable—**Setting this bit enables an interrupt when the contents of the Transmit Register are transferred to the transmit shift register.	0 = Interrupt disabled (default). 1 = Interrupt enabled.
RE Bit 9	**Receive Enable—**Enables the SAP or BBP receiver by allowing data transfer from the receive shift register to the Receive Register.	0 = Receiver disabled (default). 1 = Receiver enabled.

Table 14-8. SAP/BBP CRB Description

Name	Description	Settings
TE Bit 8	**Transmit Enable—**Enables the SAP or BBP transmitter by allowing data transfer from the Transmit Register to the transmit shift register. **Note:** The TE bit does not affect the generation of frame sync or output flags.	0 = Transmitter disabled (default). 1 = Transmitter enabled.
RCIE (BBP). Bit 7	**BBP Receive Counter Interrupt Enable—**Setting this bit enables an interrupt when the BBP receive counter rolls over.	0 = Interrupt disabled (default). 1 = Interrupt enabled.
TCIE (BBP). Bit 6	**BBP Transmit Counter Interrupt Enable—**Setting this bit enables an interrupt when the BBP transmit counter rolls over.	0 = Interrupt disabled (default). 1 = Interrupt enabled.
RCE (BBP). Bit 5	**BBP Receive Counter Enable—**Enables the BBP receive frame sync counter.	0 = Counter disabled (default). 1 = Counter enabled.
TCE (BBP). Bit 4	**BBP Transmit Counter Enable—**Enables the BBP transmit frame sync counter.	0 = Counter disabled (default). 1 = Counter enabled.
TCE (SAP). Bit 2	**SAP Timer Count Enable—**Enables the SAP general-purpose timer.	0 = Timer disabled (default). 1 = Timer enabled.
OF1 Bit 1	**Output Flag 1—**In synchronous mode (SYN bit in the SAPCRC or BBPCRC is set), this bit drives serial output flag 1 on the SC1x pin if it is configured as an output (SCD1 bit in the SAPCRC or BBPCRC is set).	
OF0 Bit 0	**Output Flag 0—**In synchronous mode (SYN bit in the SAPCRC or BBPCRC is set), this bit drives serial output flag 0 on the SC0x pin if it is configured as an output (SCD0 bit in the SAPCRC or BBPCRC is set).	

SAPCRC SAP Control Register C **X:$FFB8**
BBPCRC BBP Control Register C **X:$FFA8**

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
	FSP	FSR	FSL[1:0]		▨	▨	▨	BRM*	SHFD	CKP	SCKD	SCD2	SCD1	SCD0	MOD	SYN
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 14-9. SAP/BBP CRC Description

Name	Description	Settings
FSP Bit 15	**Frame Sync Polarity**—Determines if frame sync is active-high or active-low.	0 = Active-high (default). 1 = Active-low.
FSR Bit 14	**Frame Sync Relative Timing**—Determines if frame sync is asserted at the last bit or the previous frame or the first bit of the current frame. This bit is effective for word-length frame sync only.	0 = First bit of current frame (default). 1 = Last bit of previous frame.
FSL[1:0] Bits 13–12	**Frame Sync Length**—These bits determine the duration (word-length or bit-length) for both transmit and receive frame sync.	00 = TFS and RFS are word-length (default). 01 = TFS is bit-length; RFS is word-length. 10 = TFS and RFS are bit-length. 11 = TFS is word-length; RFS is bit-length.
BRM (SAP). Reserved **(BBP).** Bit 8	**Bit Rate Multiplier (SAP only)**—Selects either DSP_CLK or BRM_CLK as the input to the bit clock prescaler.	0 = DSP_CLK (default). 1 = BRM_CLK.
SHFD Bit 7	**Shift Direction**—Determines if data is sent and received MSB first or LSB first.	0 = MSB first (default). 1 = LSB first.
CKP Bit 6	**Clock Polarity**—Determines the bit clock edge on which frame sync is asserted and data is shifted.	0 = Transmit—bit clock rising edge Receive—bit clock falling edge (default). 1 = Transmit—bit clock falling edge Receive—bit clock rising edge.
SCKD Bit 5	**Serial Clock Pin Direction**—Determines if the SCKx pin is an output or an input.	0 = Input (default). 1 = Output.
SCD2 Bit 4	**Serial Control Pin 2 Direction**—Determines if the SC2x pin is an output or an input.	0 = Input (default). 1 = Output.
SCD1 Bit 3	**Serial Control Pin 1 Direction**—Determines if the SC1x pin is an output or an input.	0 = Input (default). 1 = Output.
SCD0 Bit 2	**Serial Control Pin 0 Direction**—Determines if the SC0x pin is an output or an input.	0 = Input (default). 1 = Output.
MOD Bit 1	**Normal/Network Mode Select**	0 = Normal mode (default). 1 = Network mode.
SYN Bit 0	**Synchronous/Asynchronous Select**	0 = Asynchronous mode (default). 1 = Synchronous mode .

SAPSR SAP Status Register A **X:$FFB9**
BBPSR BBP Status Register A **X:$FFA9**

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
									RDF	TDE	ROE	TUE	RFS	TFS	IF1	IF0
RESET	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0

The SAPSR and BBPSR are 8-bit, read-only registers.

Table 14-10. SAP/BBP Status Register Description

Name	Description	Settings
RDF Bit 7	**Receive Data Register Full—**Set when the contents of the receive shift register are transferred to the Receive Register. Cleared by reading the Receive Register.	0 = No new data received (default). 1 = New data in Receive Register.
TDE Bit 6	**Transmit Data Register Empty—**Set when the contents of the Transmit Register are transferred to the transmit shift register. Cleared by a write to the Receive Register or the Time Slot Register.	0 = Last transmit word has not yet been copied to transmit shift register. 1 = Last transmit word has been copied to transmit shift register (default).
ROE Bit 5	**Receiver Overrun Error—**Set when the last bit of a word is shifted into the receive shift register and RDF is set, meaning that the previous received word has not been read. Cleared by reading the Status Register, then the Receive Register.	0 = No receive error (default). 1 = Receiver overrun error has occurred.
TUE Bit 4	**Transmitter Underrun Error—**Set when the transmit shift register is empty and a time slot occurs, meaning that the Transmit Register has not been written since the last transmission. Cleared by reading the Status Register, then writing the Transmit Register or the Time Slot Register.	0 = No transmit error (default). 1 = Transmitter underrun error has occurred.
RFS Bit 3	**Receive Frame Sync—**This bit reflects the status of the receive frame sync signal, whether generated internally or received externally. In normal mode, RFS is always set. In network mode, RFS is set only during the first time slot of the receive frame, and remains set for the duration of the word reception, regardless of the state of the FSL bit in the SAPCRC or BBPCRC.	
TFS Bit 2	**Transmit Frame Sync—**This bit reflects the status of the transmit frame sync signal, whether generated internally or received externally. In normal mode, TFS is always set. In network mode, TFS is set only during the first time slot of the transmit frame, and remains set for the duration of the word transmission, regardless of the state of the FSL bit in the SAPCRC or BBPCRC.	
IF1 Bit 1	**Input Flag 1—**In synchronous mode, this bit reflects the state of Input Flag 1, which is driven on the SC1x pin.	
IF0 Bit 0	**Input Flag 0—**In synchronous mode, this bit reflects the state of Input Flag 0, which is driven on the SC0x pin.	

SAPRX SAP Receive Data Register X:$FFBA
BBPRX BBP Receive Data Register X:$FFAA

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
	Receive Word															
RESET	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

This read-only register accepts data from the receive shift register after the last bit of a receive word is shifted in. If the word length is less than 16 bits, the data is shifted into the most significant bits.

SAPTSR SAP Time Slot Register X:$FFBB
BBPTSR BBP Time Slot Register X:$FFAB

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
	(Dummy)															
RESET	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

This dummy write-only register is written to avoid a transmit underrun error for a time slot for which no data is to be transmitted.

SAPTX SAP Transmit Data Register X:$FFBC
BBPTX BBP Transmit Data Register X:$FFAC

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
	Transmit Word															
RESET	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

This write-only register loads its data into the transmit shift register. If the word length is less than 16 bits, writes to this register should occupy the most significant bits.

14.9.2 GPIO Registers

SAPPDR SAP Port Data Register **X:$FFBD**

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
											SAPPD5 (STDA)	SAPPD4 (SRDA)	SAPPD3 (SCKA)	SAPPD2 (SC2A)	SAPPD1 (SC1A)	SAPPD0 (SC0A)
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

BBPPDR BBP Port Data Register **X:$FFAD**

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
5											BBPPD5 (STDB)	BBPPD4 (SRDB)	BBPPD3 (SCKB)	BBPPD2 (SC2B)	BBPPD1 (SC1B)	BBPPD0 (SC0B)
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 14-11. SAP/BBP PDR Description

Name	Description	Settings
SAPPD[5:0] **BBPPD[5:0]** Bits 5–0	**Port Data**—Each of these bits contains data for the corresponding pin if it is configured as GPIO. A write to one of these registers is stored in an internal latch, and driven on any port pin configured as an output. Reads of these registers return the value sensed on input pins and the latched data driven on outputs	

SAPDDR SAP Data Direction Register **X:$FFBE**

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
											SAPDD5 (STDA)	SAPDD4 (SRDA)	SAPDD3 (SCKA)	SAPDD2 (SC2A)	SAPDD1 (SC1A)	SAPDD0 (SC0A)
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

BBPDDR BBP Data Direction Register **X:$FFAE**

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
5											BBPDD5 (STDB)	BBPDD4 (SRDB)	BBPDD3 (SCKB)	BBPDD2 (SC2B)	BBPDD1 (SC1B)	BBPDD0 (SC0B)
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 14-12. SAP/BBP DDR Description

Name	Description	Settings
SAPDD[5:0] **BBPDD[5:0]** Bits 5–0	**Data Direction**—Each of these bits determines the data direction of the associated pin if it is configured as GPIO.	0 = Input (default). 1 = Output.

SAPPCR SAP Port Control Register **X:$FFBF**

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
									PEN		SAPPC5 (STDA)	SAPPC4 (SRDA)	SAPPC3 (SCKA)	SAPPC2 (SC2A)	SAPPC1 (SC1A)	SAPPC0 (SC0A)
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

BBPPCR BBP Port Control Register **X:$FFAF**

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
5									PEN		BBPPC5 (STDB)	BBPPC4 (SRDB)	BBPPC3 (SCKB)	BBPPC2 (SC2B)	BBPPC1 (SC1B)	BBPPC0 (SC0B)
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 14-13. SAP/BBP PCR Description

Name	Description	Settings
PEN Bit 7	**Port Enable—**Setting this bit enables all SAP or BBP pins to function as defined by all other register settings. When PEN is cleared, all port pins are tri-stated.	0 = All pins tri-stated. 1 = All pins function as configured.
SAPPC[5:0] **BBPPC[5:0]** Bits 5–0	**Pin Configuration—**Each bit determines whether its associated pin functions as a peripheral (SAP or BBP) or GPIO.	0 = GPIO (default). 1 = SAP or BBP.

Chapter 15
JTAG Port

The DSP56652 includes two Joint Test Action Group (JTAG) Test Access Port (TAP) controllers that are compatible with the *IEEE 1149.1 Standard Test Access Port and Boundary Scan Architecture*. The block diagram of these two TAPs is shown in Figure 15-1.

All JTAG testing functions in the DSP56652 are performed by the DSP TAP controller. The JTAG-specific functions required by IEEE 1149.1 are not included in the MCU TAP controller, which is bypassed in JTAG compliance mode. The MCU TAP controller is only active in MCU OnCE emulation mode, in which the two controllers are enabled and connected serially. MCU OnCE operation is described in the *MMC2001 Reference Manual*. DSP OnCE operation is described in the *DSP56600 Family Manual*.

This chapter describes aspects of the JTAG implementation that are specific to the DSP56600 core, including items which the IEEE standard requires to be defined and additional information specific to the DSP core implementation. For internal details and applications of the standard, refer to the IEEE 1149.1 document.

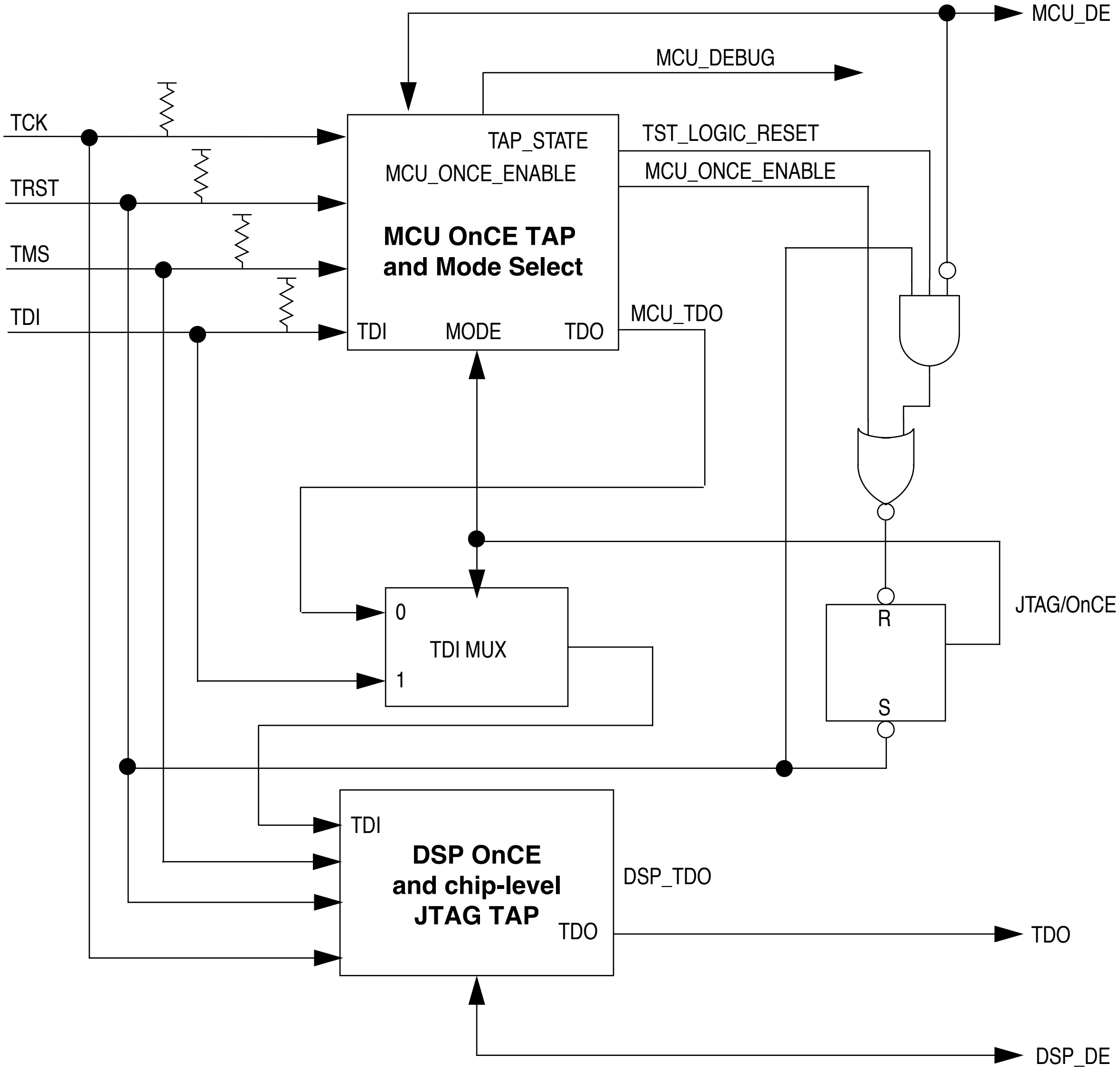


Figure 15-1. DSP56652 JTAG Block Diagram

15.1 DSP56600 Core JTAG Operation

The DSP56600 core JTAG TAP includes six signal pins, a 16-state controller, an instruction register, and three test data registers. The test logic employs a static logic design and is independent of the device system logic. A block diagram of the DSP56600 core implementation of JTAG is shown in Figure 15-2.

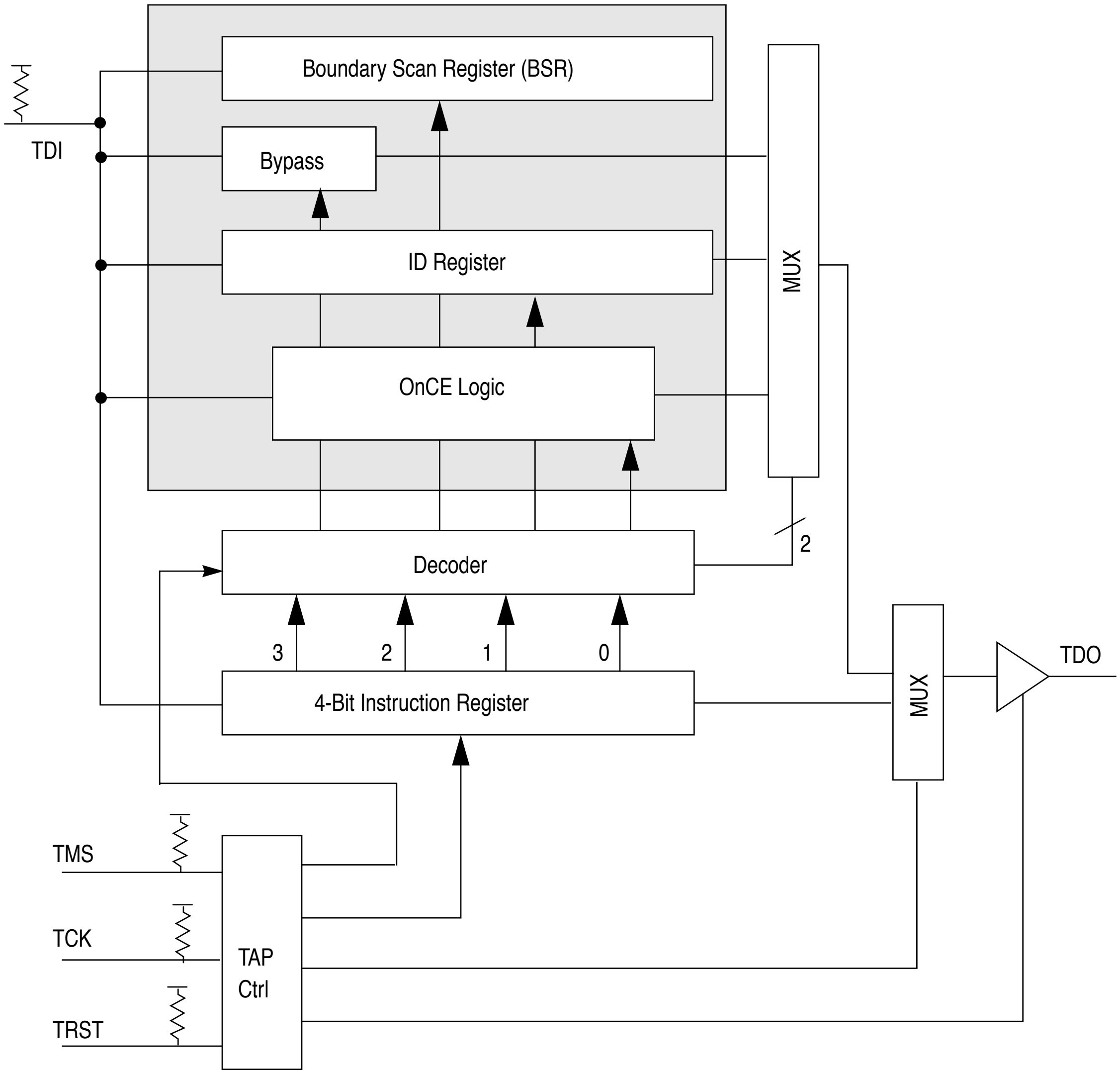


Figure 15-2. DSP56600 Core JTAG Block Diagram

15.1.1 JTAG Pins

As described in the IEEE 1149.1 document, the JTAG port requires a minimum of four pins to support the TDI, TDO, TCK, and TMS signals. The DSP TAP also provides $\overline{\text{TRST}}$ and $\overline{\text{DSP\_DE}}$ pins. The pin functions are described in Table 15-1.

Table 15-1. DSP JTAG Pins

Pin	Description
TCK	Test Clock—An input that is used to synchronize the test logic. The TCK pin has an internal pullup resistor.
TMS	Test Mode Select—An input that is used to sequence the test controller's state machine. TMS is sampled on the rising edge of TCK and includes an internal pullup resistor.
TDI	Test Data Input—Serial test instruction and data are received through the Test Data Input (TDI) pin. TDI is sampled on the rising edge of TCK and includes an internal pullup resistor.
TDO	Test Data Output—The serial output for test instructions and data. TDO is three-stateable and is actively driven in the Shift-IR and Shift-DR controller states. TDO changes on the falling edge of TCK.
$\overline{\text{TRST}}$	Test Reset—An input that is used to asynchronously initialize the test controller and select the JTAG-compliant mode of operation. The $\overline{\text{TRST}}$ pin has an internal pullup resistor.
$\overline{\text{DSP\_DE}}$	Test Data Output—A bidirectional pin used as an input to asynchronously initialize the test controller.

15.1.2 DSP TAP Controller

The DSP TAP controller is responsible for interpreting the sequence of logical values on the TMS signal. It is a synchronous state machine that controls the operation of the JTAG logic. A diagram of the TAP controller state machine is shown in Figure 15-3. The value shown adjacent to each arc represents the value of the TMS signal sampled on the rising edge of TCK signal. For a description of the TAP controller states, refer to the *IEEE 1149.1 Standard Test Access Port and Boundary Scan Architecture*.

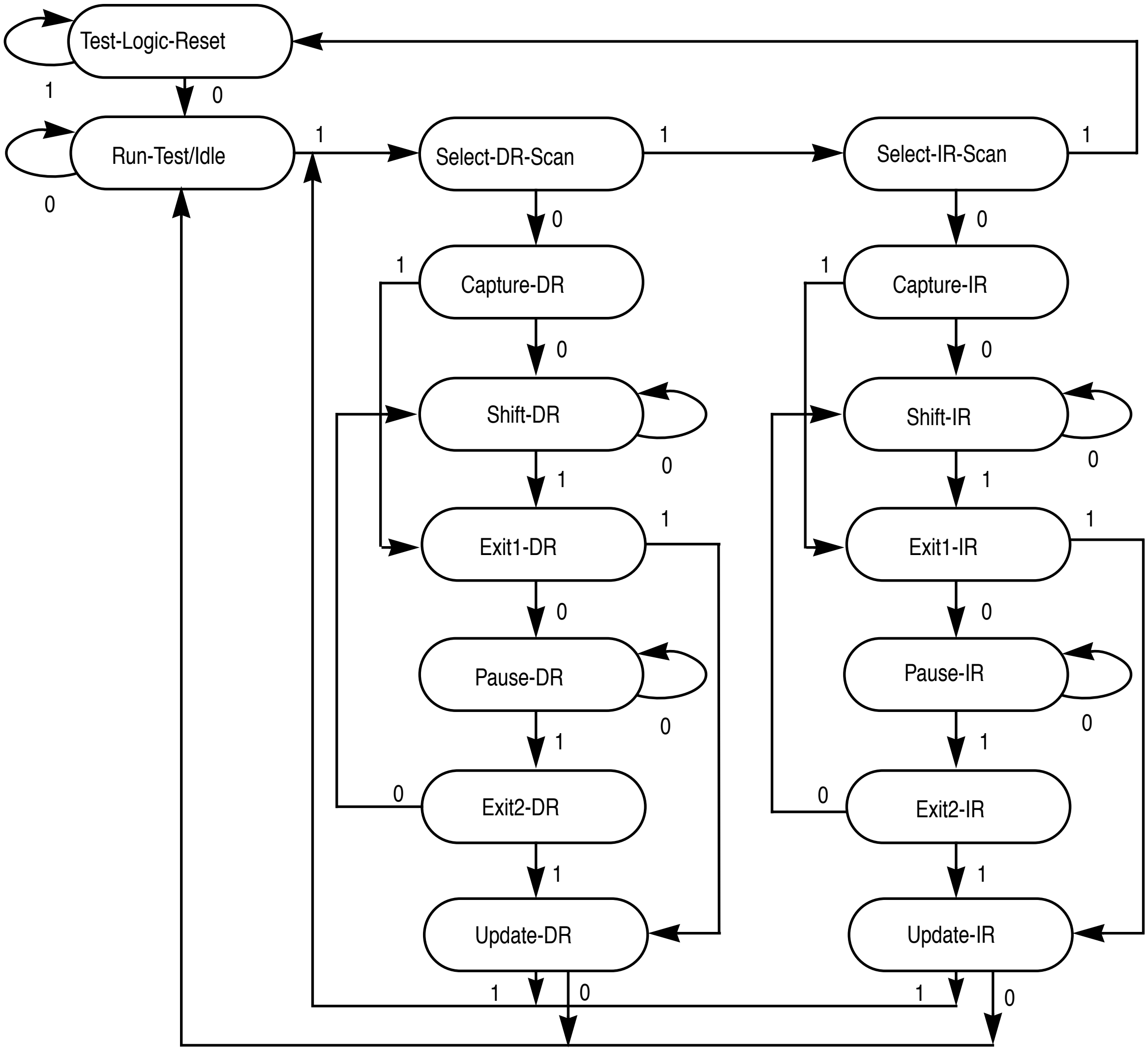


Figure 15-3. TAP Controller State Machine

15.1.3 Instruction Register

The DSP JTAG implementation includes a 4-bit instruction register without parity consisting of a shift register with four parallel outputs. Figure 15-4 shows the Instruction Register configuration.

B3	B2	B1	B0

Figure 15-4. JTAG Instruction Register

15.1.3.1 Instruction Register Operation

Data is transferred from the shift register to the parallel outputs during the Update-IR controller state. The four bits are used to decode the eight unique instructions shown in Table 15-2.

Table 15-2. JTAG Instructions

Code				Instruction
B3	B2	B1	B0	
0	0	0	0	**EXTEST**—Perform external testing for circuit-board electrical continuity using boundary scan operations.
0	0	0	1	**SAMPLE/PRELOAD**—Sample the DSP56652 device system pins during operation and transparently shift out the result in the BSR. Preload values to output pins prior to invoking the EXTEST instruction.
0	0	1	0	**IDCODE**—Query identification information (manufacturer, part number and version) from an DSP core-based device.
0	0	1	1	**ENABLE_MCU_ONCE**—Provide a means of accessing the MCU OnCE controller and circuits to control a target system.
0	1	0	0	**HI-Z**—Disable the output drive to pins during circuit-board testing.
0	1	0	1	**CLAMP**—Force test data onto the outputs of the device while replacing its boundary-scan register in the serial data path with a single bit register.
0	1	1	0	**ENABLE_DSP_ONCE**—Provide a means of accessing the DSP OnCE controller and circuits to control a target system.
0	1	1	1	**DSP_DEBUG_REQUEST**—Provide a means of entering the DSP into Debug Mode of operation.
1000–1110				Reserved for future use. Decoded as BYPASS.
1	1	1	1	**BYPASS**—Bypass the DSP56652 chip for a given circuit-board test by effectively reducing the BSR to a single cell.

In the Test-Logic-Reset controller state the Instruction Register is reset to b0010, which is equivalent to the IDCODE instruction.

In the Capture-IR controller state, the two least significant bits of the instruction shift register are parallel-loaded with b01 as required by the standard. The two most significant bits are loaded with the values of the core status bits OS1 and OS0 from the OnCE controller.

15.1.3.2 Instruction Descriptions

The DSP core JTAG implementation includes the three mandatory public instructions (EXTEST, SAMPLE/PRELOAD, and BYPASS), and also supports the optional CLAMP instruction defined by IEEE 1149.1. The public instruction HIGHZ provides the capability for disabling all device output drivers. The public instruction ENABLE_DSP_ONCE enables the JTAG port to communicate with the DSP OnCE circuitry. The public instruction DSP_DEBUG_REQUEST enables the JTAG port to force the DSP core into Debug mode.

15.1.3.2.1 EXTEST (B[3:0]=0000)

The external test (EXTEST) instruction selects the BSR and gives the test logic control of the I/O pins. EXTEST also asserts internal reset for the DSP56652 core system logic to force a predictable internal state while performing external boundary scan operations.

By using the TAP controller, the Instruction Register is capable of:

- Scanning user-defined values into the output buffers
- Capturing values presented to input pins
- Controlling the direction of bidirectional pins
- Controlling the output drive of tri-stateable output pins

For more details on the function and use of EXTEST, refer to *IEEE 1149.1*.

15.1.3.2.2 SAMPLE/PRELOAD (B[3:0]=0001)

The SAMPLE/PRELOAD instruction selects the BSR and the system logic controls the I/O pins. The SAMPLE/PRELOAD instruction provides two separate functions. First, it provides a means to obtain a snapshot of system data and control signals. 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.

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.

The second function of SAMPLE/PRELOAD is to initialize the BSR output cells prior to selection of EXTEST. This initialization ensures that known data appears on the outputs when entering the EXTEST instruction.

15.1.3.2.3 IDCODE (B[3:0]=0010)

The IDCODE instruction selects the ID register, and the system logic controls the I/O pins. This instruction is provided as a public instruction to allow the manufacturer, part number and version of a component to be determined through the TAP. The ID register is described in Section 15.2.3 on page 15-10.

Since the bypass register loads a logic 0 at the start of a scan cycle, whereas the ID register loads a logic 1 into its least significant bit, examination of the first bit of data shifted out of a component during a test data scan sequence immediately following exit from Test-Logic-Reset controller state shows whether such a register 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 by the IEEE 1149.1 standard.

15.1.3.2.4 ENABLE_MCU_ONCE (B[3:0]=0011)

The ENABLE_MCU_ONCE instruction is not included in the IEEE 1149.1 standard. It is provided as a public instruction to allow the user to perform system debug functions. When the ENABLE_MCU_ONCE instruction is decoded the DSP JTAG controller is set to the BYPASS mode. This is the only function performed by the DSP controller. OnCE operation in the MCU is controlled by the MCU’s OnCE TAP.

15.1.3.2.5 HIGHZ (B[3:0]=0100)

When the HIGHZ instruction is invoked, all output drivers, including the two-state drivers, are turned off (i.e., put in the high impedance state), and the Bypass Register is selected. The HIGHZ instruction also asserts internal reset for the DSP56652 core system logic to force a predictable internal state while performing external boundary scan operations. In this mode, all internal pullup resistors on all the pins (except the TMS, TDI, and $\overline{\text{TRST}}$ pins) are disabled.

15.1.3.2.6 CLAMP (B[3:0]=0101)

The CLAMP instruction selects the 1-bit Bypass Register as the serial path between TDI and TDO while allowing signals driven from the component pins to be determined from the BSR. During testing of ICs on PCB, it may be necessary to place static guarding values on signals that control operation of logic not involved in the test. If the EXTEST instruction were used for this purpose, the boundary-scan register would be selected and the required guarding signals would be loaded as part of the complete serial data stream shifted in, both at the start of the test and each time a new test pattern is entered. The CLAMP instruction results in substantially faster testing than the EXTEST instruction because it allows guarding values to be applied using the BSR of the appropriate ICs while selecting their bypass registers. 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. The

CLAMP instruction also asserts internal reset for the DSP56652 core system logic to force a predictable internal state while performing external boundary scan operations.

15.1.3.2.7 ENABLE_DSP_ONCE (B[3:0]=0110)

The ENABLE_DSP_ONCE instruction is not included in the IEEE 1149.1 standard. It is provided as a public instruction to allow the user to perform system debug functions. When the ENABLE_DSP_ONCE instruction is decoded, the TDI and TDO pins are connected directly to the DSP OnCE registers. The particular DSP OnCE register connected between TDI and TDO at a given time is selected by the DSP OnCE controller depending on the DSP OnCE instruction being currently executed. All communication with the DSP OnCE controller is done through the Select-DR-Scan path of the JTAG TAP controller.

15.1.3.2.8 DSP_DEBUG_REQUEST (B[3:0]=0111)

The DSP_DEBUG_REQUEST instruction is not included in the IEEE 1149.1 standard. It is provided as a public instruction to allow the user to generate a debug request signal to the DSP core. When the DSP_DEBUG_REQUEST instruction is decoded, the TDI and TDO pins are connected to the Instruction Registers. When the TAP is in the Capture-IR state, the OnCE status bits are captured in the Instruction shift register. Thus, the external JTAG controller must continue to shift in the DSP_DEBUG_REQUEST instruction while polling the status bits that are shifted out until Debug mode is entered and acknowledged by the combination 11 on OS[1:0]. After the acknowledgment of Debug mode is received, the external JTAG controller must issue the ENABLE_DSP_ONCE instruction to allow the user to perform system debug functions.

15.1.3.2.9 BYPASS (B[3:0]=1xxx)

The BYPASS instruction selects the single-bit Bypass Register and restores control of the I/O pins to system logic. This creates a shift-register path from TDI through the Bypass Register to TDO, circumventing the BSR. This instruction is used to enhance test efficiency when a component other than the DSP56652 becomes the device under test.

15.2 Test Registers

The DSP core implementation includes three test registers—a Boundary Scan Register (BSR), a 1-bit Bypass Register, and a 32-bit Identification Register (ID).

15.2.1 Boundary Scan Register (BSR)

The Boundary Scan Register (BSR) in the DSP core JTAG implementation contains bits for all device signal and clock pins and associated control signals. In addition, the BSR contains a data direction control bit for each bidirectional pin. Boundary scan bit definitions are provided in the Boundary Scan Description Language (BSDL) listing in Appendix C.

Note: As a compliance enable pin, $\overline{\text{MCU\_DE}}$ is not included in the BSR definition.

15.2.2 Bypass Register

The Bypass Register allows the serial data path to circumvent the DSP BSR. It is activated by the HIGHZ, CLAMP, and BYPASS instructions. When the Bypass Register is selected, the shift-register stage is set to a logic zero on the rising edge of TCK in the Capture-DR controller state. Therefore, the first bit to be shifted out after selecting the Bypass Register is always a logic zero. A drawing of the Bypass Register is shown in Figure 15-5.

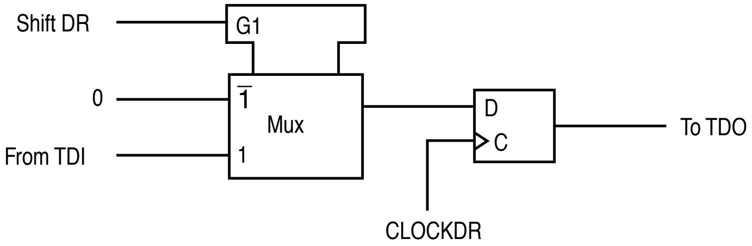


Figure 15-5. JTAG Bypass Register

15.2.3 Identification Register

The ID register contains the manufacturer, part number and version of the DSP56652. It is read by invoking the IDCODE command. It can be used to determine the manufacturer of a component on a board when multiple sourcing is used. Conforming to the IEEE 1149.1 standard in this way allows a system diagnostic controller to determine the type of component in each location through blind interrogation. This information is also available for factory process monitoring and for failure mode analysis of assembled boards.

Motorola's Manufacturer Identity is b00000001110. The Customer Part Number consists of two parts: Motorola Design Center Number (bits 27:22) and a sequence number (bits

21:12). The sequence number is divided into two parts: Core Number (bits 21:17) and Chip Derivative Number (bits 16:12). Motorola Semiconductor Israel (MSIL) Design Center Number is b000110 and DSP Core Number is b00010. Figure 15-6 shows the ID register configuration.

31 28	27 22	21 17	16 12	11 1	0
Version Number	Customer Part Number			Manufacturer Identity Number	
	Design Center Number	Core Number	Derivative Number		
0 0 0 0	0 0 0 1 1 0	0 0 0 1 0	0 0 0 0 0	0 0 0 0 0 0 0 1 1 1 0	1

Figure 15-6. JTAG ID Register

15.3 DSP56652 JTAG Port Restrictions

This section describes operation restrictions regarding the DSP56652 JTAG port in normal, test, and low-power modes.

15.3.1 Normal Operation

- **JTAG transparency—**To ensure that the JTAG test logic is kept transparent to the system logic in normal operation, the JTAG TAP controller must be initialized and kept in the Test-Logic-Reset controller state. The controller can be forced into Test-Logic-Reset by asserting $\overline{\text{TRST}}$ externally at power-up reset. The controller will remain in this state as long as TMS is not driven low.
- **Connecting the TCK pin—**The TCK pin does not have an on-board pullup resistor, and should be tied to a logic high or low during normal operation.

15.3.2 Test Modes

- **Signal contention in circuit-board testing—**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 DSP56652 output drivers are enabled into actively driven networks.
- **Executing the EXTEST instruction—**The EXTEST instruction can be performed only after power-up or regular hardware reset while EXTAL is provided. Then during the execution of EXTEST, EXTAL can remain inactive.

15.3.3 STOP Mode

- **Entering STOP—**The TAP controller must be in the Test-Logic-Reset state to enter and remain in STOP mode.

- **Minimizing power consumption**—The TMS and TDI pins include on-chip pullup resistors. In STOP mode, these two pins should remain either unconnected or connected to V_{CC} to achieve minimal power consumption. Also, the TCK input is not blocked in STOP mode and should be externally connected to V_{CC} or ground.

15.4 MCU TAP Controller

The MCU contains a TAP controller to provide MCU OnCE support. It is bypassed in JTAG-compliant mode. The MCU OnCE operating mode can be selected in two ways:

- Assertion of the $\overline{\text{MCU\_DE}}$ line while the TAP controllers are in the Test-Logic-Reset state and the $\overline{\text{TRST}}$ input is deasserted.
- Shifting the ENABLE_MCU_ONCE command into the DSP TAP controller.

In the MCU OnCE mode, the MCU and DSP TAP controllers are serially linked. The TDI pin drives the MCU TAP controller TDI input, and the MCU TAP controller TDO output drives the DSP TAP controller TDI input. The combined Instruction Registers (IRs) and Data Registers (DR’s) of the two controllers are connected, effectively allowing both to be read or written from a single serial input stream. The TMS, $\overline{\text{TRST}}$, and TCK inputs of the two controllers are connected together, forcing an identical sequence of state transitions to occur within the individual TAP controllers.

To return from the MCU OnCE configuration to JTAG-compliant mode, deassert the $\overline{\text{MCU\_DE}}$ signal and assert $\overline{\text{TRST}}$.

15.4.1 Entering MCU OnCE Mode via JTAG Control

Table 15-3 shows the TMS sequencing for entering MCU OnCE mode from JTAG-compliant mode by shifting the ENABLE_MCU_ONCE command into the DSP TAP controller.

Table 15-3. Entering MCU OnCE Mode

Step	TMS	JTAG State	OnCE	Note
a	1	Test-Logic-Reset	Idle	
b	0	Run-Test/Idle	Idle	
c	1	Select-DR-Scan	Idle	
d	1	Select-IR-Scan	Idle	
e	0	Capture-IR	Idle	Capture DSP core status bits
f	0	Shift-IR	Idle	The 4 bits of the JTAG ENABLE_MCU_ONCE instruction (0b0011) are shifted into the DSP instruction register
g	0	Shift-IR	Idle	
h	0	Shift-IR	Idle	
i	0	Shift-IR	Idle	
j	1	Exit1-IR	Idle	At this point, the IR section of the DSP is ready to be loaded. The MCU TAP controller shadow logic is ready to reset the JTAG/$\overline{\text{OnCE}}$ signal.
k	1	Update-IR	OnCE Enabled	MCU OnCE mode is enabled.
l	0	Run-Test/Idle	OnCE Enabled	MCU OnCE mode is enabled.

Note: When the MCU OnCE mode is enabled, the JTAG IR becomes the concatenation of the DSP IR (4 bits) and the MCU IR (8 bits). Subsequent shifts into the JTAG IR should be 12 bits in length.

15.4.2 Release from Debug Mode for DSP and MCU

Table 15-4 shows the TMS sequencing for simultaneously releasing the MCU and DSP from Debug mode, assuming all internal states have been restored to both cores.

Table 15-4. Releasing the MCU and DSP from Debug Modes

Step	TMS	JTAG State	OnCE	Note
a	1	Test-Logic-Reset	Idle	
b	0	Run-Test/Idle	Idle	
c	1	Select-DR-Scan	Idle	
d	1	Select-IR-Scan	Idle	
e	0	Capture-IR	Idle	Capture DSP core status bits
f	0	Shift-IR	Idle	The 4 bits of the JTAG ENABLE_DSP_ONCE instruction (0b0110) are shifted into the combined DSP + MCU instruction register
g	0	Shift-IR	Idle	
h	0	Shift-IR	Idle	
i	0	Shift-IR	Idle	
j	0	Shift-IR	Idle	The remaining 8 bits of the MCU OnCE instruction “read no register selected + go + exit” (0b11101100) are shifted into the combined DSP + MCU IR
k	0	Shift-IR	Idle	
l	0	Shift-IR	Idle	
m	0	Shift-IR	Idle	
n	0	Shift-IR	Idle	
o	0	Shift-IR	Idle	
p	0	Shift-IR	Idle	
q	0	Shift-IR	Idle	
r	1	Exit1-IR	Idle	At this point, both IR sections are ready to be loaded, the MCU with “read no register selected + go + exit”, the DSP with “Enable DSP OnCE”
s	1	Update-IR	Idle	OnCE is enabled for the DSP (already enabled for the MCU)
t	1	Select-DR-Scan	Idle	
u	0	Capture-DR	Idle	
v	0	Shift-DR	Idle	The 8 bits of the DSP OnCE command “read no register selected + go + exit” (0b11111111) are shifted in
....				
v	0	Shift-DR	Idle	
w	0	Shift-DR	Idle	A single bit of bypass data corresponding to the MCU portion of the combined DR is shifted in
x	1	Exit1-DR	Idle	
y	1	Update-DR	Idle	Following this update, both OnCE control blocks release their respective cores
z	0	Run-Test/Idle	Idle	
....				
z	0	Run-Test/Idle	Idle	

Appendix A
DSP56652 DSP Bootloader

The DSP56652 DSP Bootloader is a small program residing in the DSP program ROM that is executed when the DSP exits the reset state. The purpose of the bootloader is to provide MCU-DSP communication to enable the MCU to download a DSP program to the DSP program RAM through the MCU-DSP Interface (MDI). This appendix describes the various protocols available in the bootloader to communicate with the DSP56652 and how a protocol is selected. It also provides a listing of the bootloader program.

A.1 Boot Modes

The user can select one of the following three protocols, or modes, to use to download code for the DSP:

- **Mode A: Normal MDI boot mode** implements a protocol incorporating MDI shared memory and messaging registers that enables the user to upload and download data to or from any address in program, X, or Y memory, test the 512-byte program RAM, and start the DSP from any address in program memory.
- **Mode B: MDI shared memory boot mode** allows only downloading to program RAM using only the MDI shared memory to transfer data. The DSP program must start from program RAM address $0000. Some synchronization between the MCU and DSP is required.
- **Mode C: MDI messaging unit boot mode** allows only downloading to program RAM using only the MDI messaging unit registers to transfer data. The DSP program must start from program RAM address $0000. No MCU-DSP synchronization is required.

The bootloader reads the SAP STDA pin and the BBP STDB pin (configured as GP inputs at reset) to determine the boot mode, as shown Table A-1 on page A-2. The user must supply pull-up and/or pull-down resisters to STDA and STDB to ensure that the DSP enters the desired mode.

Table A-1. DSP56652 Boot Modes

STDA	STDB	Boot Mode
1	1	Mode A: Normal MDI boot mode.
0	1	Mode B: MDI shared memory boot mode.
1	0	Mode C: MDI messaging unit boot mode.
0	0	Reserved for Motorola test modes

A.2 Mode A: Normal MDI Boot

The normal boot mode uses MDI communication between the DSP and MCU to implement the following functions:

- Download to the DSP program, X, or Y RAM.
- Upload from the DSP program, X, or Y memories (RAM or ROM).
- Run diagnostic tests on the DSP 0.5K program RAM.
- Start the DSP at a given program address (jump to a given address)

After entering the normal boot mode, the DSP waits until a message has arrived from the MCU. When it receives a message, the DSP performs the necessary actions and in most cases returns an acknowledgment message to the MCU. The DSP remains in the normal boot mode, waiting for and executing MCU messages, until the MCU requests the DSP to exit the boot mode and start the user's application.

A.2.1 Short and Long Messages

The normal boot mode uses both the MDI messaging unit registers and the MDI shared memory for message transfers. Shorter messages are conveyed in one or both messaging unit registers[1]. For longer messages (such as downloading a program to the DSP), MDI_R0 is used to point to the rest of the message in the MDI shared memory.

The format for short messages is shown in Figure A-1 on page A-3. The most significant bit of MDI_R0 is used to indicate whether the message is a short message (S=1) or a long message (S=0). The eight least significant bits of MDI_R0 hold the message opcode. Bits

---

[1] For simplicity, the messaging unit registers (MTR0, MTR1, MRR0, and MRR1 for the MCU transmit and receive registers, respectively; DTR0, DTR1, DRR0, and DRR1 for the DSP transmit and receive registers, respectively) are referred to as MDI_R0 and MDI_R1.

8–13 can contain message information if needed. If the short message uses the MDI_R1 register as well, the DW bit (bit 14) in MDI_R0 should be set.

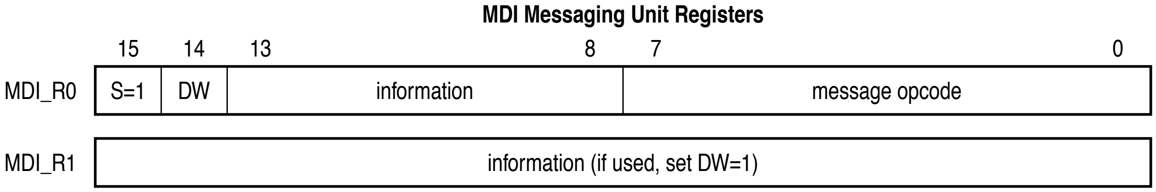


Figure A-1. Short Message Format

The format for long messages is shown in Figure A-2. The long message is indicated by clearing the S bit in MDI_R0.

The ten least significant bits of MDI_R0 indicate an offset address into the MDI shared memory. Note that this field is 10 bits wide so that it can point to an offset anywhere in the 1-Kword MDI shared memory space. The first entry in the MDI shared memory at the indicated offset location is the message opcode. This is followed by as many information words as necessary.

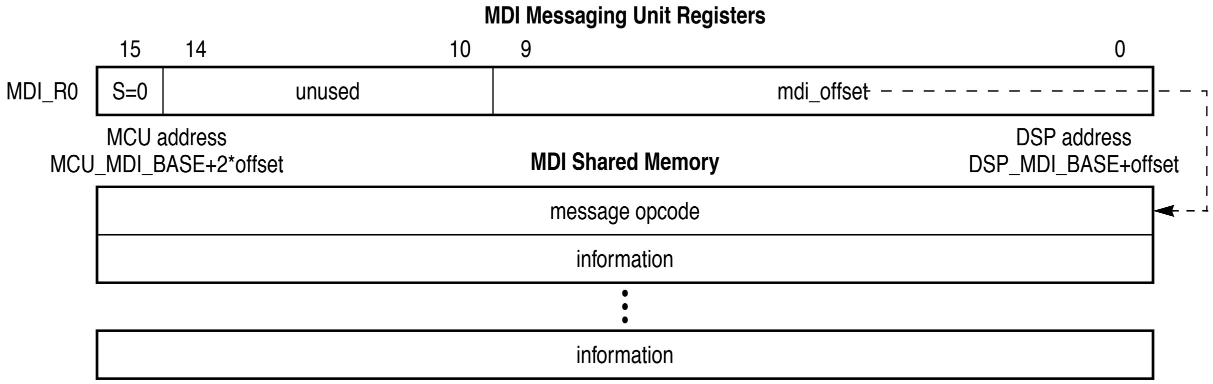


Figure A-2. Long Message Format

A.2.2 Message Descriptions

Table A-2 summarizes the messages that the bootloader supports. Initially, the bootloader is in an idle loop awaiting a message from the MCU. When it receives a message, the DSP processes and executes the command, then sends an acknowledgment message back to the MCU. The only exception to this procedure is the start_application.request message, for which there is no acknowledgment message. If the DSP receives a message it does not recognize, it returns a special invalid opcode response.

Table A-2. Message Summary

Message From MCU to DSP	Message Opcode Number	Long or Short	Acknowledgment Message From DSP to MCU	Message Opcode Number	Long or Short
memory_write.request	1	long	memory_write.response	1	short
memory_read.request	2	long	memory_read.response	2	long
memory_check.request	3	long	memory_check.response	3	long
start_application.request	4	long	(none)	NA	NA
(invalid message)	other	either	invalid_opcode.response	4	short

The following sections describe the structure of each message.

A.2.2.1 memory_write.request

`memory_write.request` is a long message from the MCU to the DSP used to write to the DSP program or data RAM. The structure of this message is shown in Figure A-3. The first entry in MDI memory is the message opcode. The second entry contains the number of words to write to DSP memory. The third entry contains two fields, XYP and source address offset. The XYP field, which occupies the upper two bits of the entry, determines which memory space to access, as shown in Table A-3. The source address offset occupies the lowest ten bits of the third entry and indicates the location in the MDI memory space of the data to be written to the DSP. The last entry of the message contains the DSP destination address to which the data is to be written. In most cases, the source address offset points to the word following the destination address, i.e., source_address_offset = mdi_offset + 4. However, the protocol allows for the data to be located anywhere in the MDI shared memory space.

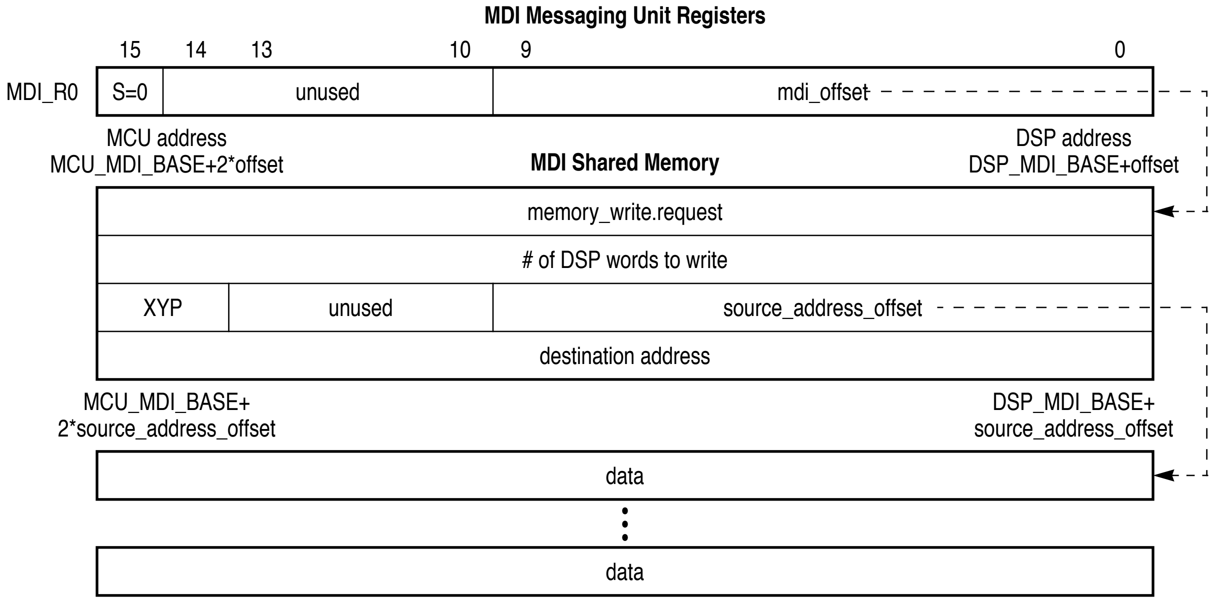


Figure A-3. Format of memory_write.request Message

Table A-3. XYP Field

XYP	DSP Memory Space
00	X
01	Y
10	P

A.2.2.2 memory_write.response

`memory_write.response` is a short message from the DSP to the MCU in response to a memory_write.request message. The format of this message is shown in Figure A-4. Note that the MDI_R1 register is not used. A RET field of 0 indicates a successful memory_write.request; if the RET field is 1, the memory_write.request failed. Thus, since memory_write.response opcode is $1, the MCU should expect the DSP to respond to a successful memory write with MDI_R0 = $8001.

MDI Messaging Unit Registers

	15	14	13 8	7 0
MDI_R0	S=1	DW=0	RET (0=success, 1=failure)	memory_write.response
MDI_R1	not used			

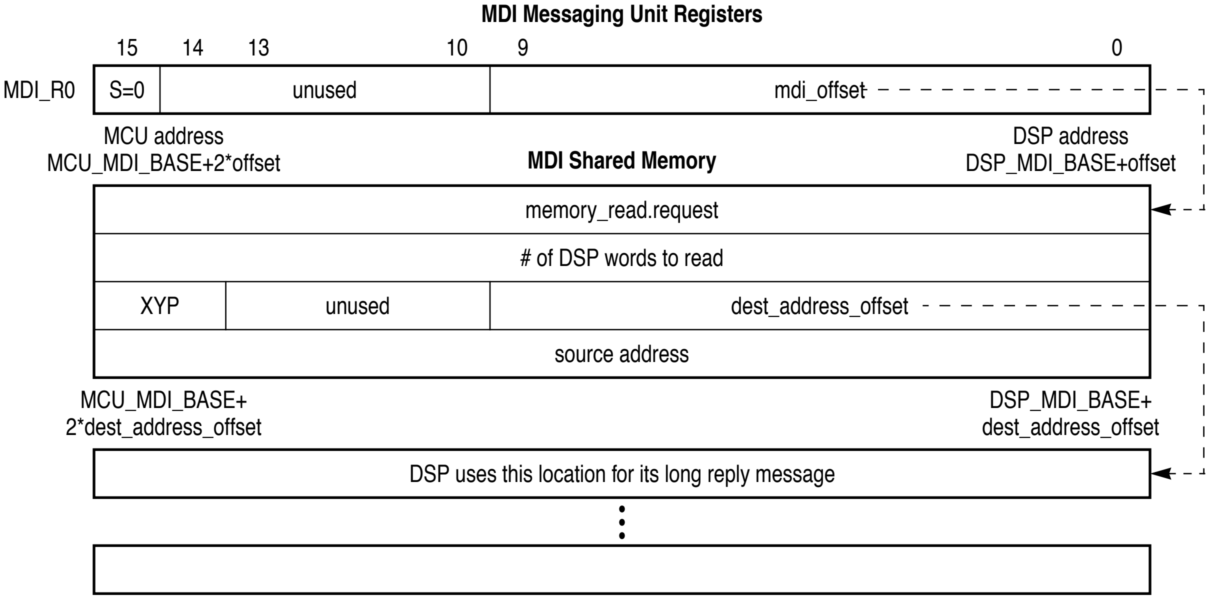
Figure A-4. Format of message_write.response Message

A.2.2.3 memory_read.request

`memory_read.request` is a long message from the MCU to the DSP requesting an upload of data from the program, X, or Y data space. The format of this message is shown in Figure A-5. The next entry in MDI memory following the `memory_read.request` opcode is the number of DSP words to read. The third entry contains two fields, XYP and destination address offset. The XYP field determines which memory space of the read, as shown in Figure A-3 on page A-5. The destination address offset contains the location in MDI shared memory at which the DSP stores the data it reads. The last entry, source address, indicates the address in DSP program, X, or Y memory space of the data to be read.

The choice of destination address offset is arbitrary, but care should be taken to ensure that the DSP does not overwrite any of the words in the original message.

Figure A-5. Format of memory_read.request Message



A.2.2.4 memory_read.response

`memory_read.response` is a long message from the DSP to the MCU in response to a `memory_read.request` message. The format of this long message is shown in Figure A-6. Note that this long message is located in MDI shared memory at the location defined by the destination address field of the `memory_read.request` message.

The entry following the `memory_write.request` opcode in MDI memory is the return code—$0000 indicates success, and $0001 indicates failure. Failure can only result from the invalid value of 11b to the XYP field in the `memory_read.request` message. If the return code indicates a failure, the DSP does not write the remaining entries in the message. The third entry in the `memory_read.response` message is the number of DSP words read. The fourth entry contains two fields. The upper two bits indicate the memory space accessed according to Table A-3 on page A-5. The lower ten bits indicate the location in MDI shared memory where the DSP has stored the read data. In all cases, the bootloader defines the destination address offset to point to the word following the source address. Therefore, dest_address_offset = mdi_offset + 5. The last entry, source address, indicates the DSP program, X, or Y space address from which the data has been read.

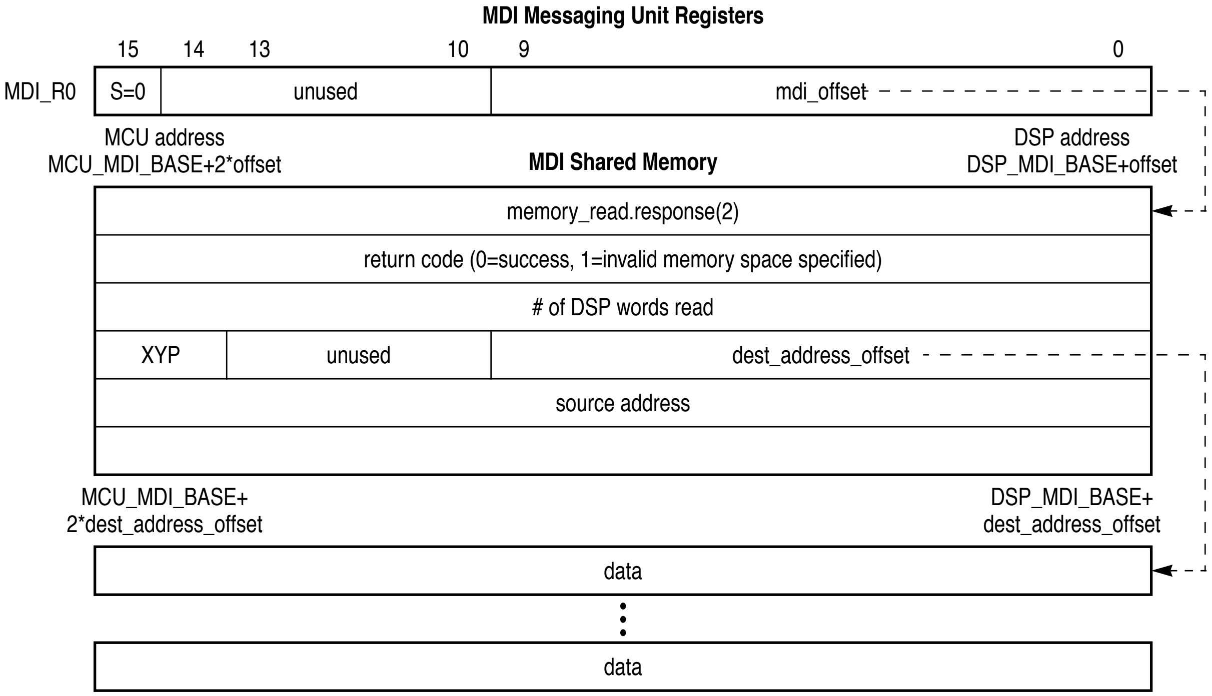


Figure A-6. Format of memory_read.response Message

A.2.2.5 memory_check.request

`memory_check.request` is a long message from the MCU to the DSP requesting a test of the DSP 0.5k program RAM.

Note: Although this protocol supports provisions to test all of the memory spaces, the bootloader only implements testing of the 0.5k program RAM space.

The format of this message is shown in Figure A-7. The entry following the opcode in shared memory contains two fields. The upper three bits specify the RAM space to be tested (always “100” for the bootloader), and the lower ten bits specify the MDI address the at which DSP stores its long reply message.

Normally, the return address offset points to the next word, so that return_address_offset = mdi_offset + 2. However, the protocol allows for the long reply message to be located anywhere in MDI memory.

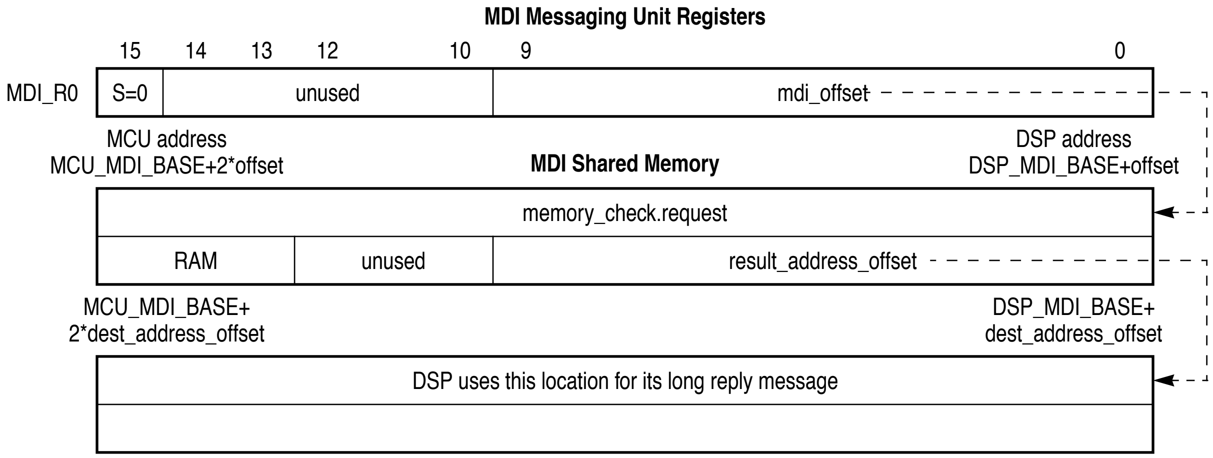


Figure A-7. Format of memory_check.request Message

A.2.2.6 memory_check.response

`memory_check.response` is a long message from the DSP to the MCU in response to a `memory_check.request` message. The format of this message is shown in Figure A-8. Note that this message resides in MDI shared memory location specified in the result_address_offset field of the `memory_check.request` message.

The entry in MDI shared memory following the `memory_check.response` opcode is the return code—$0000 indicates success, and $0001 indicates failure. The following entry is the failure address if the memory check has failed, and zero if the check is successful.

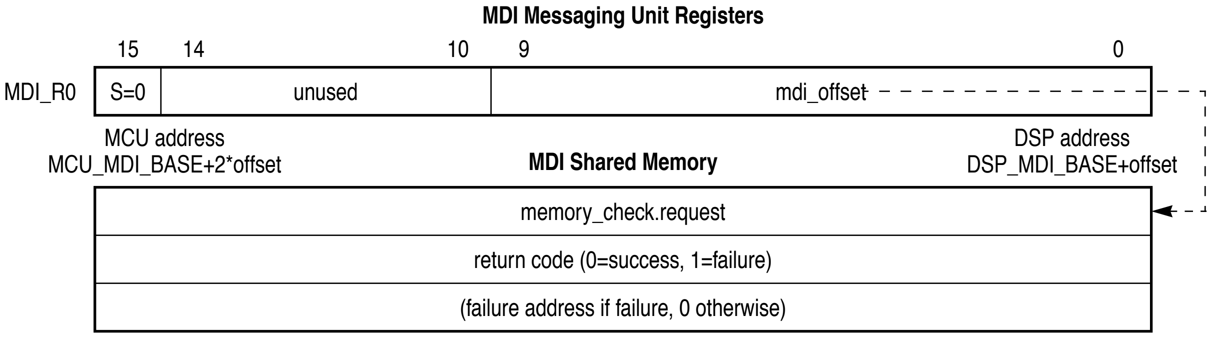


Figure A-8. Format of memory_check.request Message

A.2.2.7 start_application.request

`start_application.request` is a long message from the MCU to the DSP requesting the DSP to leave the boot mode and begin executing the user program The format of this message is shown in Figure A-9. The entry following the `start_application.request` opcode in MDI shared memory is the starting address of the user program in program memory. When the DSP receives this message, it jumps to the specified program address location and begins executing code at that location. The DSP does not generate a response to this message.

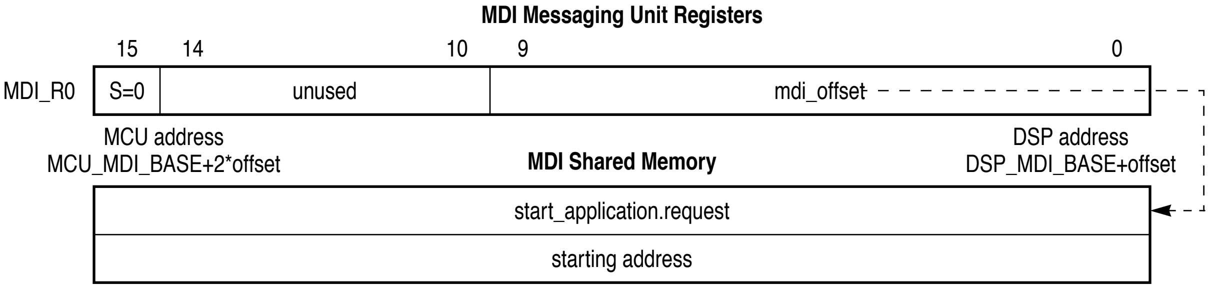


Figure A-9. Format of start_application.request Message

A.2.2.8 invalid_opcode.response

`invalid_opcode.response` is a short message from the DSP to the MCU in response to any unrecognized message opcode. The format of this message is shown in Figure A-10. The RET field in MDI_R0 is used to indicate the length of the unrecognized message (0 = long, 1 = short). The unrecognized opcode is returned in the MDI_R1 register.

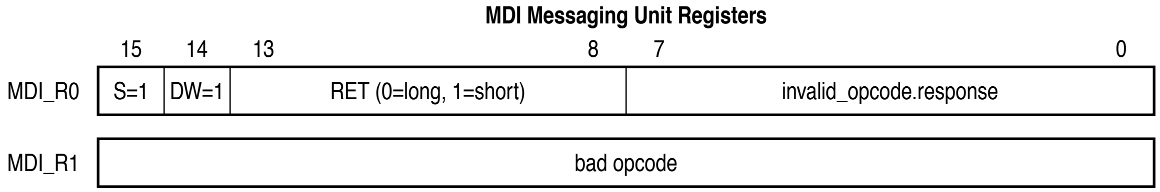


Figure A-10. Format of invalid_opcode.response Message

A.2.3 Comments on Normal Boot Mode Usage

This section describes several items to keep in mind when using the normal boot mode.

1. **Downloads and uploads of DSP program memory require two words** in the MDI shared memory space because DSP program words are 24 bits wide. The most significant portion (upper 8 bits) should always we stored in the lower memory address, followed by the least significant (lower 16 bits) in the next higher memory address, as illustrated in Figure A-11.

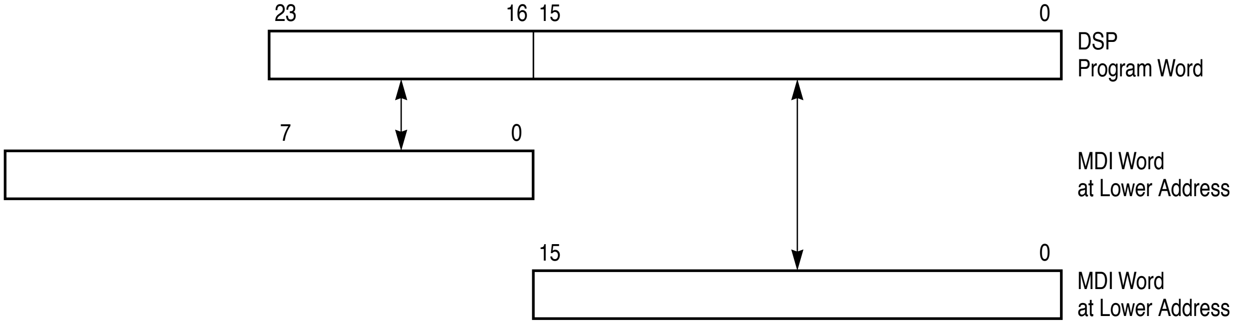


Figure A-11. Mapping of DSP Program Memory words to MDI message words

2. **MDI shared memory size is only 1 Kword.** Data transfers larger than 1 Kword must be split into multiple uploads or downloads.
3. **The DSP does not perform any error checking.** MCU software is responsible for ensuring that addresses are within the MDI memory space.
4. **Writing MDI_R0 should be the final step taken to initiate a message.** This action affects bits in the MDI status register that the DSP bootloader program polls to determines when a new message has been received in MDI_R0.
5. **Ensure that the response to a message does not overwrite that message.** Each MCU message that invokes a long message reply from the DSP defines the offset in MDI shared memory where the DSP stores the response. Care should be taken so that no portion of the reply overwrites any portion of the original message. The DSP may need to access the original message while it is writing its response message.

A.2.4 Example of Program Download and Execution

Example A-1 provides a short outline in pseudo-C code for downloading and starting a program in normal boot mode. In this example, all long messages start at the beginning of MDI shared memory, the DSP program exists in a long array called `dsp_program[]`, the program length is contained in a variable called `program_length`, and the program starting address is `dsp_program_address`.

Example A-1. Normal Boot

```
unsigned short *mdimem = (unsigned short *)MDI_MEM_ADDR;
unsigned short *MTR0 = (unsigned short *)MDI_MTR0;
volatile unsigned short *MRR0 = (unsigned short *)MDI_MRR0;
volatile unsigned short *MSR = (unsigned short *)MDI_MSR;

/* prepare to download to the DSP */
/* write long message info in shared mem */
*mdimem++ = memory_write.request;
*mdimem++ = program_length;
*mdimem++ = (%10<<14) + 4;   /* %10: download to P memory */
                             /* 4: data starts following
                                      this header information */
*mdimem++ = dsp_program_address;

/* write dsp program to MDI most significant part first */
for(i=0; i<program_length; i++)
{
      *mdimem++ = (unsigned short)(dsp_program[i]>>16);
      *mdimem++ = (unsigned short)dsp_program[i];
}

/* initiate this long message by writing to MTR0 register */
*MTR0 = 0;                           /* msb=0 -> long message */
                                     /* lsbs=0 -> offset = 0 */
/* wait for acknowledgement from DSP by polling the MRF0 bit in MSR */
while(MSR&MRF0==0)
      ;
/* read and test the short message memory_write.response*/
if(MRR0 != $8001)
      exit(1);              /* DSP write error */

/* start the DSP application */
/* reset the mdi memory pointer to beginning of mdi */
*mdimem = (unsigned short *)MDI_MEM_ADDR;
/* write the long message header */
*mdimem++ = start_application.request;
*mdimem++ = dsp_program_address;
/* initiate the long message by writing to MTR0 reg */
*MTR0 = 0;                   /* msb=0 -> long message */
                             /* lsbs=0 -> offset = 0 */
```

A.3 Mode B: Shared Memory Boot

The shared memory boot mode can be used if all that is required is to fill the lower 0.5K DSP program RAM and begin execution at DSP program address P:$0000. The MDI memory values are undefined at reset, so this boot mode requires a bit of MCU-DSP synchronization prior downloading the code. The first two 16-bit words in the shared MDI memory space are reserved for synchronization messages. To download DSP code in the boot mode, the MCU must take the following steps:

1. Download up to 511 DSP program words to the MDI memory starting at the third MDI memory location. Note that the most signification portion is stored first.
2. Write synchronization word 1 ($1234) to MDI shared memory location 0.
3. Wait for the DSP to acknowledge this by writing confirmation word 1 ($abcd) to MDI shared memory location 1.
4. Write synchronization word 2 ($5678) to MDI shared memory location 0.
5. Wait for the DSP to acknowledge this by writing confirmation word 2 ($cdef) to MDI shared memory location 1.
6. The DSP should now be reading the program from the MDI memory locations and jump to P:$0000 after the last word has been read.

These steps are demonstrated in the pseudo-C program in Example A-2.

Example A-2. Shared Memory Boot

```
unsigned short *mdimem = (unsigned short *)MDI_MEM_ADDR+2;
volatile unsigned short *mdimem0 = (unsigned short *)MDI_MEM_ADDR;
volatile unsigned short *mdimem1 = (unsigned short *)MDI_MEM_ADDR+1;

/* write 511 dsp program words starting at MDI memory offset 2 */
/*  -- write msb portion first */
for(i=0; i<511; i++)
{
      *mdimem++ = (unsigned short)(dsp_program[i]>>16);
      *mdimem++ = (unsigned short)dsp_program[i];
}

/* write syn message 1 */
*mdimem0 = $1234;

/* wait for confirm message 1 */
while(*mdimem1 != $abcd)
      ;

/* write sync message 2 */
*mdimem0 = $5678;
```

```
/* wait for confirm message 2 */
while(*mdimem1 != $cdef)
      ;
```

A.4 Mode C: Messaging Unit Boot

The messing unit memory boot mode can also be used if all that is required is to fill the lower 0.5k DSP program RAM and begin execution at DSP program address P:$0000.

This mode uses the MDI messaging unit registers so there is no need for additional synchronization logic. In this mode, the MCU should write a maximum of 511 DSP program words, one at a time, to the two messaging unit registers. The most significant portion of each word should be written to MDI_R0 and the least significant portion to MDI_R1. The DSP reads MDI_R0 first, so the MCU should write MDI_R0 first. Also, the MCU should poll the transmit empty bits in the MDI status register to ensure that the DSP has read each register before a new value is written.

Example A-2 is pseudo-C program of a boot using the MDI messaging unit.

Example A-3. Messaging Unit Boot

```
unsigned short *mtr0 = (unsigned short *)MDI_MTR0;
unsigned short *mtr1 = (unsigned short *)MDI_MTR1;
volatile unsigned short *msr = (unsigned short *)MDI_MSR;

/* write 511 dsp program words starting at MDI memory offset 2 */
/*  -- write msb portion first */
for(i=0; i<511; i++)
{
      while(*msr&MSR_MTE0==0)
             ;
      *mtr0 = (unsigned short)(dsp_program[i]>>16);
      while(*msr&MSR_MTE1 == 0)
             ;
      *mtr1 = (unsigned short)dsp_program[i];
}
```

A.5 Bootstrap Program

The following bootstrap source code is programmed into the DSP56652 at the factory. Use this listing to develop external ROM programming for DSP56652 applications.

Note: When compiling source code, the correct X I/O equate and interrupt equate files (specified by ioequ.asm and intequ.asm) must be used. Listings for these files are provided in Appendix B.

```
;---------------------------------------------------------------------------
;
; DSP BOOT LOADER CODE FOR 56652
;
; Boot mode is determined from reading the STDA, STDB pins:
;       STDA  STDB
;       ----  ----
;        1     1     boot mode A, normal boot mode
;        0     1     boot mode B, shared memory boot mode
;        1     0     boot mode C, messaging unit boot mode
;        0     0     reserved for SPS test modes
;
;---------------------------------------------------------------------------


            section     BOOTSTRAP

;
;;;;;;;;;;; BOOT MODE A MESSAGE EQUATES ;;;;;;;;;;;;;;;;;;;;;;;;;
;

; long message header
long_header             equ  $4000

; message opcodes
mem_write               equ  $0001
mem_read                equ  $0002
mem_check               equ  $0003
start_app               equ  $0004
inval_opc               equ  $0004

; long read/write memory codes (bits 14,15)
mem_x                   equ  $0000   ;%00
mem_y                   equ  $4000   ;%01
mem_p                   equ  $8000   ;%10
mem_invalid             equ  $C000   ;%11
; long memory check mem space codes (bits 13,14,15)
pram512                 equ  $8000    ;%100

; response messages
success                 equ  0
fail        equ         1
fail_inv_mem    equ     2
```

```
; short response messages
write_success           equ  (1<<15)+(success<<8)+mem_write
write_fail              equ  (1<<15)+(fail<<8)+mem_write
inval_long_msg          equ  $C000+inval_opc
inval_short_msg         equ  $C100+inval_opc


;
;;;;;;;;;;;; BOOT MODE B EQUATES ;;;;;;;;;;;;;;;;;;;;;;;;;;
;
prot_B_sig_0            equ  $1234
prot_B_sig_1            equ  $5678
prot_B_conf_0           equ  $abcd
prot_B_conf_1           equ  $cdef


;
;;;;;;;;;;;;;;;;;;; DSP I/O REGISTERS ;;;;;;;;;;;;;;;;;;;;;;;;
;


; bus switch
BPMRH       equ         $FFF2; bus switch program memory register high
BPMRL       equ         $FFF3; bus switch program memory register low
BPMRG       equ         $FFF4; bus switch program memory register (24bits)

; MDI
MDI_base    equ         $1C00; base dp ram address
DRR0        equ         $FF8F; dsp receive register 0
DRR1        equ         $FF8E; dsp receive register 1
DTR0        equ         $FF8D; dsp transmit register 0
DTR1        equ         $FF8C; dsp transmit register 1
DSR         equ         $FF8B; dsp status register
DCR         equ         $FF8A; dsp control register
; DSR bits
DF0         equ         0    ; DSR flag 0
DF1         equ         1    ; DSR flag 1
DF2         equ         2    ; DSR flag 2
DRF1        equ         12   ; DSR receive reg 1 full
DRF0        equ         13   ; DSR receive reg 0 full
DTE1        equ         14   ; DSR transmit reg 1 empty
DTE0        equ         15   ; DSR transmit reg 0 empty

; SAP/portA and BBP/portB
PCRA        equ         $FFBF; SAP GPIO control register
PRRA        equ         $FFBE; SAP GPIO data direction register
PDRA        equ         $FFBD; SAP GPIO data register
STDA        equ         5    ; used as port A gpio pin #5
PCRB        equ         $FFAF; BBP GPIO control register
PRRB        equ         $FFAE; BBP GPIO data direction register
PDRB        equ         $FFAD; BBP GPIO data register
STDB        equ         5    ; used as port B gpio pin #5

;************************************************************************
```

```
; Begining of code
;***********************************************************************
      org   P:$800      ; bootloader begins at start of ROM
START
            ; configured SAP and BBP as gpio inputs
            move        #<$80,r0
            movep       r0,x:PCRA; gpio, PEN bit set, others cleared
            movep       r0,x:PCRB; gpio, PEN bit set, others cleared
            move        #0,x0
            movep       x0,x:PRRA; gpio inputs
            movep       x0,x:PRRB; gpio inputs
            nop

            ; STDA  STDB
            ; ----  ----
            ;  1     1    boot mode A, normal boot
            ;  0     1    boot mode B, jump to user ROM
            ;  1     0    boot mode C, messaging unit boot
            ;  0     0    SPS modes
            jset        #STDA,x:PDRA,START_BOOT
            jset        #STDB,x:PDRB,START_BOOT
            ; else, continue with SPS code


;***********************************************************************
; SPS MODES
;
;   Approx 325 words of Program ROM are reserved for SPS test modes
;     at this location
;***********************************************************************
      ; code for SPS test modes resides here



;***********************************************************************
; boot modes A, B, C
;***********************************************************************
START_BOOT
            ; if we got here, STDA or STDB must have been set
            jclr        #STDA,x:PDRA,START_BOOT_MODE_B
            jclr        #STDB,x:PDRB,START_BOOT_MODE_C
            ; else, both set, continue with BOOT_MODE_A

;***********************************************************************
; BOOT MODE A “NORMAL” MoDE
;***********************************************************************
START_BOOT_MODE_A
_wait       jclr        #DRF0,x:DSR,_wait; wait till DRR0 is full

            ; read message from DRR0
            movep       x:DRR0,x0

            ; short or long message?
            jclr        #15,x0,long_message
```

```
            ; else it’s a short message

            ; handle short messages
short_message
            ; there are currently no allowed short messages
            ; return an invalid message indication

            move        #>inval_short_msg,x1
            jmp         <invalid_message


            ; handle long messages
long_message
            ; retrieve long message opcode
            move        x0,a
            and         #$03FF,a; save only lower 10 bits (offset)
            add         #MDI_base,a; add MDI base address
            move        a1,r0
            move        x:(r0)+,x0; x0=long message opcode

            ; which long message is it?
            move        x0,a
            cmp         #mem_write,a
            jeq         <memory_write
            cmp         #mem_read,a
            jeq         <memory_read
            cmp         #start_app,a
            jeq         <start_application
            cmp         #mem_check,a
            jeq         <memory_check

            ; if it didn’t match any of these, it’s invalid long message
            move        #>inval_long_msg,x1

invalid_message
            ; return a invalid message indication
_wait1      jclr        #DTE0,x:DSR,_wait1; don’t clobber a previous message
_wait2      jclr        #DTE1,x:DSR,_wait2; don’t clobber a previous message
            movep       x0,x:DTR1    ; put invalid data in DTR1
            movep       x1,x:DTR0    ; invalid_opcode.indication in DTR0
            jmp         <START_BOOT_MODE_A; and return to start

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; start memory_write.request
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
memory_write
            jsr         <download_from_mcore
            jmp         <START_BOOT_MODE_A

;-------------------------------------------------------
; download_from_mcore
;   This subroutine is used to perform
```

```
;   memory downloads from the M.CORE to the DSP.
;
; Inputs:
;   r0 -- points to MDI memory, 1 location
;         past memory_write.request
;
; Registers Used :
;             R   M   N   A   B   X   Y
;         0   c   -   c   c   c   c   -
;         1   c   -   c   c   -   -   -
;         2   -   -   -   c   -
;         3   -   -   -
;         4   -   -   -
;         5   -   -   -          c = changed
;         6   -   -   -
;         7   -   -   -
;---------------------------------------------------------

              xdef        download_from_mcore
download_from_mcore
              ; retrieve number of “words” to process
              move        x:(r0)+,n0; n0=#words

              ; retrieve memory space/MDI address
              move        x:(r0)+,x0
              move        x0,a
              and         #$03FF,a; keep lower 10 bits
              add         #MDI_base,a
              move        a1,r1       ; r1=MDI memory address

              ; retrieve DSP memory address
              move        x:(r0),r0; r0=DSP memory address

              ; which memory space?
              move        x0,a
              and         #$C000,a; keep only upper 2 bits
              cmp         #mem_x,a
              jeq         <mem_write_x
              cmp         #mem_y,a
              jeq         <mem_write_y
              cmp         #mem_p,a
              jeq         <mem_write_p
              ; if it didn’t match, it’s invalid
              move        #write_fail,b0
              jmp         <mem_write_return

mem_write_x
              do          n0,_end
              move        x:(r1)+,x0
              move        x0,x:(r0)+
_end
              jmp         <mem_write_success

mem_write_y
```

```
            do          n0,_end
            move        x:(r1)+,x0
            move        x0,y:(r0)+
_end
            jmp         <mem_write_success

mem_write_p
            move        (r1)+           ; point to low word first
            move        #3,n1
            do          n0,_end
            movep       x:(r1)-,x:BPMRL    ; read data in big-endian
            movep       x:(r1)+n1,x:BPMRH ;  format. This looks odd,
            movep       x:<<BPMRG,p:(r0)+   ;  but it's faster and more
            nop                           ;  efficient
_end
            ; continue with mem_write_success

mem_write_success
            ; return memory_write.confirm short message with SUCCESS
            move        #write_success,b0

mem_write_return
            ; return memory_write.confirm short message with FAIL
_wait       jclr        #DTE0,x:DSR,_wait; make sure DTR0 is not full
            movep       b0,x:DTR0
            rts

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; end of memory_write.request
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; start memory_read.request
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
memory_read
            jsr         <upload_to_mcore
            jmp         <START_BOOT_MODE_A

;---------------------------------------------------------
; upload_to_mcore
;   This subroutine is used to perform
;   memory uploads from the DSP to the M.CORE.
;
; Inputs:
;   r0 -- points to MDI memory, 1 location
;         past memory_read.request
;
; Registers Used :
;           R   M   N   A   B   X   Y
;       0   c   -   c   c   c   -   -
;       1   c   -   c   c   c   -   -
;       2   -   -   -   c   c
```

```
;         3   -   -   -
;         4   -   -   -
;         5   -   -   -         c = changed
;         6   -   -   -
;         7   -   -   -
;-----------------------------------------------------------

            xdef        upload_to_mcore
upload_to_mcore
            ; retrieve number of "words" to process
            move        x:(r0)+,n0; n0=#words

            ; retrieve memory space/MDI address
            move        x:(r0)+,x0
            move        x0,a
            and         #$03FF,a; keep lower 10 bits
            move        a1,n1        ; save MDI offset to n1
            add         #MDI_base,a
            move        a1,r1        ; r1=MDI memory address

            ; retrieve DSP memory address
            move        x:(r0),r0; r0=DSP memory address

            ; write 1st header word
            move        #mem_read,b0
            move        b0,x:(r1)+; memory_read.indication-long

            ; which memory space?
            move        x0,a
            and         #$C000,a; keep only upper 2 bits
            cmp         #mem_invalid,a
            jeq         <mem_read_fail
            ; if it gets here, it's a valid memory space
            ;  write (successful) MDI header info
            move        #success,b0
            move        b0,x:(r1)+; return code (success | fail)
            move        n0,x:(r1)+; # words
            move        x0,b         ; old memory space & MDI address
            add         #5,b         ;  new MDI address is offset by 5
            move        b1,x:(r1)+; memory space & MDI address
            move        r0,x:(r1)+; DSP source address
            cmp         #mem_x,a
            jeq         <mem_read_x
            cmp         #mem_y,a
            jeq         <mem_read_y
            ; only option left is mem_read_p
            jmp         <mem_read_p

mem_read_x
            do          n0,_loop
            move        x:(r0)+,x0
            move        x0,x:(r1)+
_loop
            jmp         <mem_read_return
```

```
mem_read_y
            do          n0,_loop
            move        y:(r0)+,x0
            move        x0,x:(r1)+
_loop
            jmp         <mem_read_return

mem_read_p
            do          n0,_loop
            movep       p:(r0)+,x:<<BPMRG
            movep       x:BPMRH,x:(r1)+; store p data in
            movep       x:BPMRL,x:(r1)+;  big-endian format
_loop
            jmp         <mem_read_return

mem_read_fail
            ;  write (unsuccessful) MDI header info
            move        #fail,b0
            move        b0,x:(r1)+; return code (fail)

mem_read_return
            ; form long message return (same for both success and failure)
            move        n1,a        ; MDI address for long
            or          #long_header,a
_wait       jclr        #DTE0,x:DSR,_wait; don't clobber a previous message
            movep       a1,x:DTR0

            rts


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; end of memory_read.request
;
; start "512pram" memory_check.request
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
memory_check
            ; retrieve memory_type and address
            move        x:(r0),x1
            move        x1,a1
            and         #$03FF,a; save only lower 10 bits (offset)
            move        a1,n1       ; save offset, needed for return
            add         #MDI_base,a; add MDI base address
            move        a1,r1       ; return mdi address
            move        #mem_check,b0; write memory_check.confirm
            move        b0,x:(r1)+  ;   as header
            move        x1,a
            and         #$e000,a; keep upper 3 bits
            cmp         #pram512,a
            jeq         <pram_check
            ; else, it's not a valid memory space
            move        #fail_inv_mem,b0; return code - fail invalid memory
```

```
            move        b0,x:(r1)
            jmp         <mem_check_return
            ;
            ; "check 512 word p-ram space"
            ;

pram_check
            move    #PATTERNS,r3    ; r3 points to p: test patterns

            do          #NUM_PATTERNS/4,_loop_o
            ; up(wB)

            movem   p:(r3)+,n4; get BackGround Pattern (high word)
            movem   p:(r3)+,n3; get BackGround Pattern (low word)
            movep       n4,x:BPMRH
            movep       n3,x:BPMRL

            move    #0,r0           ; r0  points to start of Memory
            rep     #512             ; fill Memory with  BG Pattern: up(wB)
            movep       x:BPMRG,p:(r0)+

            ; up(rB,wD,rD)

            clr         a
            clr         b
            move        #0,r0


            movem   p:(r3)+,n6; get Data Pattern (high word)
            movem   p:(r3)+,n5; get Data Pattern (low word)
            movep       n6,x:BPMRH
            movep       n5,x:BPMRL

            do          #512,_loop_i  ; test all locations
            move        n3,a0         ; BG Pattern  value to A
            move        n4,a1
            movep       p:(r0),x:BPMRG; read  BackGround Pattern -> BPMRG
;           move        #$ABCD,n2   ; change gdb ????
            movep       x:BPMRL,b0
            movep       x:BPMRH,b1
            cmp         a,b               ; was the Memory data as expected???
            nop
            brkne
            movep       n5,x:BPMRL  ; restore low byte of DATA from n5
            movep       n6,x:BPMRH  ; restore high byte of DATA from n6
            nop
            movep       x:BPMRG,p:(r0)      ; write Data to Memory
            move        #$ABCD,n2   ; change gdb
            movep       p:(r0),x:BPMRG; read  Data  Pattern -> BPMRG
            movep       x:BPMRL,b0  ; read  Data  Pattern -> B
            movep       x:BPMRH,b1
            move        n5,a0         ; restore low byte of DATA from n5
            move        n6,a1         ; restore high byte of DATA from n6
            cmp         a,b               ; was the Memory data as expected???
```

```
            nop
            brkne
            move        (r0)+
            nop
            nop
_loop_i
            brkne
            nop
            nop
            nop
_loop_o
            move        #success,r2
            move        #fail,r4
            tne         r4,r2
            move        r2,x:(r1)+   ; write success/fail
            move        r0,x:(r1)+   ; write address

mem_check_return
            ; form long message return (same for both success and failure)
            move        n1,a           ; n1 = offset
            or          #long_header,a
_wait       jclr        #DTE0,x:DSR,_wait; don't clobber a previous message
            movep       a1,x:DTR0

            jmp         <START_BOOT_MODE_A

            ; the following patterns are used by boot mode A mem_check.request
            BADDR       M,8  ; place on modulo boundary for burnin mode
PATTERNS
            dc          $0055; background pattern high word
            dc          $5555   ; background pattern low word
            dc          $00AA; data pattern high word
            dc          $AAAA; data pattern low word
            dc          $00CC; background pattern high word
            dc          $CCCC   ; background pattern low word
            dc          $0033; data pattern high word
            dc          $3333; data pattern low word
NUM_PATTERNSequ*-PATTERNS

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; end of memory_check.request
;
; start start_application.request
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
start_application
            move        x:(r0),r0
            jmp         r0


;*********************************************************************
; BOOT MODE B Shared memory Mode
;*********************************************************************
```

```
START_BOOT_MODE_B
             move        #MDI_base,r0
             ; look for protocol_B_signature_0
_wait0
             move        x:(r0),a
             cmp         #>prot_B_sig_0,a
             jne         <_wait0

             ; reply with protocol_B_confirm_0
             move        #>prot_B_conf_0,x0
             move        x0,x:(r0+1)

             ; look for protocol_B_signature_1
_wait1
             move        x:(r0),a
             cmp         #prot_B_sig_1,a
             jne         <_wait1

             ; reply with protocol_B_confirm_1
             move        #prot_B_conf_1,x0
             move        x0,x:(r0+1)

             ; okay, do the download
             move        #0,r1        ; start of p: memory to download
             lea         (r0+3),r0; MDI_base+3
             move        #3,n0
             do          #511,_end
             movep       x:(r0)-,x:BPMRL     ; read data in
             movep       x:(r0)+n0,x:BPMRH   ;    big-endian format
             movep       x:<<BPMRG,p:(r1)+
             nop
_end
             jmp         <0

;**********************************************************************
; BOOT MODE C Message Unit Mode
;**********************************************************************
START_BOOT_MODE_C
             move        #0,r0
             do          #511,_loop
_wait0       jclr        #DRF0,x:DSR,_wait0; wait till DRR0 is full
             movep       x:DRR0,a1
_wait1       jclr        #DRF1,x:DSR,_wait1; wait till DRR1 is full
             movep       x:DRR1,a0
             movep       a0,x:<<BPMRL
             movep       a1,x:<<BPMRH
             nop
             movep       x:<<BPMRG,p:(r0)+; write to pram512
             nop
_loop
             jmp         <0

             endsec
             end
```

Appendix B
Equates and Header Files

This appendix provides the equates for both the MCU and DSP in the DSP56652, as well as a C include file for the MCU. If code for external bootstrap loading is developed, a file containing this listing called ioequ.asm should be included in the bootstrap executable.

B.1 MCU Equates

```
//=====================================================================
//=====================================================================
//
// DSP56651/DSP56652 M.CORE Assembly equates
//
// Revision History:
// 1.0: may 28,         1998
//
//=====================================================================
//=====================================================================

// 16kb on-chip rom
      .equ  mcu_rom_base_address,    0x00000000
      .equ  mcu_rom_size,            0x00004000
// 2kb on-chip ram
      .equ  mcu_ram_base_address,    0x00100000
      .equ  mcu_ram_size,            0x00000800

// peripheral space
      .equ  mcu_peripherals_base_address,0x00200000

// 0x00300000 through 0x3fffffff is reserved

// external memory
      .equ  cs0_base_address,        0x40000000
      .equ  cs1_base_address,        0x41000000
      .equ  cs2_base_address,        0x42000000
      .equ  cs3_base_address,        0x43000000
      .equ  cs4_base_address,        0x44000000
      .equ  cs5_base_address,        0x45000000

// 0x46000000 through 0xffffffff is reserved
```

```
//==========================================================================
// MCU-DSP Interface (MDI) equates
//==========================================================================

// general definitions

      .equ  mdi_registers_base_address,    0x00202ff0
      .equ  mdi_memory_base_address,       0x00202000

// registers of the messaging unit
      .equ  mdi_mcvr,    0x2  // MCU-side Command Vector Register
      .equ  mdi_mcr,     0x4 // MCU-side Control Register
      .equ  mdi_msr,     0x6 // MCU-side Status Register
      .equ  mdi_mtr1,    0x8 // MCU-side Transmit Register 1
      .equ  mdi_mtr0,    0xa // MCU-side Transmit Register 0
      .equ  mdi_mrr1,    0xc // MCU-side Recieve Register 1
      .equ  mdi_mrr0,    0xe // MCU-side Receive Register 0

// bits of the MCU-side Command Vector register (MCVR)
      .equ  mdi_mcvr_mnmi,    0x0 // MCU-command Non-Maskable Interrupt
      .equ  mdi_mcvr_mc,      0x8 // MCU-Command active bit

// bits of the MCU-side Control Register (MCR)
      .equ  mdi_mcr_mdf0,     0x0 // MCU to DSP Flag 0
      .equ  mdi_mcr_mdf1,     0x1 // MCU to DSP Flag 1
      .equ  mdi_mcr_mdf2,     0x2 // MCU to DSP Flag 2
      .equ  mdi_mcr_mdir,     0x6 // MDI software Reset
      .equ  mdi_mcr_dhr,      0x7 // DSP Hardware Reset
      .equ  mdi_mcr_mgie1,    0xa // MCU General Interrupt 0 enable
      .equ  mdi_mcr_mgie0,    0xb // MCU General Interrupt 1 enable
      .equ  mdi_mcr_mtie1,    0xc // MCU transmit Interrupt 1 enable
      .equ  mdi_mcr_mtie0,    0xd // MCU transmit Interrupt 0 enable
      .equ  mdi_mcr_mrie1,    0xe // MCU Receive Interrupt 1 enable
      .equ  mdi_mcr_mrie0,    0xf // MCU Receive Interrupt 0 enable

// bits of the MCU-side Status Register (MSR)
      .equ  mdi_msr_mf0,      0x0 // MCU-side Flag 0
      .equ  mdi_msr_mf1,      0x1 // MCU-side Flag 1
      .equ  mdi_msr_mf2,      0x2 // MCU-side Flag 2
      .equ  mdi_msr_mep,      0x4 // MCU-side Event Pending
      .equ  mdi_msr_dpm,      0x5 // DSP power mode
      .equ  mdi_msr_msmp,     0x6 // MCU Shared Memory access pending
      .equ  mdi_msr_drs,      0x7 // DSP Reset State
      .equ  mdi_msr_dws,      0x8 // DSP Wake from Stop
      .equ  mdi_msr_mtir,     0x9 // MCU Protocol Timer wake DSP from stop & IRQ
      .equ  mdi_msr_mgip1,    0xa // MCU General Interrupt 1 pending
      .equ  mdi_msr_mgip0,    0xb // MCU General Interrupt 0 pending
      .equ  mdi_msr_mte1,     0xc // MCU transmit register 1 empty
      .equ  mdi_msr_mte0,     0xd // MCU transmit register 0 empty
      .equ  mdi_msr_mrf1,     0xe // MCU Receive register 1 full
      .equ  mdi_msr_mrf0,     0xf // MCU Receive register 0 full

//==========================================================================
// Protocol timer (prot) equates
```

```
//==========================================================================

// general definitions
      .equ  prot_memory_base_address,                    0x00203000
      .equ  prot_programable_registers_base_address,     0x00203800
      .equ  prot_testmode_registers_base_address,             0x00203c00

// programable registers of the protocol timer
      .equ  prot_tctr,   0x0 //Timer control register, old name
      .equ  prot_ptcr,   0x0 //Timer control register, NEW NAME
      .equ  prot_tier,   0x2 //timer interrupt enable register, old name
      .equ  prot_ptier,  0x2 //timer interrupt enable register, NEW NAME
      .equ  prot_tstr,   0x4 //timer status register, old name
      .equ  prot_ptsr,   0x4 //timer status register, NEW NAME
      .equ  prot_tevr,   0x6 //timer event register, old name
      .equ  prot_ptevr,  0x6 //timer event register, NEW NAME
      .equ  prot_tipr,   0x8 //time interval _prescaler_, old name
      .equ  prot_timl,   0x8 //time interval _modulus latch_, NEW NAME
      .equ  prot_ctic,   0xa //Channel time interval counter
      .equ  prot_ctipr,  0xc //Channel time interval _preload register_, old name
      .equ  prot_ctiml,  0xc //Channel time interval _modulus latch_, NEW NAME
      .equ  prot_cfc,    0xe //Channel frames counter
      .equ  prot_cfpr,   0x10 //Channel frames _preload register_, old name
      .equ  prot_cfml,   0x10 //Channel frames _modulus latch_, NEW NAME
      .equ  prot_rsc,    0x12 //Reference slot counter
      .equ  prot_rspr,   0x14 //Reference slot _preload register_, old name
      .equ  prot_rsml,   0x14 //Reference slot _modulus latch_, NEW NAME
      .equ  prot_pdpar,  0x16 //Port D functionalty register, old name
      .equ  prot_ptpcr,  0x16 //Protocol Timer Port Control Register, NEW NAME
      .equ  prot_pddr,   0x18 //Port D directivity register, old name
      .equ  prot_ptddr,  0x18 //Protocol Timer Data Direction Register, NEW NAME
      .equ  prot_pddat,  0x1a //Port D data Register, old name
      .equ  prot_ptpdr,  0x1a //Protocol Timer Port Data Register, NEW NAME
      .equ  prot_ftptr,  0x1c //Frame tables pointers
      .equ  prot_rtptr,  0x1e //Receive/Transmit Macro tables pointers, old name
      .equ  prot_mtptr,  0x1e //Macro table pointers, NEW NAME
      .equ  prot_ftbar,  0x20 //Frame tables base address register
      .equ  prot_rtbar,  0x22 //Rx/Tx Macro tables base address register, old name
      .equ  prot_mtbar,  0x22 //Macro table base address register, NEW NAME
      .equ  prot_dtptr,  0x24 //Delay tables pointers.

// bits of the Timer Control Register (TCTR)(old names)
      .equ  prot_tctr_te,     0x0 // timer enable bit.
      .equ  prot_tctr_time,   0x1 // timer immidiate enable bit.
      .equ  prot_tctr_mter,   0x2 // macro termination bit
      .equ  prot_tctr_tdzd,   0x3 // Timer doze disable.
      .equ  prot_tctr_spbp,   0x4 // slot prescaler by-pass bit
      .equ  prot_tctr_hltr,   0x5 // halt request bit
      .equ  prot_tctr_cfce,   0x8 // cfc counter enable bit
      .equ  prot_tctr_rsce,   0x9 // rsc counter enable bit

// bits of the Protocol Timer Control Register (PTCR)(NEW NAMES)
      .equ  prot_ptcr_te,     0x0 // timer enable bit.
      .equ  prot_ptcr_time,   0x1 // timer immidiate enable bit.
```

```
        .equ  prot_ptcr_mter,   0x2 // macro termination bit
        .equ  prot_ptcr_tdzd,   0x3 // Timer doze disable.
        .equ  prot_ptcr_spbp,   0x4 // slot prescaler by-pass bit
        .equ  prot_ptcr_hltr,   0x5 // halt request bit
        .equ  prot_ptcr_cfce,   0x8 // cfc counter enable bit
        .equ  prot_ptcr_rsce,   0x9 // rsc counter enable bit

// bits of the Timer Interrupt Enable Register (TIER)(old names)
        .equ  prot_ptier_cfie ,0x0 // channel frame interrupt enable bit
        .equ  prot_ptier_cfnie ,0x1 // channel frame number intpt enable bit
        .equ  prot_ptier_rsnie ,0x2 // reference slot number intpt enable bit
        .equ  prot_ptier_mcie0 ,0x4 // MCU interrupt 0 enable bit
        .equ  prot_ptier_mcie1 ,0x5 // MCU interrupt 1 enable bit
        .equ  prot_ptier_mcie2 ,0x6 // MCU interrupt 2 enable bit
        .equ  prot_ptier_dsie ,0x9 // DSP interrupt enable bit
        .equ  prot_ptier_dvie ,0xa // DSP vector interrupt enable bit
        .equ  prot_ptier_thie ,0xb // Timer haltinterrupt enable bit
        .equ  prot_ptier_terie ,0xc // Timer error interrupt enable bit

// bits of the Protocol Timer Interrupt Enable Register (PTIER)(NEW NAMES)
        .equ  prot_tier_cfie ,  0x0 // channel frame interrupt enable bit
        .equ  prot_tier_cfnie ,0x1 // channel frame number intpt enable bit
        .equ  prot_tier_rsnie ,0x2 // reference slot number intpt enable bit
        .equ  prot_tier_mcie0 ,0x4 // MCU interrupt 0 enable bit
        .equ  prot_tier_mcie1 ,0x5 // MCU interrupt 1 enable bit
        .equ  prot_tier_mcie2 ,0x6 // MCU interrupt 2 enable bit
        .equ  prot_tier_dsie ,  0x9 // DSP interrupt enable bit
        .equ  prot_tier_dvie ,  0xa // DSP vector interrupt enable bit
        .equ  prot_tier_thie ,  0xb // Timer haltinterrupt enable bit
        .equ  prot_tier_terie ,0xc // Timer error interrupt enable bit

// bits of the Timer Status Register (TSTR)(old names)
        .equ  prot_tstr_cfi ,   0x0 // channel frame interrupt bit
        .equ  prot_tstr_cfni ,  0x1 // channel frame number interrupt bit
        .equ  prot_tstr_rsni ,  0x2 // reference slot number interrupt bit
        .equ  prot_tstr_mcui0 ,0x4 // MCU interrupt 0 bit
        .equ  prot_tstr_mcui1 ,0x5 // MCU interrupt 1 bit
        .equ  prot_tstr_mcui2 ,0x6 // MCU interrupt 2 bit
        .equ  prot_tstr_dsi ,   0x9 // DSP interrupt bit
        .equ  prot_tstr_dvi ,   0xa // DSP vector interrupt bit
        .equ  prot_tstr_thi ,   0xb // Timer haltinterrupt bit
        .equ  prot_tstr_teri ,  0xc // Timer error interrupt bit

// bits of the Protocol Timer Status Register (PTSR)(NEW NAMES)
        .equ  prot_ptsr_cfi ,   0x0 // channel frame interrupt bit
        .equ  prot_ptsr_cfni ,  0x1 // channel frame number interrupt bit
        .equ  prot_ptsr_rsni ,  0x2 // reference slot number interrupt bit
        .equ  prot_ptsr_mcui0 ,0x4 // MCU interrupt 0 bit
        .equ  prot_ptsr_mcui1 ,0x5 // MCU interrupt 1 bit
        .equ  prot_ptsr_mcui2 ,0x6 // MCU interrupt 2 bit
        .equ  prot_ptsr_dsi ,   0x9 // DSP interrupt bit
        .equ  prot_ptsr_dvi ,   0xa // DSP vector interrupt bit
        .equ  prot_ptsr_thi ,   0xb // Timer haltinterrupt bit
        .equ  prot_ptsr_teri ,  0xc // Timer error interrupt bit
```

```
// bits of the Timer Event Register (TEVR)(old names)
      .equ  prot_tevr_act ,  0x0 // active table indicator bit
      .equ  prot_tevr_rxma , 0x1 // active Rx macro indicator bit
      .equ  prot_tevr_txma , 0x2 // active Tx macro indicator bit
      .equ  prot_tevr_thip , 0x3 // timer halt in progress indicator bit

// bits of the Protocol Timer Event Register (PTEVR)(NEW NAMES)
      .equ  prot_ptevr_act , 0x0 // active table indicator bit
      .equ  prot_ptevr_rxma ,0x1 // active Rx macro indicator bit
      .equ  prot_ptevr_txma ,0x2 // active Tx macro indicator bit
      .equ  prot_ptevr_thip ,0x3 // timer halt in progress indicator bit

// bits of the Time Interval Preload Register (TIPR)(old names)
      .equ  prot_tipr_tipv_0 ,0x0 // TIPR value-bit 0
      .equ  prot_tipr_tipv_1 ,0x1 // TIPR value-bit 1
      .equ  prot_tipr_tipv_2 ,0x2 // TIPR value-bit 2
      .equ  prot_tipr_tipv_3 ,0x3 // TIPR value-bit 3
      .equ  prot_tipr_tipv_4 ,0x4 // TIPR value-bit 4
      .equ  prot_tipr_tipv_5 ,0x5 // TIPR value-bit 5
      .equ  prot_tipr_tipv_6 ,0x6 // TIPR value-bit 6
      .equ  prot_tipr_tipv_7 ,0x7 // TIPR value-bit 7
      .equ  prot_tipr_tipv_8 ,0x8 // TIPR value-bit 8

// bits of the Time Interval Modulus Register (TIMR)(NEW NAMES)
      .equ  prot_timl_timv_0 ,0x0 // timl value-bit 0
      .equ  prot_timl_timv_1 ,0x1 // timl value-bit 1
      .equ  prot_timl_timv_2 ,0x2 // timl value-bit 2
      .equ  prot_timl_timv_3 ,0x3 // timl value-bit 3
      .equ  prot_timl_timv_4 ,0x4 // timl value-bit 4
      .equ  prot_timl_timv_5 ,0x5 // timl value-bit 5
      .equ  prot_timl_timv_6 ,0x6 // timl value-bit 6
      .equ  prot_timl_timv_7 ,0x7 // timl value-bit 7
      .equ  prot_timl_timv_8 ,0x8 // timl value-bit 8

// bits of the Channel Time Interval Counter (CTIC)
      .equ  prot_ctic_ctiv_0,       0x0 // CTIC value-bit 0
      .equ  prot_ctic_ctiv_1,       0x1 // CTIC value-bit 1
      .equ  prot_ctic_ctiv_2,       0x2 // CTIC value-bit 2
      .equ  prot_ctic_ctiv_3,       0x3 // CTIC value-bit 3
      .equ  prot_ctic_ctiv_4,       0x4 // CTIC value-bit 4
      .equ  prot_ctic_ctiv_5,       0x5 // CTIC value-bit 5
      .equ  prot_ctic_ctiv_6,       0x6 // CTIC value-bit 6
      .equ  prot_ctic_ctiv_7,       0x7 // CTIC value-bit 7
      .equ  prot_ctic_ctiv_8,       0x8 // CTIC value-bit 8
      .equ  prot_ctic_ctiv_9,       0x9 // CTIC value-bit 9
      .equ  prot_ctic_ctiv_10,      0xa // CTIC value-bit 10
      .equ  prot_ctic_ctiv_11,      0xb // CTIC value-bit 11
      .equ  prot_ctic_ctiv_12,      0xc // CTIC value-bit 12
      .equ  prot_ctic_ctiv_13,      0xd // CTIC value-bit 13

// bits of the Channel Time Interval Preload Register (CTIPR)(old names)
      .equ  prot_ctipr_ctipv_0,     0x0 // CTIPR value-bit 0
      .equ  prot_ctipr_ctipv_1,     0x1 // CTIPR value-bit 1
```

```
      .equ  prot_ctipr_ctipv_2,     0x2 // CTIPR value-bit 2
      .equ  prot_ctipr_ctipv_3,     0x3 // CTIPR value-bit 3
      .equ  prot_ctipr_ctipv_4,     0x4 // CTIPR value-bit 4
      .equ  prot_ctipr_ctipv_5,     0x5 // CTIPR value-bit 5
      .equ  prot_ctipr_ctipv_6,     0x6 // CTIPR value-bit 6
      .equ  prot_ctipr_ctipv_7,     0x7 // CTIPR value-bit 7
      .equ  prot_ctipr_ctipv_8,     0x8 // CTIPR value-bit 8
      .equ  prot_ctipr_ctipv_9,     0x9 // CTIPR value-bit 9
      .equ  prot_ctipr_ctipv_10,    0xa // CTIPR value-bit 10
      .equ  prot_ctipr_ctipv_11,    0xb // CTIPR value-bit 11
      .equ  prot_ctipr_ctipv_12,    0xc // CTIPR value-bit 12
      .equ  prot_ctipr_ctipv_13,    0xd // CTIPR value-bit 13

// bits of the Channel Time Interval Modulus Register (CTIMR)(NEW NAMES)
      .equ  prot_ctiml_ctimv_0,     0x0 // ctiml value-bit 0
      .equ  prot_ctiml_ctimv_1,     0x1 // ctiml value-bit 1
      .equ  prot_ctiml_ctimv_2,     0x2 // ctiml value-bit 2
      .equ  prot_ctiml_ctimv_3,     0x3 // ctiml value-bit 3
      .equ  prot_ctiml_ctimv_4,     0x4 // ctiml value-bit 4
      .equ  prot_ctiml_ctimv_5,     0x5 // ctiml value-bit 5
      .equ  prot_ctiml_ctimv_6,     0x6 // ctiml value-bit 6
      .equ  prot_ctiml_ctimv_7,     0x7 // ctiml value-bit 7
      .equ  prot_ctiml_ctimv_8,     0x8 // ctiml value-bit 8
      .equ  prot_ctiml_ctimv_9,     0x9 // ctiml value-bit 9
      .equ  prot_ctiml_ctimv_10,    0xa // ctiml value-bit 10
      .equ  prot_ctiml_ctimv_11,    0xb // ctiml value-bit 11
      .equ  prot_ctiml_ctimv_12,    0xc // ctiml value-bit 12
      .equ  prot_ctiml_ctimv_13,    0xd // ctiml value-bit 13

// bits of the Channel Frame Counter (CFC)
      .equ  prot_cfc_cfcv_0,        0x0 //CFC value-bit 0
      .equ  prot_cfc_cfcv_1,        0x1 //CFC value-bit 1
      .equ  prot_cfc_cfcv_2,        0x2 //CFC value-bit 2
      .equ  prot_cfc_cfcv_3,        0x3 //CFC value-bit 3
      .equ  prot_cfc_cfcv_4,        0x4 //CFC value-bit 4
      .equ  prot_cfc_cfcv_5,        0x5 //CFC value-bit 5
      .equ  prot_cfc_cfcv_6,        0x6 //CFC value-bit 6
      .equ  prot_cfc_cfcv_7,        0x7 //CFC value-bit 7
      .equ  prot_cfc_cfcv_8,        0x8 //CFC value-bit 8

// bits of the Channel Frame Preload Register (CFPR)(old names)
      .equ  prot_cfpr_cfpv_0,       0x0 //CFPR value- bit 1
      .equ  prot_cfpr_cfpv_1,       0x1 //CFPR value- bit 2
      .equ  prot_cfpr_cfpv_2,       0x2 //CFPR value- bit 3
      .equ  prot_cfpr_cfpv_3,       0x3 //CFPR value- bit 4
      .equ  prot_cfpr_cfpv_4,       0x4 //CFPR value- bit 5
      .equ  prot_cfpr_cfpv_5,       0x5 //CFPR value- bit 6
      .equ  prot_cfpr_cfpv_6,       0x6 //CFPR value- bit 7
      .equ  prot_cfpr_cfpv_7,       0x7 //CFPR value- bit 8
      .equ  prot_cfpr_cfpv_8,       0x8 //CFPR value- bit 9

// bits of the Channel Frame Modulus Register (CFMR)(NEW NAMES)
      .equ  prot_cfml_cfmv_0,       0x0 //cfml value- bit 1
      .equ  prot_cfml_cfmv_1,       0x1 //cfml value- bit 2
```

```
        .equ  prot_cfml_cfmv_2,       0x2 //cfml value- bit 3
        .equ  prot_cfml_cfmv_3,       0x3 //cfml value- bit 4
        .equ  prot_cfml_cfmv_4,       0x4 //cfml value- bit 5
        .equ  prot_cfml_cfmv_5,       0x5 //cfml value- bit 6
        .equ  prot_cfml_cfmv_6,       0x6 //cfml value- bit 7
        .equ  prot_cfml_cfmv_7,       0x7 //cfml value- bit 8
        .equ  prot_cfml_cfmv_8,       0x8 //cfml value- bit 9

// bits of the Reference Slot Counter (RSC)
        .equ  prot_rsc_rscv_0,        0x0 //RSC value-bit 0
        .equ  prot_rsc_rscv_1,        0x1 //RSC value-bit 1
        .equ  prot_rsc_rscv_2,        0x2 //RSC value-bit 2
        .equ  prot_rsc_rscv_3,        0x3 //RSC value-bit 3
        .equ  prot_rsc_rscv_4,        0x4 //RSC value-bit 4
        .equ  prot_rsc_rscv_5,        0x5 //RSC value-bit 5
        .equ  prot_rsc_rscv_6,        0x6 //RSC value-bit 6
        .equ  prot_rsc_rscv_7,        0x7 //RSC value-bit 7

// bits of the Reference Slot Preload Register (RSPR) (old names)
        .equ  prot_rspr_rspv_0,       0x0 //RSPR value -bit 0
        .equ  prot_rspr_rspv_1,       0x1 //RSPR value -bit 1
        .equ  prot_rspr_rspv_2,       0x2 //RSPR value -bit 2
        .equ  prot_rspr_rspv_3,       0x3 //RSPR value -bit 3
        .equ  prot_rspr_rspv_4,       0x4 //RSPR value -bit 4
        .equ  prot_rspr_rspv_5,       0x5 //RSPR value -bit 5
        .equ  prot_rspr_rspv_6,       0x6 //RSPR value -bit 6
        .equ  prot_rspr_rspv_7,       0x7 //RSPR value -bit 7

// bits of the Reference Slot Modulus Register (RSMR) (NEW NAMES)
        .equ  prot_rsml_rsmv_0,       0x0 //rsml value -bit 0
        .equ  prot_rsml_rsmv_1,       0x1 //rsml value -bit 1
        .equ  prot_rsml_rsmv_2,       0x2 //rsml value -bit 2
        .equ  prot_rsml_rsmv_3,       0x3 //rsml value -bit 3
        .equ  prot_rsml_rsmv_4,       0x4 //rsml value -bit 4
        .equ  prot_rsml_rsmv_5,       0x5 //rsml value -bit 5
        .equ  prot_rsml_rsmv_6,       0x6 //rsml value -bit 6
        .equ  prot_rsml_rsmv_7,       0x7 //rsml value -bit 7

// bits of the Port D Pin Assignment Register (PDPAR) (old names)
        .equ  prot_pdpar_pdgpc_0,0x0 //Select the function of pin 0 in port D
        .equ  prot_pdpar_pdgpc_1,0x1 //Select the function of pin 1 in port D
        .equ  prot_pdpar_pdgpc_2,0x2 //Select the function of pin 2 in port D
        .equ  prot_pdpar_pdgpc_3,0x3 //Select the function of pin 3 in port D
        .equ  prot_pdpar_pdgpc_4,0x4 //Select the function of pin 4 in port D
        .equ  prot_pdpar_pdgpc_5,0x5 //Select the function of pin 5 in port D
        .equ  prot_pdpar_pdgpc_6,0x6 //Select the function of pin 6 in port D
        .equ  prot_pdpar_pdgpc_7,0x7 //Select the function of pin 7 in port D

// bits of the Protocol Timer Port Control Register (PTPCR) (NEW NAMES)
        .equ  prot_ptpcr_ptpc_0,0x0 //Select the function of pin 0 in port D
        .equ  prot_ptpcr_ptpc_1,0x1 //Select the function of pin 1 in port D
        .equ  prot_ptpcr_ptpc_2,0x2 //Select the function of pin 2 in port D
        .equ  prot_ptpcr_ptpc_3,0x3 //Select the function of pin 3 in port D
        .equ  prot_ptpcr_ptpc_4,0x4 //Select the function of pin 4 in port D
```

```
        .equ  prot_ptpcr_ptpc_5,0x5 //Select the function of pin 5 in port D
        .equ  prot_ptpcr_ptpc_6,0x6 //Select the function of pin 6 in port D
        .equ  prot_ptpcr_ptpc_7,0x7 //Select the function of pin 7 in port D

// bits of the Port D Direction Register Register (PDDR) (old names)
        .equ  prot_pddr_pddr_0,0x0 //Select the direction of pin 0 in port D
        .equ  prot_pddr_pddr_1,0x1 //Select the direction of pin 1 in port D
        .equ  prot_pddr_pddr_2,0x2 //Select the direction of pin 2 in port D
        .equ  prot_pddr_pddr_3,0x3 //Select the direction of pin 3 in port D
        .equ  prot_pddr_pddr_4,0x4 //Select the direction of pin 4 in port D
        .equ  prot_pddr_pddr_5,0x5 //Select the direction of pin 5 in port D
        .equ  prot_pddr_pddr_6,0x6 //Select the direction of pin 6 in port D
        .equ  prot_pddr_pddr_7,0x7 //Select the direction of pin 7 in port D

// bits of the Protocol Timer Data Direction Register (PTDDR) (NEW NAMES)
        .equ  prot_ptddr_ptdd_0,0x0 //Select the direction of pin 0 in port D
        .equ  prot_ptddr_ptdd_1,0x1 //Select the direction of pin 1 in port D
        .equ  prot_ptddr_ptdd_2,0x2 //Select the direction of pin 2 in port D
        .equ  prot_ptddr_ptdd_3,0x3 //Select the direction of pin 3 in port D
        .equ  prot_ptddr_ptdd_4,0x4 //Select the direction of pin 4 in port D
        .equ  prot_ptddr_ptdd_5,0x5 //Select the direction of pin 5 in port D
        .equ  prot_ptddr_ptdd_6,0x6 //Select the direction of pin 6 in port D
        .equ  prot_ptddr_ptdd_7,0x7 //Select the direction of pin 7 in port D

// bits of the Port D Data Register (PDDAT) (old names)
        .equ  prot_pddat_pddat_0,     0x0 //Port D Data- pin 0
        .equ  prot_pddat_pddat_1,     0x1 //Port D Data- pin 1
        .equ  prot_pddat_pddat_2,     0x2 //Port D Data- pin 2
        .equ  prot_pddat_pddat_3,     0x3 //Port D Data- pin 3
        .equ  prot_pddat_pddat_4,     0x4 //Port D Data- pin 4
        .equ  prot_pddat_pddat_5,     0x5 //Port D Data- pin 5
        .equ  prot_pddat_pddat_6,     0x6 //Port D Data- pin 6
        .equ  prot_pddat_pddat_7,     0x7 //Port D Data- pin 7

// bits of the Protocol Timer Port Data Register (PTPDR) (NEW NAMES)
        .equ  prot_ptpdr_ptpd_0,      0x0 //Port D Data- pin 0
        .equ  prot_ptpdr_ptpd_1,      0x1 //Port D Data- pin 1
        .equ  prot_ptpdr_ptpd_2,      0x2 //Port D Data- pin 2
        .equ  prot_ptpdr_ptpd_3,      0x3 //Port D Data- pin 3
        .equ  prot_ptpdr_ptpd_4,      0x4 //Port D Data- pin 4
        .equ  prot_ptpdr_ptpd_5,      0x5 //Port D Data- pin 5
        .equ  prot_ptpdr_ptpd_6,      0x6 //Port D Data- pin 6
        .equ  prot_ptpdr_ptpd_7,      0x7 //Port D Data- pin 7

// bits of the Frame Table Pointer (FTPTR)
        .equ  prot_ftptr_ftptr_0,     0x0 //Frame table pointer-bit0
        .equ  prot_ftptr_ftptr_1,     0x1 //Frame table pointer-bit1
        .equ  prot_ftptr_ftptr_2,     0x2 //Frame table pointer-bit2
        .equ  prot_ftptr_ftptr_3,     0x3 //Frame table pointer-bit3
        .equ  prot_ftptr_ftptr_4,     0x4 //Frame table pointer-bit4
        .equ  prot_ftptr_ftptr_5,     0x5 //Frame table pointer-bit5
        .equ  prot_ftptr_ftptr_6,     0x6 //Frame table pointer-bit6

// bits of the Receive/Transmit macro Table Pointer (RTPTR)(old names)
```

```
        .equ   prot_rtptr_rxptr_0,      0x0 //Receive macro pointer-bit 0
        .equ   prot_rtptr_rxptr_1,      0x1 //Receive macro pointer-bit 1
        .equ   prot_rtptr_rxptr_2,      0x2 //Receive macro pointer-bit 2
        .equ   prot_rtptr_rxptr_3,      0x3 //Receive macro pointer-bit 3
        .equ   prot_rtptr_rxptr_4,      0x4 //Receive macro pointer-bit 4
        .equ   prot_rtptr_rxptr_5,      0x5 //Receive macro pointer-bit 5
        .equ   prot_rtptr_rxptr_6,      0x6 //Receive macro pointer-bit 6

        .equ   prot_rtptr_txptr_0,      0x8 //Transmit macro pointer-bit 0
        .equ   prot_rtptr_txptr_1,      0x9 //Transmit macro pointer-bit 1
        .equ   prot_rtptr_txptr_2,      0xa //Transmit macro pointer-bit 2
        .equ   prot_rtptr_txptr_3,      0xb //Transmit macro pointer-bit 3
        .equ   prot_rtptr_txptr_4,      0xc //Transmit macro pointer-bit 4
        .equ   prot_rtptr_txptr_5,      0xd //Transmit macro pointer-bit 5
        .equ   prot_rtptr_txptr_6,      0xe //Transmit macro pointer-bit 6

// bits of the Macro Table Pointer (RTPTR)(NEW NAMES)
        .equ   prot_mtptr_rxptr_0,      0x0 //Receive macro pointer-bit 0
        .equ   prot_mtptr_rxptr_1,      0x1 //Receive macro pointer-bit 1
        .equ   prot_mtptr_rxptr_2,      0x2 //Receive macro pointer-bit 2
        .equ   prot_mtptr_rxptr_3,      0x3 //Receive macro pointer-bit 3
        .equ   prot_mtptr_rxptr_4,      0x4 //Receive macro pointer-bit 4
        .equ   prot_mtptr_rxptr_5,      0x5 //Receive macro pointer-bit 5
        .equ   prot_mtptr_rxptr_6,      0x6 //Receive macro pointer-bit 6

        .equ   prot_mtptr_txptr_0,      0x8 //Transmit macro pointer-bit 0
        .equ   prot_mtptr_txptr_1,      0x9 //Transmit macro pointer-bit 1
        .equ   prot_mtptr_txptr_2,      0xa //Transmit macro pointer-bit 2
        .equ   prot_mtptr_txptr_3,      0xb //Transmit macro pointer-bit 3
        .equ   prot_mtptr_txptr_4,      0xc //Transmit macro pointer-bit 4
        .equ   prot_mtptr_txptr_5,      0xd //Transmit macro pointer-bit 5
        .equ   prot_mtptr_txptr_6,      0xe //Transmit macro pointer-bit 6

// bits of the Frame Table Base Address Register (FTBAR)
        .equ   prot_ftbar_ftba0_0,      0x0 //Frame table 0 base address-bit 0
        .equ   prot_ftbar_ftba0_1,      0x1 //Frame table 0 base address-bit 1
        .equ   prot_ftbar_ftba0_2,      0x2 //Frame table 0 base address-bit 2
        .equ   prot_ftbar_ftba0_3,      0x3 //Frame table 0 base address-bit 3
        .equ   prot_ftbar_ftba0_4,      0x4 //Frame table 0 base address-bit 4
        .equ   prot_ftbar_ftba0_5,      0x5 //Frame table 0 base address-bit 5
        .equ   prot_ftbar_ftba0_6,      0x6 //Frame table 0 base address-bit 6

        .equ   prot_ftbar_ftba1_0,      0x8 //Frame table 1 base address-bit 0
        .equ   prot_ftbar_ftba1_1,      0x9 //Frame table 1 base address-bit 1
        .equ   prot_ftbar_ftba1_2,      0xa //Frame table 1 base address-bit 2
        .equ   prot_ftbar_ftba1_3,      0xb //Frame table 1 base address-bit 3
        .equ   prot_ftbar_ftba1_4,      0xc //Frame table 1 base address-bit 4
        .equ   prot_ftbar_ftba1_5,      0xd //Frame table 1 base address-bit 5
        .equ   prot_ftbar_ftba1_6,      0xe //Frame table 1 base address-bit 6

// bits of the Receive/Transmit Base Address Register (RTBAR) (old names)
        .equ   prot_rtbar_rxba_0,       0x0 //Receive macro base address-bit 0
        .equ   prot_rtbar_rxba_1,       0x1 //Receive macro base address-bit 1
        .equ   prot_rtbar_rxba_2,       0x2 //Receive macro base address-bit 2
```

```
        .equ   prot_rtbar_rxba_3,       0x3 //Receive macro base address-bit 3
        .equ   prot_rtbar_rxba_4,       0x4 //Receive macro base address-bit 4
        .equ   prot_rtbar_rxba_5,       0x5 //Receive macro base address-bit 5
        .equ   prot_rtbar_rxba_6,       0x6 //Receive macro base address-bit 6

        .equ   prot_rtbar_txba_0,       0x8 //Transmit macro base address-bit 0
        .equ   prot_rtbar_txba_1,       0x9 //Transmit macro base address-bit 1
        .equ   prot_rtbar_txba_2,       0xa //Transmit macro base address-bit 2
        .equ   prot_rtbar_txba_3,       0xb //Transmit macro base address-bit 3
        .equ   prot_rtbar_txba_4,       0xc //Transmit macro base address-bit 4
        .equ   prot_rtbar_txba_5,       0xd //Transmit macro base address-bit 5
        .equ   prot_rtbar_txba_6,       0xe //Transmit macro base address-bit 6

// bits of the Macro Table Base Address Register (MTBAR) (NEW NAMES)
        .equ   prot_mtbar_rxba_0,       0x0 //Receive macro base address-bit 0
        .equ   prot_mtbar_rxba_1,       0x1 //Receive macro base address-bit 1
        .equ   prot_mtbar_rxba_2,       0x2 //Receive macro base address-bit 2
        .equ   prot_mtbar_rxba_3,       0x3 //Receive macro base address-bit 3
        .equ   prot_mtbar_rxba_4,       0x4 //Receive macro base address-bit 4
        .equ   prot_mtbar_rxba_5,       0x5 //Receive macro base address-bit 5
        .equ   prot_mtbar_rxba_6,       0x6 //Receive macro base address-bit 6

        .equ   prot_mtbar_txba_0,       0x8 //Transmit macro base address-bit 0
        .equ   prot_mtbar_txba_1,       0x9 //Transmit macro base address-bit 1
        .equ   prot_mtbar_txba_2,       0xa //Transmit macro base address-bit 2
        .equ   prot_mtbar_txba_3,       0xb //Transmit macro base address-bit 3
        .equ   prot_mtbar_txba_4,       0xc //Transmit macro base address-bit 4
        .equ   prot_mtbar_txba_5,       0xd //Transmit macro base address-bit 5
        .equ   prot_mtbar_txba_6,       0xe //Transmit macro base address-bit 6

// bits of the Delay Table Pointer (DTPTR)
        .equ   prot_dtptr_rdptr_0,      0x0 //Receive delay pointer-bit 0
        .equ   prot_dtptr_rdptr_1,      0x1 //Receive delay pointer-bit 1
        .equ   prot_dtptr_rdptr_2,      0x2 //Receive delay pointer-bit 2

        .equ   prot_dtptr_rdba_0,       0x3 //Receive delay base address-bit 0
        .equ   prot_dtptr_rdba_1,       0x4 //Receive delay base address-bit 1
        .equ   prot_dtptr_rdba_2,       0x5 //Receive delay base address-bit 2
        .equ   prot_dtptr_rdba_3,       0x6 //Receive delay base address-bit 3

        .equ   prot_dtptr_tdptr_0,      0x8 //Transmit delay pointer-bit 0
        .equ   prot_dtptr_tdptr_1,      0x9 //Transmit delay pointer-bit 1
        .equ   prot_dtptr_tdptr_2,      0xa //Transmit delay pointer-bit 2

        .equ   prot_dtptr_tdba_0,       0xb //Transmit delay base address-bit 0
        .equ   prot_dtptr_tdba_1,       0xc //Transmit delay base address-bit 1
        .equ   prot_dtptr_tdba_2,       0xd //Transmit delay base address-bit 2
        .equ   prot_dtptr_tdba_3,       0xe //Transmit delay base address-bit 3


//==================================================================
// UART equates
//==================================================================
```

```
// general definitions

        .equ  urx,                     0x00204000
        .equ  urx_20,                  0x00204020
        .equ  utx,                     0x00204040
        .equ  utx_60,                  0x00204060
        .equ  ucr1,                    0x00204080
        .equ  ucr2,                    0x00204082
        .equ  ubrg,                    0x00204084
        .equ  usr,                     0x00204086
        .equ  uts,                     0x00204088
        .equ  upcr,                    0x0020408a
        .equ  uddr,                    0x0020408c
        .equ  updr,                    0x0020408e

// bits of UART control register 1 (UCR1)
        .equ  ucr1_uarten,             0x0 // old name
        .equ  ucr1_uen,                0x0 // NEW NAME
        .equ  ucr1_doze,               0x1
        .equ  ucr1_sndbrk,             0x4
        .equ  ucr1_rtsden,             0x5 // old name
        .equ  ucr1_rtsdie,             0x5 // NEW NAME
        .equ  ucr1_txmptyen,           0x6 // old name
        .equ  ucr1_txeie,              0x6 // NEW NAME
        .equ  ucr1_iren,               0x7
        .equ  ucr1_rxen,               0x8
        .equ  ucr1_rrdyen,             0x9 // old name
        .equ  ucr1_rrdyie,             0x9 // NEW NAME
        .equ  ucr1_rxfl0,              0xa
        .equ  ucr1_rxfl1,              0xb
        .equ  ucr1_txen,               0xc
        .equ  ucr1_trdyen,             0xd // old name
        .equ  ucr1_trdyie,             0xd // NEW NAME
        .equ  ucr1_txfl0,              0xe
        .equ  ucr1_txfl1,              0xf

// bits of UART control register 2 (UCR2)
        .equ  ucr2_clksrc,             0x4
        .equ  ucr2_ws,                 0x5 // old name
        .equ  ucr2_chsz,               0x5 // NEW NAME
        .equ  ucr2_stpb,               0x6
        .equ  ucr2_proe,               0x7
        .equ  ucr2_pren,               0x8
        .equ  ucr2_cts,                0xc // old name
        .equ  ucr2_ctsd,               0xc // NEW NAME
        .equ  ucr2_CTSC,               0xD
        .equ  ucr2_irts,               0xe

// bits of UART status register (USR)
        .equ  usr_rtsd,                0x5
        .equ  usr_rrdy,                0x9
        .equ  usr_trdy,                0xd
        .equ  usr_rtss,                0xe
        .equ  usr_txmpty,              0xf // old name
```

```
        .equ  usr_txe,                  0xf // NEW NAME

// bits of the UART receiver register (URX)
        .equ  urx_prerr,                0xa
        .equ  urx_brk,                  0xb
        .equ  urx_frmerr,               0xc
        .equ  urx_ovrrun,               0xd
        .equ  urx_err,                  0xe
        .equ  urx_charrdy,              0xf

// bits of the UART test register (UTS)
        .equ  uts_loopir,               0xa
        .equ  uts_loop,                 0xc
        .equ  uts_frcperr,              0xd

// bits of the UART port control register (UPCR), old names
        .equ  upcr_pc0,                 0x0
        .equ  upcr_pc1,                 0x1
        .equ  upcr_pc2,                 0x2
        .equ  upcr_pc3,                 0x3

// bits of the UART port control register (UPCR), NEW NAMES
        .equ  upcr_upc0,                0x0
        .equ  upcr_upc1,                0x1
        .equ  upcr_upc2,                0x2
        .equ  upcr_upc3,                0x3

// bits of the UART data direction register (UDDR), old names
        .equ  uddr_pdc0,                0x0
        .equ  uddr_pdc1,                0x1
        .equ  uddr_pdc2,                0x2
        .equ  uddr_pdc3,                0x3

// bits of the UART data direction register (UDDR), NEW NAMES
        .equ  uddr_udd0,                0x0
        .equ  uddr_udd1,                0x1
        .equ  uddr_udd2,                0x2
        .equ  uddr_udd3,                0x3

// bits of the UART port data register (UPDR), old names
        .equ  updr_pd0,                 0x0
        .equ  updr_pd1,                 0x1
        .equ  updr_pd2,                 0x2
        .equ  updr_pd3,                 0x3

// bits of the UART port data register (UPDR), NEW NAMES
        .equ  updr_upd0,                0x0
        .equ  updr_upd1,                0x1
        .equ  updr_upd2,                0x2
        .equ  updr_upd3,                0x3


//=====================================================================
// QSPI equates
```

```
//=====================================================================

 // QSPI BASE ADDRESS
      .equ   qspi_base_address,                   0x00205000

// control ram,          split into 16-byte sections
      .equ   qspi_control_ram0_base_address,      0x00205000
      .equ   qspi_control_ram1_base_address,      0x00205010
      .equ   qspi_control_ram2_base_address,      0x00205020
      .equ   qspi_control_ram3_base_address,      0x00205030
      .equ   qspi_control_ram4_base_address,      0x00205040
      .equ   qspi_control_ram5_base_address,      0x00205050
      .equ   qspi_control_ram6_base_address,      0x00205060
      .equ   qspi_control_ram7_base_address,      0x00205070

// data ram,             split into 16-byte sections
      .equ   qspi_data_ram0_base_address,         0x00205400
      .equ   qspi_data_ram1_base_address,         0x00205410
      .equ   qspi_data_ram2_base_address,         0x00205420
      .equ   qspi_data_ram3_base_address,         0x00205430
      .equ   qspi_data_ram4_base_address,         0x00205440
      .equ   qspi_data_ram5_base_address,         0x00205450
      .equ   qspi_data_ram6_base_address,         0x00205460
      .equ   qspi_data_ram7_base_address,         0x00205470

// control register base addresses
      .equ   qspi_regs_base_address,              0x00205f00
      .equ   qspi_spsr_base_address,              0x00205f10
      .equ   qspi_trig_base_address,              0x00205ff8

 // QSPI REGISTERS ADDRESS relatitve to qspi_regs_base_address
      .equ   qspi_qpcr,        0x00
      .equ   qspi_qddr,        0x02
      .equ   qspi_qpdr,        0x04
      .equ   qspi_spcr,        0x06
      .equ   qspi_qcr0,        0x08
      .equ   qspi_qcr1,        0x0a
      .equ   qspi_qcr2,        0x0c
      .equ   qspi_qcr3,        0x0e
      .equ   qspi_spsr,        0x10
      .equ   qspi_sccr0,       0x12
      .equ   qspi_sccr1,       0x14
      .equ   qspi_sccr2,       0x16
      .equ   qspi_sccr3,       0x18
      .equ   qspi_sccr4,       0x1a

// QSPI REGISTERS ADDRESS relatitve to qspi_trig_base_address
      .equ   qspi_trigger0,    0x00
      .equ   qspi_trigger1,    0x02
      .equ   qspi_trigger2,    0x04
      .equ   qspi_trigger3,    0x06

 //BYTE ACCESS,          relative to qspi_regs_base_address
      .equ   qspi_qpcrb,       0x00
```

```
        .equ   qspi_qddrb,      0x02
        .equ   qspi_qpdrb,      0x04
        .equ   qspi_spcrb,      0x06
        .equ   qspi_qcr0b,      0x08
        .equ   qspi_qcr1b,      0x0a
        .equ   qspi_qcr2b,      0x0c
        .equ   qspi_qcr3b,      0x0e

//BYTE ACCESS,           relative to qspi_spsr_base_address
        .equ   qspi_spsrb,      0x00
        .equ   qspi_sccr0b,     0x02
        .equ   qspi_sccr1b,     0x04
        .equ   qspi_sccr2b,     0x06
        .equ   qspi_sccr3b,     0x08
        .equ   qspi_sccr4b,     0x0a

//BYTE ACCESS,           relative to qspi_trig_base_address
        .equ   qspi_trigger0b,  0x00
        .equ   qspi_trigger1b,  0x02
        .equ   qspi_trigger2b,  0x04
        .equ   qspi_trigger3b,  0x06

// QSPI QPCR BITS                           old names
        .equ   qspi_qpcr_pc0,   0
        .equ   qspi_qpcr_pc1,   1
        .equ   qspi_qpcr_pc2,   2
        .equ   qspi_qpcr_pc3,   3
        .equ   qspi_qpcr_pc4,   4
        .equ   qspi_qpcr_pc5,   5
        .equ   qspi_qpcr_pc6,   6
        .equ   qspi_qpcr_pc7,   7

// QSPI QPCR BITS                           NEW NAMES
        .equ   qspi_qpcr_qpc0,  0
        .equ   qspi_qpcr_qpc1,  1
        .equ   qspi_qpcr_qpc2,  2
        .equ   qspi_qpcr_qpc3,  3
        .equ   qspi_qpcr_qpc4,  4
        .equ   qspi_qpcr_qpc5,  5
        .equ   qspi_qpcr_qpc6,  6
        .equ   qspi_qpcr_qpc7,  7

 // QSPI QDDR BITS                          old names
        .equ   qspi_qddr_pd0,   0
        .equ   qspi_qddr_pd1,   1
        .equ   qspi_qddr_pd2,   2
        .equ   qspi_qddr_pd3,   3
        .equ   qspi_qddr_pd4,   4
        .equ   qspi_qddr_pd5,   5
        .equ   qspi_qddr_pd6,   6
        .equ   qspi_qddr_pd7,   7

 // QSPI QDDR BITS                          NEW NAMES
        .equ   qspi_qddr_qdd0,  0
```

```
        .equ   qspi_qddr_qdd1,   1
        .equ   qspi_qddr_qdd2,   2
        .equ   qspi_qddr_qdd3,   3
        .equ   qspi_qddr_qdd4,   4
        .equ   qspi_qddr_qdd5,   5
        .equ   qspi_qddr_qdd6,   6
        .equ   qspi_qddr_qdd7,   7

   // QSPI QPDR BITS // QSPI QDDR BITS   old names
        .equ   qspi_qpdr_d0,     0
        .equ   qspi_qpdr_d1,     1
        .equ   qspi_qpdr_d2,     2
        .equ   qspi_qpdr_d3,     3
        .equ   qspi_qpdr_d4,     4
        .equ   qspi_qpdr_d5,     5
        .equ   qspi_qpdr_d6,     6
        .equ   qspi_qpdr_d7,     7

   // QSPI QPDR BITS // QSPI QDDR BITS   NEW NAMES
        .equ   qspi_qpdr_qpd0,   0
        .equ   qspi_qpdr_qpd1,   1
        .equ   qspi_qpdr_qpd2,   2
        .equ   qspi_qpdr_qpd3,   3
        .equ   qspi_qpdr_qpd4,   4
        .equ   qspi_qpdr_qpd5,   5
        .equ   qspi_qpdr_qpd6,   6
        .equ   qspi_qpdr_qpd7,   7

   // QSPI SPCR BITS
        .equ   qspi_spcr_qspe,   0
        .equ   qspi_spcr_doze,   1
        .equ   qspi_spcr_halt,   2
        .equ   qspi_spcr_wie,    4
        .equ   qspi_spcr_trcie,  5
        .equ   qspi_spcr_hltie,  6
        .equ   qspi_spcr_qe0,    7
        .equ   qspi_spcr_qe1,    8
        .equ   qspi_spcr_qe2,    9
        .equ   qspi_spcr_qe3,    10
        .equ   qspi_spcr_cspol0, 11
        .equ   qspi_spcr_cspol1, 12
        .equ   qspi_spcr_cspol2, 13
        .equ   qspi_spcr_cspol3, 14
        .equ   qspi_spcr_cspol4, 15

   // QSPI QCRn BITS
        .equ   qspi_qcrn_qpn,    0x3f // queue pointer n bits mask
        .equ   qspi_qcrn_hmdn,   14
        .equ   qspi_qcrn_len,    15

  // QSPI QCR1 BITS
        .equ   qspi_qcr1_trcnt,  0x3c0 // trigger counter mask for queue 1

   // QSPI SPSR BITS
```

```
      .equ  qspi_spsr_eot0,  0
      .equ  qspi_spsr_eot1,  1
      .equ  qspi_spsr_eot2,  2
      .equ  qspi_spsr_eot3,  3
      .equ  qspi_spsr_qpwf,  4
      .equ  qspi_spsr_trc,   5
      .equ  qspi_spsr_halta, 6
      .equ  qspi_spsr_qa0,   8
      .equ  qspi_spsr_qa1,   9
      .equ  qspi_spsr_qa2,   10
      .equ  qspi_spsr_qa3,   11
      .equ  qspi_spsr_qx0,   12
      .equ  qspi_spsr_qx1,   13
      .equ  qspi_spsr_qx2,   14
      .equ  qspi_spsr_qx3,   15

 // QSPI SCCRn BITS
      .equ  qspi_sccrn_sckdfn,0x7f // sckdfn bits mask
      .equ  qspi_sccrn_csckdn,0x380 // csckdn bits mask
      .equ  qspi_sccrn_datrn,0x1c00 // datrn bits mask
      .equ  qspi_sccrn_lsbfn,13
      .equ  qspi_sccrn_ckpoln,14
      .equ  qspi_sccrn_cphan,15

//=======================================================================
// Timer/PWM
//=======================================================================

// base address
      .equ  tpwm_base_addr,         0x00206000 //MCU Timer base address

// register addresses relative to base
      .equ  tpwm_tpwcr,             0x0
      .equ  tpwm_tpwmr,             0x2
      .equ  tpwm_tpwsr,             0x4
      .equ  tpwm_twir,              0x6
      .equ  tpwm_tocr1,             0x8
      .equ  tpwm_tocr3,             0xa
      .equ  tpwm_tocr4,             0xc
      .equ  tpwm_ticr1,             0xe
      .equ  tpwm_ticr2,             0x10
      .equ  tpwm_pwor,              0x12
      .equ  tpwm_tcr,               0x14 // old name
      .equ  tpwm_tcnt,              0x14 // NEW NAME
      .equ  tpwm_pwcr,              0x16 // old name
      .equ  tpwm_pwml,              0x16 // NEW NAME
      .equ  tpwm_pwcnr,             0x18 // old name
      .equ  tpwm_pwcnt,             0x18 // NEW NAME

//tpwcr bits
      .equ  tpwm_tpwcr_pwdbg,       11 //TPWCR pwdbg bit
      .equ  tpwm_tpwcr_tdbg,        10 //TPWCR tdbg bit
      .equ  tpwm_tpwcr_pwd,         9 //TPWCR pwd bit
      .equ  tpwm_tpwcr_pwe,         8 //TPWCR pwe bit
```

```
      .equ   tpwm_tpwcr_td,          7 //TPWCR td bit
      .equ   tpwm_tpwcr_te,          6 //TPWCR te bit
      .equ   tpwm_tpwcr_pspw2,       5 //TPWCR pspw2 bit
      .equ   tpwm_tpwcr_pspw1,       4 //TPWCR pspw1 bit
      .equ   tpwm_tpwcr_pspw0,       3 //TPWCR pspw0 bit
      .equ   tpwm_tpwcr_pst2,        2 //TPWCR pst2 bit
      .equ   tpwm_tpwcr_pst1,        1 //TPWCR pst1 bit
      .equ   tpwm_tpwcr_pst0,        0 //TPWCR pst0 bit

//tpwmr bits
      .equ   tpwm_tpwmr_pwc,         14 //TPWMR pwc bit
      .equ   tpwm_tpwmr_pwp,         13 //TPWMR pwp bit
      .equ   tpwm_tpwmr_fo4,         12 //TPWMR fo4 bit
      .equ   tpwm_tpwmr_fo3,         11 //TPWMR fo3 bit
      .equ   tpwm_tpwmr_fo1,         10 //TPWMR fo1 bit
      .equ   tpwm_tpwmr_im21,        9 //TPWMR im21 bit
      .equ   tpwm_tpwmr_im20,        8 //TPWMR im20 bit
      .equ   tpwm_tpwmr_im11,        7 //TPWMR im11 bit
      .equ   tpwm_tpwmr_im10,        6 //TPWMR im10 bit
      .equ   tpwm_tpwmr_om41,        5 //TPWMR om41 bit
      .equ   tpwm_tpwmr_om40,        4 //TPWMR om40 bit
      .equ   tpwm_tpwmr_om31,        3 //TPWMR om31 bit
      .equ   tpwm_tpwmr_om30,        2 //TPWMR om30 bit
      .equ   tpwm_tpwmr_om11,        1 //TPWMR om11 bit
      .equ   tpwm_tpwmr_om10,        0 //TPWMR om10 bit

//tpwsr bits
      .equ   tpwm_tpwsr_pwo,         7 //TPWSR pwo bit
      .equ   tpwm_tpwsr_tov,         6 //TPWSR tov bit
      .equ   tpwm_tpwsr_pwf,         5 //TPWSR pwf bit
      .equ   tpwm_tpwsr_if2,         4 //TPWSR if2 bit
      .equ   tpwm_tpwsr_if1,         3 //TPWSR if1 bit
      .equ   tpwm_tpwsr_of4,         2 //TPWSR of4 bit
      .equ   tpwm_tpwsr_of3,         1 //TPWSR of3 bit
      .equ   tpwm_tpwsr_of1,         0 //TPWSR of1 bit

//twir bits
      .equ   tpwm_twir_pwoie,        7 //TWIR pwoie bit
      .equ   tpwm_twir_tovie,        6 //TWIR tovie bit
      .equ   tpwm_twir_pwfie,        5 //TWIR pwfie bit
      .equ   tpwm_twir_if2ie,        4 //TWIR if2ie bit
      .equ   tpwm_twir_if1ie,        3 //TWIR if1ie bit
      .equ   tpwm_twir_of4ie,        2 //TWIR of4ie bit
      .equ   tpwm_twir_of3ie,        1 //TWIR of3ie bit
      .equ   tpwm_twir_of1ie,        0 //TWIR of1ie bit


//=======================================================================
// ckctl, rsr, emddr, emdr, and gpcr registers
//=======================================================================

// register addresses
      .equ   ckctl,                  0x0020c000
      .equ   rsr,                    0x0020c400
```

```
        .equ   emddr,                 0x0020c800
        .equ   emdr,                  0x0020c802
        .equ   gpcr,                  0x0020cc00

// bits of CKCTL
        .equ   ckctl_ckihd,           0x0
        .equ   ckctl_mcs,             0x1
        .equ   ckctl_mcd0,            0x2
        .equ   ckctl_mcd1,            0x3
        .equ   ckctl_mcd2,            0x4
        .equ   ckctl_ckos,            0x5
        .equ   ckctl_ckoe,            0x6
        .equ   ckctl_ckohe,           0x7
        .equ   ckctl_dcs,             0x8

// bits of RSR
        .equ   rsr_exr,               0x0
        .equ   rsr_wdr,               0x1

// bits of EMDDR
        .equ   emddr_emdd0,           0x0
        .equ   emddr_emdd1,           0x1
        .equ   emddr_emdd2,           0x2
        .equ   emddr_emdd3,           0x3
        .equ   emddr_emdd4,           0x4
        .equ   emddr_emdd5,           0x5

// bits of EMDR
        .equ   emdr_emd0,             0x0
        .equ   emdr_emd1,             0x1
        .equ   emdr_emd2,             0x2
        .equ   emdr_emd3,             0x3
        .equ   emdr_emd4,             0x4
        .equ   emdr_emd5,             0x5

// bits of GPCR
        .equ   gpcr_gpc0,             0x0
        .equ   gpcr_gpc1,             0x1
        .equ   gpcr_gpc2,             0x2
        .equ   gpcr_gpc3,             0x3
        .equ   gpcr_gpc4,             0x4
        .equ   gpcr_gpc5,             0x5
        .equ   gpcr_gpc6,             0x6
        .equ   gpcr_gpc7,             0x7


//=============================================================================
// Keypad Port
//=============================================================================



        .equ   kpp_base_address,      0x0020a000 // Module Base Address
```

```
        .equ   kpp_kpcr,                0x0    // Port Control Register
        .equ   kpp_kpsr,                0x2    // Port Status Register
        .equ   kpp_kddr,                0x4    // Data direction Register
        .equ   kpp_kpdr,                0x6    // Data value Register


//==============================================================================
// Subscriber Interface Module (SmartCard Port)
//==============================================================================

// old names
        .equ   scp_base_address,0x0020b000// Module Base Address

        .equ   scp_simcr,       0x0       // SIM Control Register
        .equ   scp_siacr,       0x2       // SIM Activation Control Register
        .equ   scp_siicr,       0x4       // SIM Interrupt Control Register
        .equ   scp_simsr,       0x6       // SIM Status Register
        .equ   scp_simdr,       0x8       // SIM Tx and Rx Data Register
        .equ   scp_sipcr,       0xa       // SIM Pins Control Register

// NEW NAMES
        .equ   scp_base_address,0x0020b000// Module Base Address

        .equ   scp_scpcr,       0x0       // SCP Control Register
        .equ   scp_scacr,       0x2       // SCP Activation Control Register
        .equ   scp_scpier,      0x4       // SCP Interrupt Control Register
        .equ   scp_scpsr,       0x6       // SCP Status Register
        .equ   scp_scpdr,       0x8       // SCP Tx and Rx Data Register
        .equ   scp_scppcr,      0xa       // SCP Pins Control Register


//==============================================================================
// External Interrupts
//==============================================================================

        .equ   wext_base_address,0x00209000    // Module Base Address

        .equ   wext_eppar,              0x0    // Edge Port Pin Assignment Register
        .equ   wext_epddr,              0x2    // Edge Port Data Direction Register
        .equ   wext_epdr,               0x4    // Edge Port Data Register
        .equ   wext_epfr,               0x6    // Edge Port Flag Register


//==============================================================================
// EIM
//==============================================================================

        .equ   eim_registers_base_address,0x00201000    // eim base address

// register addresses relative to base address
        .equ   eim_cs0_control_reg,     0x0
        .equ   eim_cs1_control_reg,     0x4
        .equ   eim_cs2_control_reg,     0x8
        .equ   eim_cs3_control_reg,     0xc
```

```
        .equ   eim_cs4_control_reg,     0x10
        .equ   eim_cs5_control_reg,     0x14
        .equ   eim_configuration_reg,   0x18

// Bits definitions for the EIM CS configuration registers
        .equ   eim_cs_csen,      0x0     // Chip select enable
        .equ   eim_cs_pa,        0x1     // Output value for CS0 only when csen = 0
        .equ   eim_cs_wp,        0x2     // Write Protec
        .equ   eim_cs_sp,        0x3     // Supervisor Protect
        .equ   eim_cs_dsz,       0x4     // Data Port Size
        .equ   eim_cs_ebc,       0x6     // Enable Byte Control
        .equ   eim_cs_wen,       0x7     // Determines when EB0-1 outputs are
                                            negated during a write cycle.
        .equ   eim_cs_oea,       0x8     // Determines when OE is asserted during a
                                            read cycle.
        .equ   eim_cs_csa,       0x9     // Chip Select Assert
        .equ   eim_cs_edc,       0xa     // Extra Dead Cycle
        .equ   eim_cs_wws,       0xb     // Write Wait-State
        .equ   eim_cs_wsc,       0xc     //Wait-State Control


// EIM configuration register bits definitions
        .equ   eim_cr_shen,             0x0
        .equ   eim_cr_hdb,              0x2
        .equ   eim_cr_sprom,            0x3
        .equ   eim_cr_spram,            0x4
        .equ   eim_cr_spiper,           0x5
        .equ   eim_cr_epen,             0x6

//=========================================================================
// Peripheral Interrupt Controller
//=========================================================================
        .equ   pic_base_address,        0x00200000
        .equ   pic_isr ,                0x0
        .equ   pic_normal_enable,       0x4
        .equ   pic_fast_enable ,        0x8
        .equ   pic_normal_pending,      0xc
        .equ   pic_fast_pending,        0x10
        .equ   pic_control ,            0x14

// Bits in the PIC registers
        .equ   pic_sw0 ,                0x0
        .equ   pic_sw1 ,                0x1
        .equ   pic_sw2 ,                0x2
        .equ   pic_int0 ,               0x5
        .equ   pic_int1 ,               0x6
        .equ   pic_int2 ,               0x7
        .equ   pic_int3 ,               0x8
        .equ   pic_int4 ,               0x9
        .equ   pic_int5 ,               0xa
        .equ   pic_int6 ,               0xb
        .equ   pic_int7 ,               0xc
        .equ   pic_urts ,               0xd
        .equ   pic_kpd ,                0xe
```

```
        .equ   pic_pit ,               0x10
        .equ   pic_tpw ,               0x11
        .equ   pic_sim ,               0x16
        .equ   pic_mdi ,               0x17
        .equ   pic_qspi ,              0x18
        .equ   pic_prot ,              0x19
        .equ   pic_prot0 ,             0x1a
        .equ   pic_prot1 ,             0x1b
        .equ   pic_prot2 ,             0x1c
        .equ   pic_utx ,               0x1d
        .equ   pic_smpdint ,           0x13   // old name
        .equ   pic_smpd ,              0x13   // NEW NAME
        .equ   pic_urx ,               0x1f


//======================================================================
// Watchdog Timer
//======================================================================
        .equ   wdt_base_address,       0x00208000

        .equ   wdt_wcr,                0x0
        .equ   wdt_wsr,                0x2

//======================================================================
// Periodic Interrupt Timer
//======================================================================
        .equ   pit_base_address,       0x00207000

        .equ   pit_itcsr,              0x0 // old names
        .equ   pit_itdr,               0x2
        .equ   pit_itadr,              0x4

        .equ   pit_pitcsr,             0x0 // NEW NAMES
        .equ   pit_pitml,              0x2
        .equ   pit_pitcnt,             0x4

//======================================================================
// PSR bits
//======================================================================
        .equ   psr_tc ,                0xc
        .equ   psr_sc ,                0xa
        .equ   psr_mm ,                0x9
        .equ   psr_ee ,                0x8
        .equ   psr_ic ,                0x7
        .equ   psr_ie ,                0x6
        .equ   psr_fe ,                0x4
        .equ   psr_af ,                0x1
        .equ   psr_c ,                 0x0
```

B.2 MCU Include File

```
/*
 * DSP56651/DSP56652 C include file for M.CORE
 *
 * Revision History:
 * 1.0: may 28,         1998
 */

#ifndef _REDCAP_H_
#define _REDCAP_H_

/* *********************
 REDCAP MCU MEMORY MAP
 ********************* */

/* On-chip ROM: 16 KB starting at location 0 */
#define REDCAP_MCU_ROM_BASE 0x00000000
#define REDCAP_MCU_ROM_SIZE 0x00004000

/* On-chip RAM: 2KB starting at specified location */
#define REDCAP_MCU_RAM_BASE 0x00100000
#define REDCAP_MCU_RAM_SIZE x00000800

/* On-chip peripherals - base addresses */
#define REDCAP_MCU_PIC 0x00200000 /* Interrupt Controller */
#define REDCAP_MCU_EIM 0x00201000 /* External Interface Module */
#define REDCAP_MCU_MDI 0x00202000 /* MCU-DSP Interface */
#define REDCAP_MCU_PROT 0x00203000 /* Protocol Timer */
#define REDCAP_MCU_UART 0x00204000 /* UART */
#define REDCAP_MCU_QSPI 0x00205000 /* Queued SPI */
#define REDCAP_MCU_PWM 0x00206000 /* PWM/Input Capture Timers */
#define REDCAP_MCU_PIT 0x00207000 /* Periodic Interrupt Timer */
#define REDCAP_MCU_WDT 0x00208000 /* Watchdog Timer */
#define REDCAP_MCU_INTPINS 0x00209000 /* Interrupt Pins Control */
#define REDCAP_MCU_KPP 0x0020A000 /* Keypad Port */
#define REDCAP_MCU_SCP 0x0020B000 /* Smart Card Port */
#define REDCAP_MCU_CKCTL 0x0020C000 /* Clock Control Register */
#define REDCAP_MCU_RSR 0x0020C400 /* Reset Source Register */
#define REDCAP_MCU_EMULPORT 0x0020C800 /* Emulation Port Control */
#define REDCAP_MCU_GPCR 0x0020CC00 /* General Port Control */

/* Reserved 0x00300000 through 03ffffff */

/* External memory associated with chip Selects */
#define REDCAP_MCU_CS0_BASE 0x40000000 /* Chip Select 0 */
#define REDCAP_MCU_CS0_SIZE 0x01000000
#define REDCAP_MCU_CS1_BASE 0x41000000 /* Chip Select 1 */
#define REDCAP_MCU_CS1_SIZE 0x01000000
#define REDCAP_MCU_CS2_BASE 0x42000000 /* Chip Select 2 */
#define REDCAP_MCU_CS2_SIZE 0x01000000
#define REDCAP_MCU_CS3_BASE 0x43000000 /* Chip Select 3 */
#define REDCAP_MCU_CS3_SIZE 0x01000000
#define REDCAP_MCU_CS4_BASE 0x44000000 /* Chip Select 4 */
```

```
#define REDCAP_MCU_CS4_SIZE 0x01000000
#define REDCAP_MCU_CS5_BASE 0x45000000 /* Chip Select 5 */
#define REDCAP_MCU_CS5_SIZE 0x01000000


/* *****************************
 REDCAP Clock Control Register
 example usage:
 unsigned short *clock = (unsigned short *)REDCAP_MCU_CKCTL;
 ***************************** */

#define REDCAP_CKCTL_CKIHD 0x0001
#define REDCAP_CKCTL_MCS 0x0002
#define REDCAP_CKCTL_MCD_1 0x0000
#define REDCAP_CKCTL_MCD_2 0x0004
#define REDCAP_CKCTL_MCD_4 0x0008
#define REDCAP_CKCTL_MCD_8 0x000C
#define REDCAP_CKCTL_MCD_16 0x0010
#define REDCAP_CKCTL_CKOS 0x0020
#define REDCAP_CKCTL_CKOE 0x0040
#define REDCAP_CKCTL_CKOHE 0x0080
#define REDCAP_CKCTL_DCS 0x0100


/* *********************
 REDCAP Reset Source Register
 example usage:
 unsigned short *reset_source= (unsigned short *)REDCAP_MCU_RSR;
 ********************* */

#define REDCAP_RSR_EXR 0x0001
#define REDCAP_RSR_WDR 0x0002


/* *********************
 REDCAP Emulation Port Control
 example usage:
 struct redcap_emulport *em_port= (struct redcap_emulport*)REDCAP_MCU_EMPORT;
 ********************* */

#ifdef _ASSEM
#define EMU_DIR 0
#define EMU_DATA 2
#else
struct redcap_emulport {
 unsigned short emddr; /* em port data direction register */
 volatile unsigned short emdr; /* em port data register */
};
#endif


/* *********************
 REDCAP General Port Control
 example usage:
```

```
 unsigned short *gpcr= (unsigned short *)REDCAP_MCU_GPCR;
 ********************* */


/* *********************
 REDCAP External Interface Module
 example usage:
 struct redcap_eim *eim= (struct redcap_eim*)REDCAP_MCU_EIM;
 ********************* */

#ifdef _ASSEM
#define EIM_CS0CR 0x0
#define EIM_CS1CR 0x4
#define EIM_CS2CR 0x8
#define EIM_CS3CR 0xc
#define EIM_CS4CR 0x10
#define EIM_CS5CR 0x14
#define EIM_CR 0x18
#else
struct redcap_eim {
 unsigned long cs0cr; /* chip select 0 control register */
 unsigned long cs1cr; /* chip select 0 control register */
 unsigned long cs2cr; /* chip select 0 control register */
 unsigned long cs3cr; /* chip select 0 control register */
 unsigned long cs4cr; /* chip select 0 control register */
 unsigned long cs5cr; /* chip select 0 control register */
 unsigned long eimcr; /* eim configuration register */
};
#endif


/* *********************
 REDCAP Interrupt controller
 example usage:
 struct redcap_pic *pic= (struct redcap_pic *)REDCAP_MCU_PIC;
 ********************* */

#ifdef _ASSEM
#define PIC_ISR 0x00
#define PIC_NIER 0x04
#define PIC_FIER 0x08
#define PIC_NIPR 0x0C
#define PIC_FIPR 0x10
#define PIC_ICR 0x14
#else
struct redcap_pic {
 volatile unsigned long isr; /* interrupt source register*/
 unsigned long nier; /* normal interrupt enable register*/
 unsigned long fier; /* fast interrupt enable register*/
 volatile unsigned long nipr; /* normal interrupt pending register*/
 volatile unsigned long fipr; /* fast interrupt pending register*/
 unsigned long icr; /* interrupt control register*/
};
#endif
```

```
/* Bit masks which apply to isr, nier, nipr, fier, and fipr. */
#define REDCAP_INT_URX 0x80000000
#define REDCAP_INT_SMPD 0x40000000
#define REDCAP_INT_UTX 0x20000000
#define REDCAP_INT_PT2 0x10000000
#define REDCAP_INT_PT1 0x08000000
#define REDCAP_INT_PT0 0x04000000
#define REDCAP_INT_PTM 0x02000000
#define REDCAP_INT_QSPI 0x01000000
#define REDCAP_INT_MDI 0x00800000
#define REDCAP_INT_SIM 0x00400000
#define REDCAP_INT_TPW 0x00020000
#define REDCAP_INT_PIT 0x00010000
#define REDCAP_INT_KPD 0x00004000
#define REDCAP_INT_URTS 0x00002000
#define REDCAP_INT_INT7 0x00001000
#define REDCAP_INT_INT6 0x00000800
#define REDCAP_INT_INT5 0x00000400
#define REDCAP_INT_INT4 0x00000200
#define REDCAP_INT_INT3 0x00000100
#define REDCAP_INT_INT2 0x00000080
#define REDCAP_INT_INT1 0x00000040
#define REDCAP_INT_INT0 0x00000020
#define REDCAP_INT_S2 0x00000004
#define REDCAP_INT_S1 0x00000002
#define REDCAP_INT_S0 0x00000001

/* icr manipulation */
#define REDCAP_ICR_ENABLE 0x00008000
#define REDCAP_ICR_MAKE_SRC(x) (((x)&0x1f)<<7)
#define REDCAP_ICR_MAKE_VECTOR(x) ((x)&0x7f)
#define REDCAP_ICR_GET_SRC(x) (((x)>>7)&0x1f)
#define REDCAP_ICR_GET_VECTOR(x) ((x)&0x7f)

/* ***************************************************************
 REDCAP MCU-DSP Interface
 example usage:
 unsigned short *mdi_shared = (unsigned short *)MDI_SHARED_BASE;
 struct redcap_mdi_regs *mdi_regs = (struct redcap_mdi_regs*)MDI_REG_BASE;
 *************************************************************** */
/* Shared memory */
#define MDI_SHARED_BASE (REDCAP_MCU_MDI+0x000)
/* Registers are at the end of the 1K space */
#define MDI_REG_BASE (REDCAP_MCU_MDI+0xFF2)

#ifdef _ASSEM
#define MDI_MCVR 0x2
#define MDI_MCR 0x4
#define MDI_MSR 0x6
#define MDI_MTR1 0x8
#define MDI_MTR0 0xa
#define MDI_MRR1 0xc
#define MDI_MRR0 0xe
```

```
#else
struct redcap_mdi_regs {
 volatile unsigned short mcvr; /* command vector register,volatile MC bit */
 volatile unsigned short mcr; /* control register,volatile MDIR bit */
 volatile unsigned short msr; /* status register*/
 unsigned short mtr1; /* transmit register 1 */
 unsigned short mtr0; /* transmit register 0*/
 volatile unsigned short mrr1; /* receive register 1 - read-only */
 volatile unsigned short mrr0; /* receive register 0 - read-only */
};
#endif

/* mcvr register bits */
#define MCVR_MNMI 0x0001
#define MCVR_MCV0 0x0002
#define MCVR_MCV1 0x0004
#define MCVR_MCV2 0x0008
#define MCVR_MCV3 0x0010
#define MCVR_MCV4 0x0020
#define MCVR_MCV5 0x0040
#define MCVR_MCV6 0x0080
#define MCVR_MC 0x0100
/* mcr register bits */
#define MCR_MF0 0x0001
#define MCR_MF1 0x0002
#define MCR_MF2 0x0004
#define MCR_MDIR 0x0040
#define MCR_DHR 0x0080
#define MCR_MGIE1 0x0400
#define MCR_MGIE0 0x0800
#define MCR_MTIE1 0x1000
#define MCR_MTIE0 0x2000
#define MCR_MRIE1 0x4000
#define MCR_MRIE0 0x8000
/* msr register bits */
#define MSR_MF0 0x0001
#define MSR_MF1 0x0002
#define MSR_MF2 0x0004
#define MSR_MEP 0x0010
#define MSR_DPM 0x0020
#define MSR_MSMP 0x0040
#define MSR_DRS 0x0080
#define MSR_DWS 0x0100
#define MSR_MGIP1 0x0400
#define MSR_MGIP2 0x0800
#define MSR_MTE1 0x1000
#define MSR_MTE0 0x2000
#define MSR_MRF1 0x4000
#define MSR_MRF0 0x8000

/****************************
 REDCAP Keypad Port (KPP).
 example usage:
 struct redcap_kppb *kppb= (struct redcap_kppb *)REDCAP_MCU_KPP;
```

```
 **************************** */
#ifdef _ASSEM
#define KPP_KPCR 0x0
#define KPP_KPSR 0x2
#define KPP_KDDR 0x4
#define KPP_KPDR 0x6
#else
/* this structure uses halfwords */
struct redcap_kpp {
 unsigned short kpcr; /* keypad control reg*/
 volatile unsigned short kpsr; /* keypad status reg */
 unsigned short kddr; /* keypad data dir reg */
 volatile unsigned short kpdr; /* keypad data reg */
};
/* this structure uses byte addressing */
struct redcap_kppb {
 unsigned char kpcr_col; /* keypad control reg - cols*/
 unsigned char kpcr_row; /* keypad control reg - rows*/
 unsigned char reserved; /* byte not used*/
 volatile unsigned char kpsr; /* keypad status reg */
 unsigned char kddr_col; /* keypad data dir reg - cols*/
 unsigned char kddr_row; /* keypad data dir reg - rows*/
 volatile unsigned char kpdr_col; /* keypad data reg - cols*/
 volatile unsigned char kpdr_row; /* keypad data reg - rows*/
};
#endif

/* kpsr register bits */
#define KPSR_KPKD 0x0001

/******************************
 * REDCAP Timer/PWM (TPWM).
 example usage:
 struct redcap_tpwm *twpm= (struct redcap_tpwm*)REDCAP_MCU_TPWM;
 ***************************** */

#ifdef _ASSEM
#define TPWM_TPWCR 0x00
#define TPWM_TPWMR 0x02
#define TPWM_TPWSR 0x04
#define TPWM_TWIR 0x06
#define TPWM_TOCR1 0x08
#define TPWM_TOCR3 0x0A
#define TPWM_TOCR4 0x0C
#define TPWM_TICR1 0x0E
#define TPWM_TICR2 0x10
#define TPWM_PWOR 0x12
#define TPWM_TCR 0x14
#define TPWM_PWCR 0x16
#define TPWM_PWCNR 0x18
#else
struct redcap_tpwm {
 unsigned short tpwcr; /* control reg*/
 unsigned short tpwmr; /* mode reg */
```

```
 volatile unsigned short tpwsr; /* status reg*/
 unsigned short twir; /* interrupts enable reg */
 unsigned short tocr1; /* timer output compare 1*/
 unsigned short tocr3; /* timer output compare 3*/
 unsigned short tocr4; /* timer output compare 4*/
 volatile unsigned short ticr1; /* timer input capture 1,read-only*/
 volatile unsigned short ticr2; /* input capture 2*/
 unsigned short pwor; /* pwm output compare */
 volatile unsigned short tcr; /* timer counter register,read-only*/
 unsigned short pwcr; /* pwm count register*/
 volatile unsigned short pwcnr; /* pwm counter register,read-only*/
};
#endif

/* tpwcr register bits */
#define TPWCR_TE 0x0040
/* tpwmsr register bits */
#define TPWSR_OF1 0x0001

/* ********************************
 REDCAP Periodic Interrupt Timer
 example usage:
 struct redcap_pit *pit= (struct redcap_pit *)REDCAP_MCU_PIT;
 ******************************** */

#ifdef _ASSEM
#define PIT_ITCSR 0
#define PIT_ITDR 2
#define PIT_ITADR 4
#else
struct redcap_pit{
 volatile unsigned short itcsr; /* control and status register */
 unsigned short itdr; /* data register (determines modulo) */
 volatile unsigned short itadr; /* alternate data register,read-only */
};
#endif

/* ********************************
 REDCAP Watchdog Timer
 example usage:
 struct redcap_wdt *wdt= (struct redcap_wdt *)REDCAP_MCU_WDT;
 ******************************** */

#ifdef _ASSEM
#define WDT_WCR 0
#define WDT_WSR 2
#else
struct redcap_wdt{
 unsigned short wcr; /* watchdog control register */
 unsigned short wsr; /* watchdog service register */
};
#endif

/* ********************************
```

```
 REDCAP Interrupt Pins
 example usage:
 struct redcap_intpins *intpins= (struct redcap_intpins *)REDCAP_MCU_INTPINS;
 ******************************** */

#ifdef _ASSEM
#define INTPINS_EPPAR 0
#define INTPINS_EPDDR 2
#define INTPINS_EPDR 4
#define INTPINS_EPFR 6
#else
struct redcap_intpins{
 unsigned short eppar; /* pin assignment register */
 unsigned short epddr; /* data direction register */
 volatile unsigned short epdr; /* data register */
 volatile unsigned short epfr; /* flag register */
};
#endif

/* ********************************
 REDCAP Smart Cart Port
 example usage:
 struct redcap_scp *scp= (struct redcap_scp *)REDCAP_MCU_SCP;
 ******************************** */

#ifdef _ASSEM
#define SCP_SIMCR 0x0
#define SCP_SIACR 0x2
#define SCP_SIICR 0x4
#define SCP_SIMSR 0x6
#define SCP_SIMDR 0x8
#deinfe SCP_SIPCR 0xA

#else
struct redcap_intpins{
 unsigned short simcr; /* control register */
 unsigned short siacr; /* activation control register */
 unsigned short siicr; /* interrupt control register */
 volatile unsigned short simsr; /* status register */
 volatile unsigned short simdr; /* transmit and receive data register */
 volatile unsigned short sipcr; /* pins control register */
};
#endif

/* ********************************
 REDCAP UART
 note: rx and tx registers are "short" (halfwords) but are lie on
 "long" (word) boundaries to support the use of ldm/stm instructions.
 example usage:
 unsigned long *uart_rx_data = (unsigned long *)UART_R_REG;
 unsigned long *uart_tx_data = (unsigned long *)UART_T_REG;
 struct redcap_uart_ctrl *uart_ctrl= (struct redcap_uart_ctrl *)UART_C_REG;
 ******************************** */
```

```
/* receive data registers */
#define UART_R_REG (REDCAP_MCU_UART+0x00)
/* transmit data registers */
#define UART_T_REG (REDCAP_MCU_UART+0x40)
/* Control Registers */
#define UART_C_REG (REDCAP_MCU_UART+0x80)

#ifdef _ASSEM
#define UART_UCR1 0x0
#define UART_UCR2 0x2
#define UART_UBRG 0x4
#define UART_USR 0x6
#define UART_UTS 0x8
#define UART_UPCR 0xa
#define UART_UDDR 0xc
#define UART_UPDR 0xe
#else
struct redcap_uart_ctrl{
 unsigned short ucr1; /* control register 1 */
 unsigned short ucr2; /* control register 2 */
 unsigned short ubrg; /* baud-rate-generator register */
 volatile unsigned short usr; /* status register */
 unsigned short uts; /* test register */
 unsigned short upcr; /* port control register */
 unsigned short uddr; /* port data direction register */
 volatile unsigned short updr; /* port data register */
};
#endif

/* ********************************
 REDCAP QSPI
 example usage:
 unsigned short *qspi_control_ram = (unsigned short *)QSPI_C_RAM;
 unsigned short *qspi_data_ram = (unsigned short *)QSPI_D_RAM;
 struct redcap_qspi_c_reg *qspi_ctrl = (struct redcap_qspi_c_reg*)QSPI_C_REG;
 struct redcap_qspi_t_reg *qspi_trigs = (struct
redcap_qspi_t_reg*)QSPIDCAP_T_REG;
 ******************************** */

/* control ram */
#define QSPI_C_RAM (REDCAP_MCU_QSPI+0x000)
/* data ram */
#define QSPI_D_RAM (REDCAP_MCU_QSPI+0x400)
/* Control Registers */
#define QSPI_C_REG (REDCAP_MCU_UART+0xf00)
/* Manual Trigger Registers */
#define QSPI_T_REG (REDCAP_MCU_UART+0xff8)

#ifdef _ASSEM
/* control registers */
#define QSPI_QPCR 0x00
#define QSPI_QDDR 0x02
#define QSPI_QPDR 0x04
#define QSPI_SPCR 0x06
```

```
#define QSPI_QCR0 0x08
#define QSPI_QCR1 0x0a
#define QSPI_QCR2 0x0c
#define QSPI_QCR3 0x0e
#define QSPI_SPSR 0x10
#define QSPI_SCCR0 0x12
#define QSPI_SCCR1 0x14
#define QSPI_SCCR2 0x16
#define QSPI_SCCR3 0x18
#define QSPI_SCCR4 0x1a
/* trigger registers */
#define QSPI_TRIG0 0x0
#define QSPI_TRIG1 0x2
#define QSPI_TRIG2 0x4
#define QSPI_TRIG3 0x6
#else
struct redcap_qspi_c_reg{
 unsigned short qpcr; /* port control register */
 unsigned short qddr; /* port data direction register */
 volatile unsigned short qpdr /* port data register */
 unsigned short spcr; /* spi control register */
 volatile unsigned short qcr0; /* queue control register 0 */
 volatile unsigned short qcr1; /* queue control register 1 */
 volatile unsigned short qcr2; /* queue control register 2 */
 volatile unsigned short qcr3; /* queue control register 3 */
 volatile unsigned short spsr; /* spi status register */
 unsigned short sccr0; /* serial channel control register 0 */
 unsigned short sccr1; /* serial channel control register 1 */
 unsigned short sccr2; /* serial channel control register 2 */
 unsigned short sccr3; /* serial channel control register 3 */
 unsigned short sccr4; /* serial channel control register 4 */
};
struct redcap_qspi_t_reg{
 unsigned short trig0; /* trigger for queue 0 */
 unsigned short trig1; /* trigger for queue 1 */
 unsigned short trig2; /* trigger for queue 2 */
 unsigned short trig3; /* trigger for queue 3 */
};
#endif

/* ********************************
 REDCAP Protocol Timer
 example usage:
 unsigned short *event_table = (unsigned short *)PROT_ET_BASE;
 struct redcap_prot_ctrl *prot_ctrl = (struct redcap_prot_ctrl *)PROT_C_REG_BASE;
 ******************************** */

/* event table base address */
#define PROT_ET_BASE (REDCAP_MCU_PROT+0x000)
/* control registers base address*/
#define PROT_C_REG_BASE (REDCAP_MCU_PROT+0x800)

#ifdef _ASSEM
#define PROT_TCTR 0x00
```

```
#define PROT_TIER 0x02
#define PROT_TSTR 0x04
#define PROT_TEVR 0x06
#define PROT_TIPR 0x08
#define PROT_CTIC 0x0A
#define PROT_CTIPR 0x0C
#define PROT_CFC 0x0E
#define PROT_CFPR 0x10
#define PROT_RSC 0x12
#define PROT_RSPR 0x14
#define PROT_PDPAR 0x16
#define PROT_PDDR 0x18
#define PROT_PDDAT 0x1A
#define PROT_FTPTR 0x1C
#define PROT_RTPTR 0x1E
#define PROT_FTBAR 0x20
#define PROT_RTBAR 0x22
#define PROT_DTPTR 0x24
#else
struct redcap_prot_ctrl{
 unsigned short tctr; /* timer control register */
 unsigned short tier; /* timer interrupt enable register */
 volatile unsigned short tstr; /* timer status register */
 volatile unsigned short tevr; /* timer event register */
 unsigned short tipr; /* time interval prescaler register */
 volatile unsigned short ctic; /* channel time internal counter */
 unsigned short ctipr; /* channel time interval preload regiser */
 volatile unsigned short cfc; /* channel frame counter */
 unsigned short cfpr; /* channel frame preload register */
 volatile unsigned short rsc; /* reference slot counter */
 unsigned short rspr; /* reference slot preload register */
 unsigned short pdpar; /* port d pin assignment register */
 unsigned short pddr; /* port d direction register */
 volatile unsigned short pddat; /* port d data register */
 volatile unsigned short ftptr; /* frame table pointer register */
 volatile unsigned short rtptr; /* receive/transmit macro tables pointer regis-
ter*/
 unsigned short ftbar; /* frame table base address register */
 unsigned short rtbar; /* receive/tranmit macro tables base address register */
 volatile unsigned short dtptr; /* delay table pointer register */
};
#endif

#endif
```

B.3 DSP Equates

```
;****************************************************************
; DSP Equates for DSP56651/DSP56652
```

```
;
; Revision History:
; 1.0: may 28 1998
;***************************************************************


; Register Addresses for IPR register

M_IPRC EQU $FFFF ; Interrupt Priority Register Core
M_IPRP EQU $FFFE ; Interrupt Priority Register Peripheral

; Register Addresses of PLL

M_PCTL0 EQU $FFFD ; PLL Control Register 0
M_PCTL1 EQU $FFFC ; PLL Control Register 1

; PLL Control Register 0 (PCTL0)

M_MF EQU $0FFF ; Multiplication Factor Bits Mask (MF0-MF11)
M_PD EQU $F000 ; PreDivider Factor Bits Mask (PD3-PD0)
M_PD03 EQU $F000 ; PreDivider Factor Bits Mask (PD3-PD0)

; PLL Control Register 1 (PCTL1)

M_PD46 EQU $0E00 ; PreDivider Factor Bits Mask (PD6-PD4)
M_DF EQU $7 ; Division Factor Bits Mask (DF0-DF2)
M_XTLR EQU 3 ; XTAL Range select bit
M_XTLD EQU 4 ; XTAL Disable Bit
M_PSTP EQU 5 ; STOP Processing State Bit
M_PEN EQU 6 ; PLL Enable Bit
M_PCOD EQU 7 ; PLL Clock Output Disable Bit


; Register Addresses Of BIU

M_BCR EQU $FFFA ; Bus Control Register <-- not used in this device
M_IDR EQU $FFF9 ; ID Register

; Register Addresses Of PATCH

M_PA0 EQU $FFF8 ; Patch Address Register 0
M_PA1 EQU $FFF7 ; Patch Address Register 1
M_PA2 EQU $FFF6 ; Patch Address Register 2
M_PA3 EQU $FFF5 ; Patch Address Register 3

; Register Addresses Of BPMR

M_BPMRG EQU $FFF4 ; BPMRG Register
M_BPMRL EQU $FFF3 ; BPMRL Register
M_BPMRH EQU $FFF2 ; BPMRH Register


;-----------------------------------------------------------------------
;
```

```
; EQUATES for SR and OMR
;
;------------------------------------------------------------------------

; control and status bits in SR

M_C EQU 0 ; Carry
M_V EQU 1 ; Overflow
M_Z EQU 2 ; Zero
M_N EQU 3 ; Negative
M_U EQU 4 ; Unnormalized
M_E EQU 5 ; Extension
M_L EQU 6 ; Limit
M_S EQU 7 ; Scaling Bit
M_I0 EQU 8 ; Interupt Mask Bit 0
M_I1 EQU 9 ; Interupt Mask Bit 1
M_S0 EQU 10 ; Scaling Mode Bit 0
M_S1 EQU 11 ; Scaling Mode Bit 1
M_FV EQU 12 ; DO-Forever Flag
M_SM EQU 13 ; Arithmetic Saturation
M_RM EQU 14 ; Rounding Mode
M_LF EQU 15 ; DO-Loop Flag

; control and status bits in OMR

M_MA EQU 0 ; Operating Mode A
M_MB EQU 1 ; Operating Mode B
M_MC EQU 2 ; Operating Mode C
M_MD EQU 3 ; Operating Mode D
M_EBD EQU 4 ; External Bus Disable bit in OMR
M_PCD EQU 5 ; PC relative logic disable
M_SD EQU 6 ; Stop Delay
M_XYS EQU 8 ; Stack Extention space select
M_EUN EQU 9 ; Extended Stack Underflow Flag
M_EOV EQU 10 ; Extended Stack Overflow Flag
M_WRP EQU 11 ; Extended Stack Wrap Flag
M_SEN EQU 12 ; Stack Extended Enable
M_ATE EQU 15 ; Address Tracing Enable bit in OMR.


;------------------------------------------------------------------------
;
; EQUATES for MDI
;
;------------------------------------------------------------------------

MDI_SHARED_MEMORY_BASE equ $1c00

MDI_IO_BASE equ $ff80
MDR_IRQ_BASE equ $60

; WMDI DSP-side registers

DRR0 equ MDI_IO_BASE+$f ;DSP-side receive register 0
```

```
DRR1 equ MDI_IO_BASE+$e ;DSP-side receive register 1
DTR0 equ MDI_IO_BASE+$d ;DSP-side transmit register 0
DTR1 equ MDI_IO_BASE+$c ;DSP-side transmit register 1
DSR equ MDI_IO_BASE+$b ;DSP-side status register
DCR equ MDI_IO_BASE+$a ;DSP-side control register

; WMDI DSP-side Status Register (DSR) bits

DF0 equ 0 ;DSP-side Flag 0
DF1 equ 1 ;DSP-side Flag 1
DF2 equ 2 ;DSP-side Flag 2
DEP equ 4 ;DSP Event Pending
MPM0 equ 5 ;MCU Power Mode bit 0
MPM1 equ 6 ;MCU Power Mode bit 1
DWSC equ 7 ;DSP Wake from Stop interrupt Clear
MCP equ 8 ;MCU Command Pending
DTIC equ 9 ;DSP Protocol Timer Interrupt clear
DGIR1 equ 10 ;DSP General Interrupt Request 1 bit
DGIR0 equ 11 ;DSP General Interrupt Request 0 bit
DRF1 equ 12 ;DSP Receive register 1 Full
DRF0 equ 13 ;DSP Receive register 0 Full
DTE1 equ 14 ;DSP Transmit register 1 Empty
DTE0 equ 15 ;DSP Transmit register 0 Empty

; WMDI DSP-side Control Register (DCR) bits

DMF0 equ 0 ;DSP-side MCU messaging flag 0
DMF1 equ 1 ;DSP-side MCU messaging flag 1
DMF2 equ 2 ;DSP-side MCU messaging flag 2
MCIE equ 8 ;MCU Command Interrupt Enable
DRIE1 equ 12 ;DSP Recieve 1 Interrupt Enable
DRIE0 equ 13 ;DSP Recieve 0 Interrupt Enable
DTIE1 equ 14 ;DSP Transmit 1 Interrupt Enable
DTIE0 equ 15 ;DSP Transmit 0 Interrupt Enable


;------------------------------------------------------------------------
;
; EQUATES for Base Band Port (BBP)
;
;------------------------------------------------------------------------

;
; Register Addresses of BBP

BBP_PCRB EQU $FFAF ; BBP Port Control Register
BBP_PRRB EQU $FFAE ; BBP GPIO Direction Register
BBP_PDRB EQU $FFAD ; BBP GPIO Data Register
BBP_TXB EQU $FFAC ; BBP Transmit Data Register
BBP_TSRB EQU $FFAB ; BBP Time Slot Register
BBP_RXB EQU $FFAA ; BBP Receive Data Register
BBP_SSISRB EQU $FFA9 ; BBP Status Register
BBP_CRCB EQU $FFA8 ; BBP Control Register C
BBP_CRBB EQU $FFA7 ; BBP Control Register B
```

```
BBP_CRAB EQU $FFA6 ; BBP Control Register A
BBP_TCRB EQU $FFA5 ; BBP Tran. Frame Preload counter
BBP_RCRB EQU $FFA4 ; BBP Rec. Frame Preload counter


; BBP Control Register A Bit Flags

BBP_PSR EQU 15 ; Prescaler Range
BBP_DC EQU $1F00 ; Frame Rate Divider Control Mask (DC0-DC7)
BBP_WL EQU $6000 ; Word Length Control Mask (WL0-WL7)

; BBP Control register B Bit Flags

BBP_OF EQU $3 ; Serial Output Flag Mask
BBP_OF0 EQU 0 ; Serial Output Flag 0
BBP_OF1 EQU 1 ; Serial Output Flag 1
BBP_TCE EQU 4 ; BBP Tr Frame Cnt enable
BBP_RCE EQU 5 ; BBP Rc Frame Cnt enable
BBP_TCIE EQU 6 ; BBP Tr Frame RO enable
BBP_RCIE EQU 7 ; BBP Rc Frame RO enable
BBP_TE EQU 8 ; BBP Transmit Enable
BBP_RE EQU 9 ; BBP Receive Enable
BBP_TIE EQU 10 ; BBP Transmit Interrupt Enable
BBP_RIE EQU 11 ; BBP Receive Interrupt Enable
BBP_TLIE EQU 12 ; BBP Transmit Last Slot Interrupt Enable
BBP_RLIE EQU 13 ; BBP Receive Last Slot Interrupt Enable
BBP_TEIE EQU 14 ; BBP Transmit Error Interrupt Enable
BBP_REIE EQU 15 ; BBP Receive Error Interrupt Enable

; BBP Control Register C Bit Flags

BBP_SYN EQU 0 ; Sync/Async Control
BBP_MOD EQU 1 ; BBP Mode Select
BBP_SCD EQU $1C ; Serial Control Direction Mask
BBP_SCD0 EQU 2 ; Serial Control 0 Direction
BBP_SCD1 EQU 3 ; Serial Control 1 Direction
BBP_SCD2 EQU 4 ; Serial Control 2 Direction
BBP_SCKD EQU 5 ; Clock Source Direction
BBP_CKP EQU 6 ; Clock Polarity
BBP_SHFD EQU 7 ; Shift Direction
BBP_FSL EQU $3000 ; Frame Sync Length Mask (FSL0-FSL1)
BBP_FSL0 EQU 12 ; Frame Sync Length 0
BBP_FSL1 EQU 13 ; Frame Sync Length 1
BBP_FSR EQU 14 ; Frame Sync Relative Timing
BBP_FSP EQU 15 ; Frame Sync Polarity

; BBP Status Register Bit Flags

BBP_IF EQU $3 ; Serial Input Flag Mask
BBP_IF0 EQU 0 ; Serial Input Flag 0
BBP_IF1 EQU 1 ; Serial Input Flag 1
BBP_TFS EQU 2 ; Transmit Frame Sync Flag
BBP_RFS EQU 3 ; Receive Frame Sync Flag
BBP_TUE EQU 4 ; Transmitter Underrun Error FLag
```

```
BBP_ROE EQU 5 ; Receiver Overrun Error Flag
BBP_TDE EQU 6 ; Transmit Data Register Empty
BBP_RDF EQU 7 ; Receive Data Register Full



;------------------------------------------------------------------------
;
; EQUATES for Serial Audio Port (SAP)
;
;------------------------------------------------------------------------


; Register Addresses Of SAP

SAP_PCRA EQU $FFBF ; SAP Port Control Register
SAP_PRRA EQU $FFBE ; SAP GPIO Direction Register
SAP_PDRA EQU $FFBD ; SAP GPIO Data Register
SAP_TXA EQU $FFBC ; SAP Transmit Data Register
SAP_TSRA EQU $FFBB ; SAP Time Slot Register
SAP_RXA EQU $FFBA ; SAP Receive Data Register
SAP_SSISRA EQU $FFB9 ; SAP Status Register
SAP_CRCA EQU $FFB8 ; SAP Control Register C
SAP_CRBA EQU $FFB7 ; SAP Control Register B
SAP_CRAA EQU $FFB6 ; SAP Control Register A
SAP_TCLR EQU $FFB5 ; SAP Timer Preload register
SAP_TCRA EQU $FFB4 ; SAP Timer count register


; SAP Control Register A Bit Flags

SAP_PSR EQU 15 ; Prescaler Range
SAP_DC EQU $1F00 ; Frame Rate Divider Control Mask (DC0-DC7)
SAP_WL EQU $6000 ; Word Length Control Mask (WL0-WL7)

; SAP Control register B Bit Flags

SAP_OF EQU $3 ; Serial Output Flag Mask
SAP_OF0 EQU 0 ; Serial Output Flag 0
SAP_OF1 EQU 1 ; Serial Output Flag 1
SAP_TCE EQU 2 ; SAP Timer enable
SAP_TE EQU 8 ; SAP Transmit Enable
SAP_RE EQU 9 ; SAP Receive Enable
SAP_TIE EQU 10 ; SAP Transmit Interrupt Enable
SAP_RIE EQU 11 ; SAP Receive Interrupt Enable
SAP_TLIE EQU 12 ; SAP Transmit Last Slot Interrupt Enable
SAP_RLIE EQU 13 ; SAP Receive Last Slot Interrupt Enable
SAP_TEIE EQU 14 ; SAP Transmit Error Interrupt Enable
SAP_REIE EQU 15 ; SAP Receive Error Interrupt Enable

; SAP Control Register C Bit Flags

SAP_SYN EQU 0 ; Sync/Async Control
SAP_MOD EQU 1 ; SAP Mode Select
```

```
SAP_SCD EQU $1C ; Serial Control Direction Mask
SAP_SCD0 EQU 2 ; Serial Control 0 Direction
SAP_SCD1 EQU 3 ; Serial Control 1 Direction
SAP_SCD2 EQU 4 ; Serial Control 2 Direction
SAP_SCKD EQU 5 ; Clock Source Direction
SAP_CKP EQU 6 ; Clock Polarity
SAP_SHFD EQU 7 ; Shift Direction
SAP_BRM EQU 8 ; Binary Rate Multiplier (BRM) enable
SAP_FSL EQU $3000 ; Frame Sync Length Mask (FSL0-FSL1)
SAP_FSL0 EQU 12 ; Frame Sync Length 0
SAP_FSL1 EQU 13 ; Frame Sync Length 1
SAP_FSR EQU 14 ; Frame Sync Relative Timing
SAP_FSP EQU 15 ; Frame Sync Polarity

; SAP Status Register Bit Flags

SAP_IF EQU $3 ; Serial Input Flag Mask
SAP_IF0 EQU 0 ; Serial Input Flag 0
SAP_IF1 EQU 1 ; Serial Input Flag 1
SAP_TFS EQU 2 ; Transmit Frame Sync Flag
SAP_RFS EQU 3 ; Receive Frame Sync Flag
SAP_TUE EQU 4 ; Transmitter Underrun Error FLag
SAP_ROE EQU 5 ; Receiver Overrun Error Flag
SAP_TDE EQU 6 ; Transmit Data Register Empty
SAP_RDF EQU 7 ; Receive Data Register Full

;------------------------------------------------------------------------
;
; EQUATES for Exception Processing
;
;------------------------------------------------------------------------


 if @DEF(I_VEC)
 ; leave user definition as it is.
 else
I_VEC equ $0
 endif

;------------------------------------------------------------------------
; Non-Maskable interrupts
;------------------------------------------------------------------------
I_RESET EQU I_VEC+$00 ; Hardware RESET
I_STACK EQU I_VEC+$02 ; Stack Error
I_ILL EQU I_VEC+$04 ; Illegal Instruction
I_DBG EQU I_VEC+$06 ; Debug Request
I_TRAP EQU I_VEC+$08 ; Trap

;------------------------------------------------------------------------
; Interrupt Request Pins
;------------------------------------------------------------------------
I_IRQA EQU I_VEC+$10 ; IRQA
I_IRQB EQU I_VEC+$12 ; IRQB - from DSP_IRQ pin
I_IRQC EQU I_VEC+$14 ; IRQC - from MDI wake up from stop
```

```
I_IRQD EQU I_VEC+$16 ; IRQD - from Protocol Timer wake from stop

;-----------------------------------------------------------------------
; Protocol Timer Interrupts
;-----------------------------------------------------------------------
I_PT_CVR0 EQU I_VEC+$20 ; Protocol Timer CVR0
I_PT_CVR1 EQU I_VEC+$22 ; Protocol Timer CVR1
I_PT_CVR2 EQU I_VEC+$24 ; Protocol Timer CVR2
I_PT_CVR3 EQU I_VEC+$26 ; Protocol Timer CVR3
I_PT_CVR4 EQU I_VEC+$28 ; Protocol Timer CVR4
I_PT_CVR5 EQU I_VEC+$2A ; Protocol Timer CVR5
I_PT_CVR6 EQU I_VEC+$2C ; Protocol Timer CVR6
I_PT_CVR7 EQU I_VEC+$2E ; Protocol Timer CVR7
I_PT_CVR8 EQU I_VEC+$30 ; Protocol Timer CVR8
I_PT_CVR9 EQU I_VEC+$32 ; Protocol Timer CVR9
I_PT_CVR10 EQU I_VEC+$34 ; Protocol Timer CVR10
I_PT_CVR11 EQU I_VEC+$36 ; Protocol Timer CVR11
I_PT_CVR12 EQU I_VEC+$38 ; Protocol Timer CVR12
I_PT_CVR13 EQU I_VEC+$3A ; Protocol Timer CVR13
I_PT_CVR14 EQU I_VEC+$3C ; Protocol Timer CVR14
I_PT_CVR15 EQU I_VEC+$3E ; Protocol Timer CVR15

;-----------------------------------------------------------------------
; SAP Interrupts
;-----------------------------------------------------------------------
I_SAP_RD EQU I_VEC+$40 ; SAP Receive Data
I_SAP_RDE EQU I_VEC+$42 ; SAP Receive Data With Exception Status
I_SAP_RLS EQU I_VEC+$44 ; SAP Receive last slot
I_SAP_TD EQU I_VEC+$46 ; SAP Transmit data
I_SAP_TDE EQU I_VEC+$48 ; SAP Transmit Data With Exception Status
I_SAP_TLS EQU I_VEC+$4A ; SAP Transmit last slot
I_SAP_TRO EQU I_VEC+$4C ; SAP Timer counter roll-over

;-----------------------------------------------------------------------
; BBP Interrupts
;-----------------------------------------------------------------------
I_BBP_RD EQU I_VEC+$50 ; BBP Receive Data
I_BBP_RDE EQU I_VEC+$52 ; BBP Receive Data With Exception Status
I_BBP_RLS EQU I_VEC+$54 ; BBP Receive last slot
I_BBP_RRO EQU I_VEC+$56 ; BBP Receive Frame rolls over
I_BBP_TD EQU I_VEC+$58 ; BBP Transmit data
I_BBP_TDE EQU I_VEC+$5A ; BBP Transmit Data With Exception Status
I_BBP_TLS EQU I_VEC+$5C ; BBP Transmit last slot
I_BBP_TRO EQU I_VEC+$5E ; BBP Transmit Frame rolls over


;-----------------------------------------------------------------------
; MDI DSP-side interrupts
;-----------------------------------------------------------------------

I_MDI_MCU EQU I_VEC+$60 ; MDI MCU default command vector
I_MDI_RR0 EQU I_VEC+$62 ; MDI Receive Register 0 interrupt
I_MDI_RR1 EQU I_VEC+$64 ; MDI Receive Register 1 interrupt
I_MDI_TR0 EQU I_VEC+$66 ; MDI Transmit Register 0 interrupt
```

```
I_MDI_TR1 EQU I_VEC+$68 ; MDI Transmit Register 1 interrupt

;------------------------------------------------------------------------
; INTERRUPT ENDING ADDRESS
;------------------------------------------------------------------------
I_INTEND EQU I_VEC+$FF ; last address of interrupt vector space
```

Appendix C
Boundary Scan Register

This appendix provides detailed information on the Boundary Scan Register (BSR), including bit descriptions and the Boundary Scan Description Language (BSDL) listing for the DSP56652 in the 196-pin Plastic Ball Grid Array (PBGA) package.

C.1 BSR Bit Definitions

Table C-1 is a list of the BSR bit definitions.

Table C-1. BSR Bit Definitions

Bit #	Pin Name	Pin Type	Cell Type	Bit #	Pin Name	Pin Type	Cell Type
0	DSP_DE	-	control	40	COLUMN4	-	control
1	DSP_DE	input/output	data	41	COLUMN4	input/output	data
2	ROW7	-	control	42	COLUMN3	-	control
3	ROW7	input/output	data	43	COLUMN3	input/output	data
4	ROW6	-	control	44	COLUMN2	-	control
5	ROW6	input/output	data	45	COLUMN2	input/output	data
6	ROW5	-	control	46	COLUMN1	-	control
7	ROW5	input/output	data	47	COLUMN1	input/output	data
8	ROW4	-	control	48	COLUMN0	-	control
9	ROW4	input/output	data	49	COLUMN0	input/output	data
10	ROW3	-	control	50	STO	output	data
11	ROW3	input/output	data	51	RESET_IN	input	data
12	ROW2	-	control	52	RESET_OUT	output	data
13	ROW2	input/output	data	53	BMODE	input	data
14	ROW1	-	control	54	SIMRESET	-	control
15	ROW1	input/output	data	55	SIMRESET	input/output	data
16	ROW0	-	control	56	SENSE	-	control
17	ROW0	input/output	data	57	SENSE	input/output	data
18	INT7	-	control	58	SIMDATA	-	control
19	INT7	input/output	data	59	SIMDATA	input/output	data
20	INT6	-	control	60	PWR_EN	-	control
21	INT6	-	control	61	PWR_EN	input/output	data
22	INT5	-	control	62	SIMCLK	-	control
23	INT5	input/output	data	63	SIMCLK	input/output	data
24	INT4	-	control	64	DATA15	input/output	data
25	INT4	input/output	data	65	DATA14	input/output	data
26	INT3	-	control	66	DATA13	input/output	data
27	INT3	input/output	data	67	DATA12	input/output	data
28	INT2	-	control	68	DATA11	input/output	data
29	INT2	input/output	data	69	DATA10	input/output	data
30	INT1	-	control	70	DATA9	input/output	data
31	INT1	input/output	data	71	DATA8	input/output	data
32	INT0	-	control	72	DATA[15:8]	-	control
33	INT0	input/output	data	73	DATA[7:0]	-	control
34	COLUMN7	-	control	74	DATA7	input/output	data
35	COLUMN7	input/output	data	75	DATA6	input/output	data
36	COLUMN6	-	control	76	DATA5	input/output	data
37	COLUMN6	input/output	data	77	DATA4	input/output	data
38	COLUMN5	-	control	78	DATA3	input/output	data
39	COLUMN5	input/output	data	79	DATA2	input/output	data

Table C-1. BSR Bit Definitions

Bit #	Pin Name	Pin Type	Cell Type	Bit #	Pin Name	Pin Type	Cell Type
80	DATA1	input/output	data	120	ADDR21	output	data
81	DATA0	input/output	data	121	TOUT0	-	control
82	CS5	output	data	122	TOUT0	input/output	data
83	CS4	output	data	123	TOUT1	-	control
84	CS3	output	data	124	TOUT1	input/output	data
85	CS2	output	data	125	TOUT2	-	control
86	R/$\overline{W}$	-	control	126	TOUT2	input/output	data
87	$\overline{EB0}$,$\overline{EB1}$	-	control	127	TOUT3	-	control
88	CS1	output	data	128	TOUT3	input/output	data
89	CS0	output	data	129	TOUT4	-	control
90	R/W	input/output	data	130	TOUT4	input/output	data
91	OE	output	data	131	TOUT5	-	control
92	CKO	output	data	132	TOUT5	input/output	data
96	CKOH	output	data	133	TOUT6	-	control
94	CKIL	input	data	134	TOUT6	input/output	data
95	EB0	input/output	data	135	TOUT7	-	control
96	EB1	input/output	data	136	TOUT7	input/output	data
97	ADDR0	input/output	data	137	SPICS4	-	control
98	ADDR1	input/output	data	138	SPICS4	input/output	data
99	ADDR2	input/output	data	139	SPICS3	-	control
100	ADDR3	input/output	data	140	SPICS3	input/output	data
101	ADDR[7:0]	-	control	141	SPICS2	-	control
102	ADDR4	input/output	data	142	SPICS2	input/output	data
103	ADDR5	input/output	data	143	SPICS1	-	control
104	ADDR6	input/output	data	144	SPICS1	input/output	data
105	ADDR7	input/output	data	145	SPICS0	-	control
106	ADDR8	input/output	data	146	SPICS0	input/output	data
107	ADDR9	input/output	data	147	SCK	-	control
108	ADDR10	input/output	data	148	SCK	input/output	data
109	ADDR11	input/output	data	149	MISO	-	control
110	ADDR12	input/output	data	150	MISO	input/output	data
111	ADDR13	input/output	data	151	MOSI	-	control
112	ADDR14	input/output	data	152	MOSI	input/output	data
113	ADDR15	input/output	data	153	DSP_IRQ	input	data
114	ADDR[19:8]	-	control	154	SCKB	-	control
115	ADDR16	input/output	data	155	SCKB	input/output	data
116	ADDR17	input/output	data	156	SCB0	-	control
117	ADDR18	input/output	data	157	SCB0	input/output	data
118	ADDR19	input/output	data	158	SCB1	-	control
119	ADDR20	output	data	159	SCB1	input/output	data

Table C-1. BSR Bit Definitions

Bit #	Pin Name	Pin Type	Cell Type	Bit #	Pin Name	Pin Type	Cell Type
160	SCB2	-	control	179	PSTAT3	input/output	data
161	SCB2	input/output	data	180	PSTAT2	-	control
162	SRDB	-	control	181	PSTAT2	input/output	data
163	SRDB	input/output	data	182	PSTAT1	-	control
164	STDB	-	control	183	PSTAT1	input/output	data
165	STDB	input/output	data	184	PSTAT0	-	control
166	SCA2	-	control	185	PSTAT0	input/output	data
167	SCA2	input/output	data	186	SIZ1	-	control
168	SCA1	-	control	187	SIZ1	input/output	data
169	SCA1	input/output	data	188	SIZ0	-	control
170	SCA0	-	control	189	SIZ0	input/output	data
171	SCA0	input/output	data	190	CTS	-	control
172	SCKA	-	control	191	CTS	input/output	data
173	SCKA	input/output	data	192	RTS	-	control
174	SRDA	-	control	193	RTS	input/output	data
175	SRDA	input/output	data	194	RX	-	control
176	STDA	-	control	195	RX	input/output	data
177	STDA	input/output	data	196	TX	-	control
178	PSTAT3	-	control	197	TX	input/output	data

C.2 Boundary Scan Description Language

The following is a listing of the DSP56652 Boundary Scan Description Language.

```
-- 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: Sun Feb 23 11:09:20 1997
--
-- Revision History:
--

entity DSP56652 is
        generic (PHYSICAL_PIN_MAP : string := "PBGA196");

        port (   TRST_B:        in      bit;
                    TCK:        in      bit;
                    TMS:        in      bit;
                    TDI:        in      bit;
```

```
                TDO:        out     bit;
               ADDR:        inout   bit_vector(0 to 19);
               DATA:        inout   bit_vector(0 to 15);
               RW_B:        inout   bit;
               EB_B:        inout   bit_vector(0 to 1);
               OE_B:        buffer  bit;
                INT:        inout   bit_vector(0 to 7);
          DSP_IRQ_B:        in      bit;
               CS_B:        buffer  bit_vector(0 to 4);
               CKIH:        linkage bit;
               CKIL:        in      bit;
                CKO:        buffer  bit;
               CKOH:        buffer  bit;
              BMODE:        in      bit;
        RESET_OUT_B:        buffer  bit;
         RESET_IN_B:        in      bit;
                STO:        buffer  bit;
            MUX_CTL:        linkage bit;
               PCAP:        linkage bit;
               PVCC:        linkage bit;
               PGND:        linkage bit;
              P1GND:        linkage bit;
                SIZ:        inout   bit_vector(0 to 1);
              PSTAT:        inout   bit_vector(0 to 3);
           MCU_DE_B:        linkage bit;
           DSP_DE_B:        inout   bit;
               TEST:        linkage bit;
             COLUMN:        inout   bit_vector(0 to 7);
                ROW:        inout   bit_vector(0 to 7);
TX:         inout   bit;
                 RX:        inout   bit;
              RTS_B:        inout   bit;
              CTS_B:        inout   bit;
               TOUT:        inout   bit_vector(0 to 7);
               STDA:        inout   bit;
               SRDA:        inout   bit;
               SCKA:        inout   bit;
                SCA:        inout   bit_vector(0 to 2);
               STDB:        inout   bit;
               SRDB:        inout   bit;
               SCKB:        inout   bit;
                SCB:        inout   bit_vector(0 to 2);
               MOSI:        inout   bit;
               MISO:        inout   bit;
                SCK:        inout   bit;
              SPICS:        inout   bit_vector(0 to 4);
             SIMCLK:        inout   bit;
              SENSE:        inout   bit;
            SIMDATA:        inout   bit;
         SIMRESET_B:        inout   bit;
             PWR_EN:        inout   bit;
               AVDD:        linkage bit_vector(0 to 1);
               AGND:        linkage bit_vector(0 to 1);
               CVDD:        linkage bit;
               CGND:        linkage bit;
               DVDD:        linkage bit;
```

```
                    DGND:       linkage bit;
                    EVDD:       linkage bit;
                    EGND:       linkage bit;
                    FVDD:       linkage bit;
                    FGND:       linkage bit;
                    GVDD:       linkage bit_vector(0 to 1);
                    GGND:       linkage bit_vector(0 to 1);
                    HVDD:       linkage bit;
                    HGND:       linkage bit;
                    KVDD:       linkage bit;
                    KGND:       linkage bit;
                    BVDD:       linkage bit;
                    BGND:       linkage bit;
                    QVCC:       linkage bit_vector(0 to 3);
                   QVCCH:       linkage bit_vector(0 to 3);
                    QGND:       linkage bit_vector(0 to 3);
                     CS5:       buffer  bit;
                RESERVED:       linkage bit;
                  ADDR20:       buffer  bit;
                  ADDR21:       buffer  bit);

        use STD_1149_1_1994.all;
attribute COMPONENT_CONFORMANCE of DSP56652: entity is
        "STD_1149_1_1993";   -- complies with Std. 1149.1a-1993

 attribute PIN_MAP of DSP56652 : entity is PHYSICAL_PIN_MAP;
constant PBGA196 : PIN_MAP_STRING :=
                "ADDR20:        A2, " &
                "TOUT:         (A3, C4, B4, A4, D5, C5, A5, B5), " &
                "SPICS:        (E7, B6, E6, D6, A6), " &
                "HGND:          A7, " &
                "QVCCH:        (A8, G12, H5, P7), " &
                "DSP_IRQ_B:     A9, " &
                "SRDB:          A10, " &
                "EGND:          A11, " &
                "SRDA:          A12, " &
                "STDA:          A13, " &
                "AGND:         (B1, G2), " &
                "ADDR:         (G1, H3, G5, G4, G3, F5, F4, F2, E1, F3, E4, E3, E2,
D1, D4, D2, D3, C2, B2, C3), " &
                "ADDR21:        B3, " &
                "QVCC:         (B7, H1, J14, M8), " &
                "MOSI:          B8, " &
                "SCB:          (E9, D9, B9), " &
                "SCA:          (B10, C10, D10), " &
                "SCKA:          B11, " &
                "PSTAT:        (C13, B13, B12, C11), " &
                "KGND:          B14, " &
                "AVDD:         (C1, F1), " &
                "HVDD:          C6, " &
                "QGND:         (C7, H12, J2, N8), " &
                "SCKB:          C8, " &
                "STDB:          C9, " &
                "KVDD:          C12, " &
                "SIZ:          (D12, C14), " &
                "SCK:           D7, " &
```

```
                "MISO:          D8, " &
                "EVDD:          D11, " &
                "MUX_CTL:       D13, " &
                "CTS_B:         D14, " &
                "RESERVED:      E8, " &
                "RTS_B:         E11, " &
                "RX:            E12, " &
                "TEST:          E13, " &
                "TX:            E14, " &
                "TDO:           F10, " &
                "TCK:           F11, " &
                "DSP_DE_B:      F12, " &
                "TDI:           F13, " &
                "TRST_B:        F14, " &
                "MCU_DE_B:      G10, " &
"ROW:          (K14, J13, J11, J10, H13, H14, G13, G11), " &
 "TMS:           G14, " &
                "EB_B:         (H4, H2), " &
                "GGND:         (H10, L13), " &
                "GVDD:         (H11, L11), " &
                "FGND:          J1, " &
                "CKIH:          J3, " &
                "CKOH:          J4, " &
                "CKIL:          J5, " &
                "INT:          (L12, N14, M14, L14, K13, K12, K11, J12), " &
                "CKO:           K1, " &
                "FVDD:          K2, " &
                "OE_B:          K3, " &
                "RW_B:          K4, " &
                "DATA:         (N3, M4, P2, P3, N4, L4, P4, N5, M6, P5, N6, L6, K6,
M7, P6, N7), " &
                "PWR_EN:        K7, " &
                "BGND:          K8, " &
                "PVCC:          K9, " &
                "CS_B:         (L1, L2, M2, N1, M3), " &
                "CVDD:          L3, " &
                "DGND:          L5, " &
                "SIMCLK:        L7, " &
                "BVDD:          L8, " &
                "PCAP:          L9, " &
                "RESET_IN_B:    L10, " &
                "CGND:          M1, " &
                "DVDD:          M5, " &
                "SIMDATA:       M9, " &
                "RESET_OUT_B:   M10, " &
                "COLUMN:       (N11, M11, P12, N12, P13, M12, N13, M13), " &
                "CS5:           N2, " &
                "SIMRESET_B:    N9, " &
                "P1GND:         N10, " &
                "SENSE:         P8, " &
                "PGND:          P9, " &
                "BMODE:         P10, " &
                "STO:           P11 ";

        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 TRST_B : signal is true;
        attribute TAP_SCAN_CLOCK of    TCK : signal is (20.0e6, BOTH);

        attribute INSTRUCTION_LENGTH of DSP56652 : entity is 4;

        attribute INSTRUCTION_OPCODE of DSP56652 : entity is
           "EXTEST              (0000)," &
"SAMPLE              (0001)," &
           "IDCODE              (0010)," &
           "CLAMP               (0101)," &
           "HIGHZ               (0100)," &
           "ENABLE_MCU_ONCE     (0011)," &
           "ENABLE_DSP_ONCE     (0110)," &
           "DSP_DEBUG_REQUEST   (0111)," &
           "BYPASS              (1111, 1000, 1001, 1010, 1011, 1100, 1101, 1110)";

        attribute INSTRUCTION_CAPTURE of DSP56652 : entity is "0001";
        attribute IDCODE_REGISTER   of DSP56652 : entity is
           "0000"          & -- version
           "000110"        & -- manufacturer's use
           "0001000010"    & -- sequence number
           "00000001110"   & -- manufacturer identity
           "1";              -- 1149.1 requirement


        attribute REGISTER_ACCESS of DSP56652 : entity is
           "BYPASS   (ENABLE_MCU_ONCE,ENABLE_DSP_ONCE,DSP_DEBUG_REQUEST)" ;

        attribute BOUNDARY_LENGTH of DSP56652 : entity is 198;

        attribute BOUNDARY_REGISTER of DSP56652 : entity is
        -- num    cell   port         func     safe [ccell dis rslt]
           "0     (BC_1, *,            control,  1)," &
           "1     (BC_6, DSP_DE_B,     bidir,    X,   0,   1,   Z)," &
           "2     (BC_1, *,            control,  1)," &
           "3     (BC_6, ROW(7),       bidir,    X,   2,   1,   Z)," &
           "4     (BC_1, *,            control,  1)," &
           "5     (BC_6, ROW(6),       bidir,    X,   4,   1,   Z)," &
           "6     (BC_1, *,            control,  1)," &
           "7     (BC_6, ROW(5),       bidir,    X,   6,   1,   Z)," &
           "8     (BC_1, *,            control,  1)," &
           "9     (BC_6, ROW(4),       bidir,    X,   8,   1,   Z)," &
           "10    (BC_1, *,            control,  1)," &
           "11    (BC_6, ROW(3),       bidir,    X,   10,  1,   Z)," &
           "12    (BC_1, *,            control,  1)," &
           "13    (BC_6, ROW(2),       bidir,    X,   12,  1,   Z)," &
           "14    (BC_1, *,            control,  1)," &
           "15    (BC_6, ROW(1),       bidir,    X,   14,  1,   Z)," &
           "16    (BC_1, *,            control,  1)," &
           "17    (BC_6, ROW(0),       bidir,    X,   16,  1,   Z)," &
           "18    (BC_1, *,            control,  1)," &
           "19    (BC_6, INT(7),       bidir,    X,   18,  1,   Z)," &
        -- num    cell   port         func     safe [ccell dis rslt]
           "20    (BC_1, *,            control,  1)," &
           "21    (BC_6, INT(6),       bidir,    X,   20,  1,   Z)," &
```

```
           "22    (BC_1, *,             control,  1)," &
"23    (BC_6, INT(5),       bidir,     X,   22,  1,   Z)," &
           "24    (BC_1, *,             control,  1)," &
           "25    (BC_6, INT(4),        bidir,    X,   24,  1,   Z)," &
           "26    (BC_1, *,             control,  1)," &
           "27    (BC_6, INT(3),        bidir,    X,   26,  1,   Z)," &
           "28    (BC_1, *,             control,  1)," &
           "29    (BC_6, INT(2),        bidir,    X,   28,  1,   Z)," &
           "30    (BC_1, *,             control,  1)," &
           "31    (BC_6, INT(1),        bidir,    X,   30,  1,   Z)," &
           "32    (BC_1, *,             control,  1)," &
           "33    (BC_6, INT(0),        bidir,    X,   32,  1,   Z)," &
           "34    (BC_1, *,             control,  1)," &
           "35    (BC_6, COLUMN(7),     bidir,    X,   34,  1,   Z)," &
           "36    (BC_1, *,             control,  1)," &
           "37    (BC_6, COLUMN(6),     bidir,    X,   36,  1,   Z)," &
           "38    (BC_1, *,             control,  1)," &
           "39    (BC_6, COLUMN(5),     bidir,    X,   38,  1,   Z)," &
        -- num    cell   port          func     safe [ccell dis rslt]
           "40    (BC_1, *,             control,  1)," &
           "41    (BC_6, COLUMN(4),     bidir,    X,   40,  1,   Z)," &
           "42    (BC_1, *,             control,  1)," &
           "43    (BC_6, COLUMN(3),     bidir,    X,   42,  1,   Z)," &
           "44    (BC_1, *,             control,  1)," &
           "45    (BC_6, COLUMN(2),     bidir,    X,   44,  1,   Z)," &
           "46    (BC_1, *,             control,  1)," &
           "47    (BC_6, COLUMN(1),     bidir,    X,   46,  1,   Z)," &
           "48    (BC_1, *,             control,  1)," &
           "49    (BC_6, COLUMN(0),     bidir,    X,   48,  1,   Z)," &
           "50    (BC_1, STO,           output2,  X)," &
           "51    (BC_1, RESET_IN_B,    input,    0)," &
           "52    (BC_1, RESET_OUT_B,   output2,  X)," &
           "53    (BC_1, BMODE,         input,    X)," &
           "54    (BC_1, *,             control,  1)," &
           "55    (BC_6, SIMRESET_B,    bidir,    X,   54,  1,   Z)," &
           "56    (BC_1, *,             control,  1)," &
           "57    (BC_6, SENSE,         bidir,    X,   56,  1,   Z)," &
           "58    (BC_1, *,             control,  1)," &
           "59    (BC_6, SIMDATA,       bidir,    X,   58,  1,   Z)," &
        -- num    cell   port          func     safe [ccell dis rslt]
           "60    (BC_1, *,             control,  1)," &
           "61    (BC_6, PWR_EN,        bidir,    X,   60,  1,   Z)," &
           "62    (BC_1, *,             control,  1)," &
           "63    (BC_6, SIMCLK,        bidir,    X,   62,  1,   Z)," &
           "64    (BC_6, DATA(15),      bidir,    X,   72,  1,   Z)," &
           "65    (BC_6, DATA(14),      bidir,    X,   72,  1,   Z)," &
           "66    (BC_6, DATA(13),      bidir,    X,   72,  1,   Z)," &
           "67    (BC_6, DATA(12),      bidir,    X,   72,  1,   Z)," &
           "68    (BC_6, DATA(11),      bidir,    X,   72,  1,   Z)," &
"69    (BC_6, DATA(10),     bidir,     X,   72,  1,   Z)," &
           "70    (BC_6, DATA(9),       bidir,    X,   72,  1,   Z)," &
           "71    (BC_6, DATA(8),       bidir,    X,   72,  1,   Z)," &
           "72    (BC_1, *,             control,  1)," &
           "73    (BC_1, *,             control,  1)," &
           "74    (BC_6, DATA(7),       bidir,    X,   73,  1,   Z)," &
           "75    (BC_6, DATA(6),       bidir,    X,   73,  1,   Z)," &
```

```
           "76    (BC_6, DATA(5),      bidir,    X,   73,  1,   Z)," &
           "77    (BC_6, DATA(4),      bidir,    X,   73,  1,   Z)," &
           "78    (BC_6, DATA(3),      bidir,    X,   73,  1,   Z)," &
           "79    (BC_6, DATA(2),      bidir,    X,   73,  1,   Z)," &
        -- num    cell   port         func     safe [ccell dis rslt]
           "80    (BC_6, DATA(1),      bidir,    X,   73,  1,   Z)," &
           "81    (BC_6, DATA(0),      bidir,    X,   73,  1,   Z)," &
           "82    (BC_1, CS5,          output2,  X)," &
           "83    (BC_1, CS_B(4),      output2,  X)," &
           "84    (BC_1, CS_B(3),      output2,  X)," &
           "85    (BC_1, CS_B(2),      output2,  X)," &
           "86    (BC_1, *,            control,  1)," &
           "87    (BC_1, *,            control,  1)," &
           "88    (BC_1, CS_B(1),      output2,  X)," &
           "89    (BC_1, CS_B(0),      output2,  X)," &
           "90    (BC_6, RW_B,         bidir,    X,   86,  1,   Z)," &
           "91    (BC_1, OE_B,         output2,  X)," &
           "92    (BC_1, CKO,          output2,  X)," &
           "93    (BC_1, CKOH,         output2,  X)," &
           "94    (BC_1, CKIL,         input,    0)," &
           "95    (BC_6, EB_B(0),      bidir,    X,   87,  1,   Z)," &
           "96    (BC_6, EB_B(1),      bidir,    X,   87,  1,   Z)," &
           "97    (BC_6, ADDR(0),      bidir,    X,   101, 1,   Z)," &
           "98    (BC_6, ADDR(1),      bidir,    X,   101, 1,   Z)," &
           "99    (BC_6, ADDR(2),      bidir,    X,   101, 1,   Z)," &
        -- num    cell   port         func     safe [ccell dis rslt]
           "100   (BC_6, ADDR(3),      bidir,    X,   101, 1,   Z)," &
           "101   (BC_1, *,            control,  1)," &
           "102   (BC_6, ADDR(4),      bidir,    X,   101, 1,   Z)," &
           "103   (BC_6, ADDR(5),      bidir,    X,   101, 1,   Z)," &
           "104   (BC_6, ADDR(6),      bidir,    X,   101, 1,   Z)," &
           "105   (BC_6, ADDR(7),      bidir,    X,   101, 1,   Z)," &
           "106   (BC_6, ADDR(8),      bidir,    X,   114, 1,   Z)," &
           "107   (BC_6, ADDR(9),      bidir,    X,   114, 1,   Z)," &
           "108   (BC_6, ADDR(10),     bidir,    X,   114, 1,   Z)," &
           "109   (BC_6, ADDR(11),     bidir,    X,   114, 1,   Z)," &
           "110   (BC_6, ADDR(12),     bidir,    X,   114, 1,   Z)," &
           "111   (BC_6, ADDR(13),     bidir,    X,   114, 1,   Z)," &
           "112   (BC_6, ADDR(14),     bidir,    X,   114, 1,   Z)," &
           "113   (BC_6, ADDR(15),     bidir,    X,   114, 1,   Z)," &
           "114   (BC_1, *,            control,  1)," &
           "115   (BC_6, ADDR(16),     bidir,    X,   114, 1,   Z)," &
"116   (BC_6, ADDR(17),     bidir,    X,   114, 1,   Z)," &
           "117   (BC_6, ADDR(18),     bidir,    X,   114, 1,   Z)," &
           "118   (BC_6, ADDR(19),     bidir,    X,   114, 1,   Z)," &
           "119   (BC_1, ADDR20,       output2,  X)," &
        -- num    cell   port         func     safe [ccell dis rslt]
           "120   (BC_1, ADDR21,       output2,  X)," &
           "121   (BC_1, *,            control,  1)," &
           "122   (BC_6, TOUT(0),      bidir,    X,   121, 1,   Z)," &
           "123   (BC_1, *,            control,  1)," &
           "124   (BC_6, TOUT(1),      bidir,    X,   123, 1,   Z)," &
           "125   (BC_1, *,            control,  1)," &
           "126   (BC_6, TOUT(2),      bidir,    X,   125, 1,   Z)," &
           "127   (BC_1, *,            control,  1)," &
           "128   (BC_6, TOUT(3),      bidir,    X,   127, 1,   Z)," &
```

```
           "129   (BC_1, *,            control,  1)," &
           "130   (BC_6, TOUT(4),      bidir,    X,   129, 1,   Z)," &
           "131   (BC_1, *,            control,  1)," &
           "132   (BC_6, TOUT(5),      bidir,    X,   131, 1,   Z)," &
           "133   (BC_1, *,            control,  1)," &
           "134   (BC_6, TOUT(6),      bidir,    X,   133, 1,   Z)," &
           "135   (BC_1, *,            control,  1)," &
           "136   (BC_6, TOUT(7),      bidir,    X,   135, 1,   Z)," &
           "137   (BC_1, *,            control,  1)," &
           "138   (BC_6, SPICS(4),     bidir,    X,   137, 1,   Z)," &
           "139   (BC_1, *,            control,  1)," &
           "140   (BC_6, SPICS(3),     bidir,    X,   139, 1,   Z)," &
        -- num    cell   port         func     safe [ccell dis rslt]
           "141   (BC_1, *,            control,  1)," &
           "142   (BC_6, SPICS(2),     bidir,    X,   141, 1,   Z)," &
           "143   (BC_1, *,            control,  1)," &
           "144   (BC_6, SPICS(1),     bidir,    X,   143, 1,   Z)," &
           "145   (BC_1, *,            control,  1)," &
           "146   (BC_6, SPICS(0),     bidir,    X,   145, 1,   Z)," &
           "147   (BC_1, *,            control,  1)," &
           "148   (BC_6, SCK,          bidir,    X,   147, 1,   Z)," &
           "149   (BC_1, *,            control,  1)," &
           "150   (BC_6, MISO,         bidir,    X,   149, 1,   Z)," &
           "151   (BC_1, *,            control,  1)," &
           "152   (BC_6, MOSI,         bidir,    X,   151, 1,   Z)," &
           "153   (BC_1, DSP_IRQ_B,    input,    X)," &
           "154   (BC_1, *,            control,  1)," &
           "155   (BC_6, SCKB,         bidir,    X,   154, 1,   Z)," &
           "156   (BC_1, *,            control,  1)," &
           "157   (BC_6, SCB(0),       bidir,    X,   156, 1,   Z)," &
           "158   (BC_1, *,            control,  1)," &
           "159   (BC_6, SCB(1),       bidir,    X,   158, 1,   Z)," &
        -- num    cell   port         func     safe [ccell dis rslt]
           "160   (BC_1, *,            control,  1)," &
           "161   (BC_6, SCB(2),       bidir,    X,   160, 1,   Z)," &
"162   (BC_1, *,            control,  1)," &
           "163   (BC_6, SRDB,         bidir,    X,   162, 1,   Z)," &
           "164   (BC_1, *,            control,  1)," &
           "165   (BC_6, STDB,         bidir,    X,   164, 1,   Z)," &
           "166   (BC_1, *,            control,  1)," &
           "167   (BC_6, SCA(2),       bidir,    X,   166, 1,   Z)," &
           "168   (BC_1, *,            control,  1)," &
           "169   (BC_6, SCA(1),       bidir,    X,   168, 1,   Z)," &
           "170   (BC_1, *,            control,  1)," &
           "171   (BC_6, SCA(0),       bidir,    X,   170, 1,   Z)," &
           "172   (BC_1, *,            control,  1)," &
           "173   (BC_6, SCKA,         bidir,    X,   172, 1,   Z)," &
           "174   (BC_1, *,            control,  1)," &
           "175   (BC_6, SRDA,         bidir,    X,   174, 1,   Z)," &
           "176   (BC_1, *,            control,  1)," &
           "177   (BC_6, STDA,         bidir,    X,   176, 1,   Z)," &
           "178   (BC_1, *,            control,  1)," &
           "179   (BC_6, PSTAT(3),     bidir,    X,   178, 1,   Z)," &
        -- num    cell   port         func     safe [ccell dis rslt]
           "180   (BC_1, *,            control,  1)," &
           "181   (BC_6, PSTAT(2),     bidir,    X,   180, 1,   Z)," &
```

```
          "182   (BC_1, *,            control,  1)," &
          "183   (BC_6, PSTAT(1),     bidir,    X,   182, 1,   Z)," &
          "184   (BC_1, *,            control,  1)," &
          "185   (BC_6, PSTAT(0),     bidir,    X,   184, 1,   Z)," &
          "186   (BC_1, *,            control,  1)," &
          "187   (BC_6, SIZ(1),       bidir,    X,   186, 1,   Z)," &
          "188   (BC_1, *,            control,  1)," &
          "189   (BC_6, SIZ(0),       bidir,    X,   188, 1,   Z)," &
          "190   (BC_1, *,            control,  1)," &
          "191   (BC_6, CTS_B,        bidir,    X,   190, 1,   Z)," &
          "192   (BC_1, *,            control,  1)," &
          "193   (BC_6, RTS_B,        bidir,    X,   192, 1,   Z)," &
          "194   (BC_1, *,            control,  1)," &
          "195   (BC_6, RX,           bidir,    X,   194, 1,   Z)," &
          "196   (BC_1, *,            control,  1)," &
          "197   (BC_6, TX,           bidir,    X,   196, 1,   Z)" ;

end DSP56652;
```

Appendix D
Programmer's Reference

This appendix provides a set of reference tables to simplify programming the DSP56652. The tables include the following:

- Instruction set summaries for both the MCU and DSP.
- I/O memory maps listing the configuration registers in numerical order.
- A register index providing an alphabetical list of registers and the page numbers in this manual where they are described.
- A list of acronym and bit name changes from previous 56000 and M•CORE family devices.

D.1 MCU Instruction Reference Tables

Table D-1 provides a brief summary of the instruction set for the MCU. Table D-2 on page D-6 and Table D-3 on page D-6 list the abbreviations used in the instruction set summary table. For complete MCU instruction set details, see Section 3 of the *MCU Reference Manual* (MCORERM/AD).

Table D-1. MCU Instruction Set Summary

Mnemonic	Instruction Syntax	Opcode	C Bit
ABS	ABS RX	0000 0001 1110 rrrr	Unaffected
ADDC	ADDC RX,RY	0000 0110 ssss rrrr	C←carryout
ADDI	ADDI RX,OIMM5	0010 000i iiii rrrr	Unaffected
ADDU	ADDU RX,RY	0001 1100 ssss rrrr	Unaffected
AND	AND RX,RY	0001 0110 ssss rrrr	Unaffected
ANDI	ANDI RX,IMM5	0010 0011 0000 rrrr	Unaffected
ANDN	ANDN RX,RY	0001 1111 ssss rrrr	Unaffected
ASR	ASR RX,RY	0001 1010 ssss rrrr	Unaffected

Table D-1. MCU Instruction Set Summary (Continued)

Mnemonic	Instruction Syntax	Opcode	C Bit
ASRC	ASRC RX	0011 1010 0000 rrrr	RX copied into C bit before shifting
ASRI	ASRI RX,IMM5	0011 101i iiii rrrr	Unaffected
BCLRI	BCLRI RX,IMM5	0011 000i iiii rrrr	Unaffected
BF	BF LABEL	1110 1ddd dddd dddd	Unaffected
BGENI	BGENI RX,IMM5	0011 0010 0111 rrrr	Unaffected
BGENR	BGENR RX,RY	0001 0011 ssss rrrr	Unaffected
BKPT	BKPT	0000 0000 0000 0000	n/a
BMASKI	BMASKI RX,IMM5	0010 0011 0000 rrrr	Unaffected
BR	BR LABEL	1111 0ddd dddd dddd	Unaffected
BREV	BREV RX	0000 0000 1111 rrrr	Unaffected
BSETI	BSETI RX,IMM5	0011 010i iiii rrrr	Unaffected
BSR	BSR LABEL	1111 1ddd dddd dddd	Unaffected
BT	BT LABEL	1110 0ddd dddd dddd	Unaffected
BTSTI	BTSTI RX,IMM5	0011 011i iiii rrrr	Set to value of RX pointed to by IMM5
CLRF	CLRF RX	0000 0001 1101 rrrr	Unaffected
CLRT	CLRT RX	0000 0001 1100 rrrr	Unaffected
CMPHS	CMPHS RX,RY	0000 1100 ssss rrrr	Set as a result of comparison
CMPLT	CMPLT RX,RY	0000 1101 ssss rrrr	Set as a result of comparison
CMPLTI	CMPLTI RX,OIMM5	0010 001i iiii rrrr	Set as a result of comparison
CMPNE	CMPNE RX,RY	0000 1111 ssss rrrr	Set as a result of comparison
CMPNEI	CMPNEI RX,IMM5	0010 101i iiii rrrr	Set as a result of comparison
DECF	DECF RX	0000 0000 1001 rrrr	Unaffected
DECGT	DECGT RX	0000 0001 1010 rrrr	Set if RX > 0, else bit is cleared
DECLT	DECLT RX	0000 0001 1000 rrrr	Set if RX < 0, else bit is cleared
DECNE	DECNE RX	0000 0001 1011 rrrr	Set if RX ≠ 0, else bit is cleared
DECT	DECT RX	0000 0000 1000 rrrr	Unaffected

Table D-1. MCU Instruction Set Summary (Continued)

Mnemonic	Instruction Syntax	Opcode	C Bit
DIVS	DIVS RX,R1	0011 0010 0001 rrrr	Undefined
DIVU	DIVU RX,R1	0010 0011 0001 rrrr	Undefined
DOZE	DOZE	0000 0000 0000 0110	Unaffected
FF1	FF1 RX,R1	0000 0000 1110 rrrr	Unaffected
INCF	INCF RX	0000 0000 1011 rrrr	Unaffected
INCT	INCT RX	0000 0000 1010 rrrr	Unaffected
IXH	IXH RX,RY	0001 1101 ssss rrrr	Unaffected
IXW	IXW RX,RY	0001 0101 ssss rrrr	Unaffected
JMP	JMP RX	0000 0000 1100 rrrr	Unaffected
JMPI	JMPI [LABEL]	0111 0000 dddd dddd	Unaffected
JSR	JSR RX	0000 0000 1101 rrrr	Unaffected
JSRI	JSRI [LABEL]	0111 1111 dddd dddd	Unaffected
LD.[BHW]	LD.[B, H, W] RZ, (RX,DISP) [LD, LDB, LDH, LDW] RZ,(RX,DISP)	1000 zzzz iiii rrrr	Unaffected
LDM	LDM RF–R15,(R0)	0000 0000 0110 rrrr	Unaffected
LDQ	LDQ R4–R7,(RX)	0000 0000 0100 rrrr	Unaffected
LOOPT	LOOPT RY,LABEL	0000 0100 ssss bbbb	Set if signed result in RY > 0, else bit is cleared
LRW	LRW RZ,LABEL	0111 zzzz dddd dddd	Unaffected
LSL	LSL RX,RY	0001 1011 ssss rrrr	Unaffected
LSLC	LSLC RX	0011 1100 0000 rrrr	Copy RX[31] into C before shifting
LSLI	LSLI RX,IMM5	0011 110i iiii rrrr	Unaffected
LSR	LSR RX,RY	0000 1011 ssss rrrr	Unaffected
LSRC	LSRC RX	0011 1110 0000 rrrr	Copy RX0 into C before shifting
LSRI	LSRI RX,IMM5	0011 111i iiii rrrr	Unaffected
MFCR	MFCR RX,CRY	0001 000c cccc rrrr	Unaffected
MOV	MOV RX,RY	0001 0010 ssss rrrr	Unaffected
MOVF	MOVF RX,RY	0000 1010 ssss rrrr	Unaffected
MOVI	MOVI RX,IMM7	0110 0iii iiii rrrr	Unaffected
MOVT	MOVT RX,RY	0000 0010 ssss rrrr	Unaffected

Table D-1. MCU Instruction Set Summary (Continued)

Mnemonic	Instruction Syntax	Opcode	C Bit
MTCR	MTCR RX, CRY	0001 100c cccc rrrr	Unaffected unless CR0 (PSR) specified
MULT	MULT RX,RY	0000 0011 ssss rrrr	Unaffected
MVC	MVC RX	0000 0000 0001 rrrr	Unaffected
MVCV	MVCV RX	0000 0000 0011 rrrr	Unaffected
NOT	NOT RX	0000 0001 1111 rrrr	Unaffected
OR	OR RX,RY	0001 1110 ssss rrrr	Unaffected
RFI	RFI	0000 0000 0000 0011	
ROTLI	ROTLI RX,IMM5	0011 100i iiii rrrr	Unaffected
RSUB	RSUB RX,RY	0001 0100 ssss rrrr	Unaffected
RSUBI	RSUBI RX,IMM5	0010 100i iiii rrrr	Unaffected
RTE	RTE	0000 0000 0000 0010	n/a
SEXTB	SEXTB RX	0000 0001 0101 rrrr	Unaffected
SEXTH	SEXTH RX	0000 0001 0111 rrrr	Unaffected
ST.[BHW]	ST.[B, H, W] RZ, (RX,DISP) [ST, STB, STH, STW] RZ,(RX,DISP)	1001 zzzz iiii rrrr	Unaffected
STM	STM RF–R15,(R0)	0000 0000 0111 rrrr	Unaffected
STOP	STOP	0000 0000 0000 0100	Unaffected
STQ	STQ R4–R7,(RX)	0000 0000 0101 rrrr	Unaffected
SUBC	SUBC RX,RY	0000 0111 ssss rrrr	C←carryout
SUBI	SUBI RX,IMM5	0010 010i iiii rrrr	Unaffected
SUBU	SUBU RX,RY SUB RX,RY	0000 0101 ssss rrrr	Unaffected
SYNC	SYNC	0000 0000 0000 0001	Unaffected
TRAP	TRAP #TRAP_NUMBER	0000 0000 0000 10ii	Unaffected
TST	TST RX,RY	0000 1110 ssss rrrr	Set if (RX & RY) ≠ 0, else bit is cleared
TSTNBZ	TSTNBZ RX	0000 0001 1001 rrrr	Set to result of test
WAIT	WAIT	0000 0000 0000 0101	n/a
XOR	XOR RX,RY	0001 0111 ssss rrrr	Unaffected
XSR	XSR RX	0011 1000 0000 rrrr	Set to original value of RX[0]
XTRB0	XTRB0 R1,RX	0000 0001 0011 rrrr	Set if result ≠ 0, else bit is cleared

Table D-1. MCU Instruction Set Summary (Continued)

Mnemonic	Instruction Syntax	Opcode	C Bit
XTRB1	XTRB1 R1,RX	0000 0001 001 0 rrrr	Set if result ≠ 0, else bit is cleared
XTRB2	XTRB2 R1,RX	0000 0001 0001 rrrr	Set if result ≠ 0, else bit is cleared
XTRB3	XTRB3 R1,RX	0000 0001 0000 rrrr	Set if result ≠ 0, else bit is cleared
ZEXTB	ZEXTB RX	0000 0001 0100 rrrr	Unaffected
ZEXTH	ZEXTH RX	0000 0001 0110 rrrr	Unaffected

Table D-2. MCU Instruction Syntax Notation

Symbol	Description
RX	**Source or destination register R0–R15**
RY	Source or destination register R0–R15
RZ	Source or destination register R0–R15 (range may be restricted)
IMM5	5-bit immediate value
OIMM5	5-bit immediate value offset (incremented) by 1
IMM7	7-bit immediate value
LABEL	
R1	Register R1
DISP	Displacement specified
B	Byte (8 bits)
H	Half-word (16 bits)
W	Word (32 bits)
RF	Register First (any register from R1 to R14; R0 and R15 are invalid)
R4–R7	The four registers R4–R7
CRY	Source control register CR0–CR31

Table D-3. MCU Instruction Opcode Notation

Symbol	Description
rrrr	RX field
ssss	RY field
zzzz	RZ field
ffff	Rfirst field
cccc	Control register specifier
iii ... i	One of several immediate fields
xx ... x	Undefined fields

D.2 DSP Instruction Reference Tables

Table D-4 provide a brief summary of the instruction set for the DSP core. Table D-5, Table D-6, and Table D-7 list the abbreviations used in the instruction set summary table. For complete DSP instruction set details, see Appendix A of the *DSP56600 Family Manual* (DSP56600FM/AD).

Table D-4. DSP Instruction Set Summary

Mnemonic	Syntax	P	T	CCR							
				S	L	E	U	N	Z	V	C
ABS	ABS D	P		*	*	*	*	*	*	*	—
ADC	ADC S,D	P		*	*	*	*	*	*	*	*
ADD	ADD S,D	P		*	*	*	*	*	*	*	*
	ADD #iiiiii,D	—	2	*	*	*	*	*	*	*	*
	ADD #iii,D	—	1	*	*	*	*	*	*	*	*
ADDL	ADDL S,D	P		*	*	*	*	*	*	?	*
ADDR	ADDR S,D	P		*	*	*	*	*	*	*	*
AND	AND S,D	P		*	—	—	—	?	?	0	—
	AND #iiiiii,D	—	2	*	—	—	—	?	?	0	—
AND	AND #iii,D	—	1	*	—	—	—	?	?	0	—
ANDI	ANDI EE	—	3	?	?	?	?	?	?	?	?
ASL	ASL S,D	P		*	*	*	*	*	*	?	?
	ASL #ii,S,D	—	1	*	*	*	*	*	*	?	?
	ASL sss,S,D	—	1	*	*	*	*	*	*	?	?
ASR	ASR S,D	P		*	*	*	*	*	*	0	?
	ASR sss,S,D	—	1	*	*	*	*	*	*	0	?
	ASR #ii,S,D	—	1	*	*	*	*	*	*	0	?
Bcc	Bcc (PC + Rn)	—	4	—	—	—	—	—	—	—	—
	Bcc (PC + aa)	—	4	—	—	—	—	—	—	—	—
BCHG	BCHG #bbbb , S:<aa>	—	2	?	?	?	?	?	?	?	?
	BCHG #bbbb , S:<ea>	—	2 + U + A	?	?	?	?	?	?	?	?
	BCHG #bbbb , S:<pp>	—	2	?	?	?	?	?	?	?	?
	BCHG #bbbb , S:<qq>	—	2	?	?	?	?	?	?	?	?
	BCHG #bbbb, DDDDDD	—	2	?	?	?	?	?	?	?	?
BCLR	BCLR #bbbb , S:<pp>	—	2	?	?	?	?	?	?	?	?
	BCLR #bbbb , S:<ea>	—	2 + U + A	?	?	?	?	?	?	?	?
	BCLR #bbbb , S:<aa>	—	2	?	?	?	?	?	?	?	?
	BCLR #bbbb , S:<qq>	—	2	?	?	?	?	?	?	?	?
	BCLR #bbbb , DDDDDD	—	2	?	?	?	?	?	?	?	?
BRA	BRA (PC + Rn)	—	4	—	—	—	—	—	—	—	—
	BRA (PC + aa)	—	4	—	—	—	—	—	—	—	—

Table D-4. DSP Instruction Set Summary (Continued)

Mnemonic	Syntax	P	T	CCR							
				S	L	E	U	N	Z	V	C
BRKcc	BRKcc	—	5	—	—	—	—	—	—	—	—
BScc	BScc (PC + Rn)	—	4	—	—	—	—	—	—	—	—
	BScc (PC + aa)	—	4	—	—	—	—	—	—	—	—
BSET	BSET #bbbb,S:<pp>	—	2	?	?	?	?	?	?	?	?
	BSET #bbbb, S:<ea>	—	2 + U + A	?	?	?	?	?	?	?	?
	BSET #bbbb, S:<aa>	—	2	?	?	?	?	?	?	?	?
	BSET #bbbb , DDDDDD	—	2	?	?	?	?	?	?	?	?
	BSET #bbbb , S:<qq>	—	2	?	?	?	?	?	?	?	?
BSR	BSR (PC + Rn)	—	4	—	—	—	—	—	—	—	—
	BSR (PC + aa)	—	4	—	—	—	—	—	—	—	—
BTST	BTST #bbbb,S:<pp>	—	2	*	*	—	—	—	—	—	?
	BTST #bbb ,S:<ea>	—	2 + U + A	*	*	—	—	—	—	—	?
	BTST #bbbb,S:<aa>	—	2	*	*	—	—	—	—	—	?
	BTST #bbbb , DDDDDD	—	2	*	*	—	—	—	—	—	?
	BTST #bbbb,S:<qq>	—	2	*	*	—	—	—	—	—	?
CLB	CLB S,D	—	1	—	—	—	—	?	?	0	—
CLR	CLR D	P		*	*	0	1	0	1	0	—
CMP	CMP S1,S2	P		*	*	*	*	*	*	*	*
	CMP #iiiiii,D	—	2	*	*	*	*	*	*	*	*
	CMP #iii,D	—	1	*	*	*	*	*	*	*	*
CMPM	CMPM S1,S2	P		*	*	*	*	*	*	*	*
CMPU	CMPU ggg,D	—	1	—	—	—	—	*	?	0	*
DEBUG	DEBUG	—	1	—	—	—	—	—	—	—	—
DEBUGcc	DEBUGcc	—	5	—	—	—	—	—	—	—	—
DEC	DEC	—	1	—	*	*	*	*	*	*	*
DIV	DIV	—	1	—	?	—	—	—	—	?	?
DMAC	DMAC S1,S2,D (ss,su,uu)	N	1	—	*	*	*	*	*	*	—
DO	DO #xxx,aaaa	—	5	?	?	—	—	—	—	—	—
	DO DDDDDD,aaaa	—	5	?	?	—	—	—	—	—	—
	DO S:<ea>,aaaa	—	5 + U	?	?	—	—	—	—	—	—
	DO S:<aa>,aaaa	—	5	?	?	—	—	—	—	—	—
DO FOREVER	DO FOREVER , (aaaa)	—	4	—	—	—	—	—	—	—	—
ENDDO	ENDDO	—	1	—	—	—	—	—	—	—	—
EOR	EOR S,D	P		*	*	—	—	?	?	0	—
	EOR #iiiiii,D	—	2	*	*	—	—	?	?	0	—
	EOR #iii,D	—	1	*	*	—	—	?	?	0	—
EXTRACT	EXTRACT SSS,s,D	—	1	—	—	*	*	*	*	0	0
	EXTRACT #iiii,s,D	—	2	—	—	*	*	*	*	0	0

Table D-4. DSP Instruction Set Summary (Continued)

Mnemonic	Syntax	P	T	CCR							
				S	L	E	U	N	Z	V	C
EXTRACTU	EXTRACTU SSS,s,D	—	1	—	—	*	*	*	*	0	0
	EXTRACTU #iiii,s,D	—	2	—	—	*	*	*	*	0	0
IFcc	IFcc	—	1	—	—	—	—	—	—	—	—
IFcc(.U)	IFcc(.U)	—		?	?	?	?	?	?	?	?
ILLEGAL	ILLEGAL	—	5	—	—	—	—	—	—	—	—
INC	INC D	—	1	—	*	*	*	*	*	*	*
INSERT	INSERT SSS,qqq,D	—	1	—	—	*	*	*	*	0	0
	INSERT #iiii,qqq,D	—	2	—	—	*	*	*	*	0	0
Jcc	Jcc aa	—	4	—	—	—	—	—	—	—	—
	Jcc ea	—	4	—	—	—	—	—	—	—	—
JCLR	JCLR #bbbb,S:<ea>,aaaa	—	4 + U	*	*	—	—	—	—	—	—
	JCLR #bbbb,S:<pp>,aaaa	—	4	*	*	—	—	—	—	—	—
	JCLR #bbbb ,S:<aa>,aaaa	—	4	*	*	—	—	—	—	—	—
	JCLR #bbbb,DDDDDD,aaaa	—	4	*	*	—	—	—	—	—	—
	JCLR #bbbb, S:<qq>,aaaa	—	4	*	*	—	—	—	—	—	—
JMP	JMP aa	—	3	—	—	—	—	—	—	—	—
	JMP ea	—	3 + U + A	—	—	—	—	—	—	—	—
JScc	JScc aa	—	4	—	—	—	—	—	—	—	—
	JScc ea	—	4	—	—	—	—	—	—	—	—
JSCLR	JSCLR #bbbb,S:<pp>,aaaa	—	4	*	*	—	—	—	—	—	—
	JSCLR #bbbb , S:<ea>,aaaa	—	4 + U	*	*	—	—	—	—	—	—
	JSCLR #bbbb , S:<aa>,aaaa	—	4	*	*	—	—	—	—	—	—
	JSCLR #bbbb, DDDDDD,aaaa	—	4	*	*	—	—	—	—	—	—
	JSCLR #bbbb , S:<qq>,aaaa	—	4	*	*	—	—	—	—	—	—
JSET	JSET #bbbb , S:<pp>,aaaa	—	4	*	*	—	—	—	—	—	—
	JSET #bbbb , S:<ea>,aaaa	—	4 + U	*	*	—	—	—	—	—	—
	JSET #bbbb , S:<aa>,aaaa	—	4	*	*	—	—	—	—	—	—
	JSET #bbbb, DDDDDD,aaaa	—	4	*	*	—	—	—	—	—	—
	JSET #bbbb , S:<qq>,aaaa	—	4	*	*	—	—	—	—	—	—
JSR	JSR aa	—	3	—	—	—	—	—	—	—	—
	JSR ea	—	3 + U + A	—	—	—	—	—	—	—	—
JSSET	JSSET #bbbb,S:<pp>,aaaa	—	4	*	*	—	—	—	—	—	—
	JSSET #bbbb,S:<ea>,aaaa	—	4 + U	*	*	—	—	—	—	—	—
	JSSET #bbbb,S:<aa>,aaaa	—	4	*	*	—	—	—	—	—	—
	JSSET #bbbb, DDDDDD,aaaa	—	4	*	*	—	—	—	—	—	—
	JSSET #bbbb,S:<qq>,aaaa	—	4	*	*	—	—	—	—	—	—
LRA	LRA (PC + Rn) → 0DDDDD	—	3	—	—	—	—	—	—	—	—
	LRA (PC + aaaa) → 0DDDDD	—	3	—	—	—	—	—	—	—	—

Table D-4. DSP Instruction Set Summary (Continued)

Mnemonic	Syntax	P	T	CCR							
				S	L	E	U	N	Z	V	C
LSL	LSL D	P		*	*	—	—	?	?	0	?
	LSL sss,D	—	1	*	*	—	—	?	?	0	?
	LSL #ii,D	—	1	*	*	—	—	?	?	0	?
LSR	LSR D	P		*	*	—	—	?	?	0	?
	LSR #ii,D	—	1	*	*	—	—	?	?	0	?
	LSR sss,D	—	1	*	*	—	—	?	?	0	?
LUA, LEA	LUA ea → 0DDDDD	—	3	—	—	—	—	—	—	—	—
	LUA (Rn + aa) → 01DDDD	—	3	—	—	—	—	—	—	—	—
MAC	MAC ± 2**s,QQ,d	—	1	*	*	*	*	*	*	*	—
	MAC S1,S2,D	—	1	*	*	*	*	*	*	*	—
MAC (su,uu)	MAC S1,S2,D	N	1	—	*	*	*	*	*	*	—
MACI	MACI ± #iiiiii,QQ,D	—	2	—	*	*	*	*	*	*	—
MACR	MACR ±2**s,QQ,d	—	1	*	*	*	*	*	*	*	—
MACRI	MACRI ± #iiiiii,QQ,D	—	2	—	*	*	*	*	*	*	—
MAX	MAX A,B	P	1	*	*	—	—	—	—	—	?
MAXM	MAXM A,B	P	1	*	*	—	—	—	—	—	?
MERGE	MERGE SSS,D	—	1	—	—	—	—	?	?	0	—
MOVE	No Parallel Data Move (DALU)	N	1	—	—	—	—	—	—	—	—
	MOVE #xx→DDDDD	—	1	—	—	—	—	—	—	—	—
	MOVE ddddd→DDDDD	—	1	*	*	—	—	—	—	—	—
	U move	—	1	—	—	—	—	—	—	—	—
	MOVE S:<ea>,DDDDD	—	1+U+A+I	*	*	—	—	—	—	—	—
	MOVE S:<aa>,DDDDD	—	1	*	*	—	—	—	—	—	—
	MOVE S:<Rn + aa>,DDDD	—	2	*	*	—	—	—	—	—	—
	MOVE S:<Rn + aaaa>,DDDDDD	—	3	*	*	—	—	—	—	—	—
	MOVE d →X Y:<ea>,YY	—	1+U+A+I	*	*	—	—	—	—	—	—
	MOVE X:<ea>,XX & d→Y	—	1+U+A+I	*	*	—	—	—	—	—	—
	MOVE A → X:<ea> X0 A	—	1 + U	*	*	—	—	—	—	—	—
	MOVE B → X:<ea> X0 B	—	1 + U	*	*	—	—	—	—	—	—
	MOVE Y0 → A A Y:<ea>	—	1 + U	*	*	—	—	—	—	—	—
	MOVE Y0 → B B Y:<ea>	—	1 + U	*	*	—	—	—	—	—	—
	MOVE L:<ea>,LLL	—	1 + U + A	*	*	—	—	—	—	—	—
	MOVE L:<aa>,LLL	—	1	*	*	—	—	—	—	—	—
	MOVE X:<ea>,XX & Y:<ea>,YY	—	1	*	*	—	—	—	—	—	—
MOVEC	MOVEC #xx → 1DDDDD	—	1	?	?	?	?	?	?	?	?
	MOVEC S:<ea>,1DDDDD	—	1+U+A+I	?	?	?	?	?	?	?	?
	MOVEC S:<aa>,1DDDDD	—	1	?	?	?	?	?	?	?	?
MOVEC	MOVEC DDDDDD, 1ddddd	—	1	?	?	?	?	?	?	?	?

Table D-4. DSP Instruction Set Summary (Continued)

Mnemonic	Syntax	P	T	CCR							
				S	L	E	U	N	Z	V	C
MOVEM	MOVEM P:<ea>,DDDDDD	—	6 + U + A	?	?	?	?	?	?	?	?
	MOVEM P:<aa>,DDDDDD	—	6	?	?	?	?	?	?	?	?
MOVEP	MOVEP S:<pp>,s:<ea>	—	2 + U + A	?	?	?	?	?	?	?	?
	MOVEP S:<pp>,P:<ea>	—	6 + U + A	?	?	?	?	?	?	?	?
	MOVEP S:<pp>,DDDDDD	—	1	?	?	?	?	?	?	?	?
	MOVEP X:<qq>,s:<ea>	—	2 + U + A	?	?	?	?	?	?	?	?
	MOVEP Y:<qq>,s:<ea>	—	2 + U + A	?	?	?	?	?	?	?	?
	MOVEP X:<qq>,DDDDDD	—	1	?	?	?	?	?	?	?	?
	MOVEP Y:<qq>,DDDDDD	—	1	?	?	?	?	?	?	?	?
	MOVEP S:<qq>,P:<ea>	—	6 + U + A	?	?	?	?	?	?	?	?
MPY	MPY ± 2**s,QQ,d	—	1	*	*	*	*	*	*	*	—
MPY(su,uu)	MPY S1,S2,D (su,uu)	—	1	—	*	*	*	*	*	*	—
MPYI	MPYI ± #iiiiii,QQ,D	—	2	—	*	*	*	*	*	*	—
MPYR	MPYR ± 2**s,QQ,d	—	1	*	*	*	*	*	*	*	—
MPYRI	MPYRI ± #iiiiii,QQ,D	—	2	—	*	*	*	*	*	*	—
NEG	NEG D	P		*	*	*	*	*	*	*	—
NOP	NOP	—	1	—	—	—	—	—	—	—	—
NORMF	NORMF SSS,D	—	1	—	*	*	*	*	*	?	—
NOT	NOT D	P		*	*	—	—	?	?	0	—
OR	OR SD	P		*	*	—	—	?	?	0	—
	OR #iiiiii,D	—	2	*	*	—	—	?	?	0	—
	OR #iii,D	—	1	*	*	—	—	?	?	0	—
ORI	ORI EE	—	3	?	?	?	?	?	?	?	?
REP	REP #xxx	—	5	*	*	—	—	—	—	—	—
	REP DDDDDD	—	5	*	*	—	—	—	—	—	—
	REP S:<ea>	—	5 + U	*	*	—	—	—	—	—	—
	REP S:<aa>	—	5	*	*	—	—	—	—	—	—
RESET	RESET	—	7	—	—	—	—	—	—	—	—
RND	RND D	P		*	*	*	*	*	*	*	—
ROL	ROL D	P		*	*	—	—	?	?	0	?
ROR	ROR D	P		*	*	—	—	?	?	0	?
RTI	RTI	—	3	?	?	?	?	?	?	?	?
RTS	RTS	—	3	—	—	—	—	—	—	—	—
SBC	SBC S,D	P		*	*	*	*	*	*	*	*
STOP	STOP	—	10	—	—	—	—	—	—	—	—
SUB	SUB S,D	P		*	*	*	*	*	*	*	*
	SUB #iiiiii,D	—	2	*	*	*	*	*	*	*	*
	SUB #iii,D	—	1	*	*	*	*	*	*	*	*
SUBL	SUBL S,D	P		*	*	*	*	*	*	?	*

Table D-4. DSP Instruction Set Summary (Continued)

Mnemonic	Syntax	P	T	CCR							
				S	L	E	U	N	Z	V	C
SUBR	SUBR S,D	P		*	*	*	*	*	*	*	*
Tcc	Tcc JJJ → D ttt TTT	—	1	—	—	—	—	—	—	—	—
	Tcc JJJ → D	—	1	—	—	—	—	—	—	—	—
	Tcc ttt → TTT	—	1	—	—	—	—	—	—	—	—
TFR	TFR S,D	P		*	*	—	—	—	—	—	—
TRAP	TRAP	—	9	—	—	—	—	—	—	—	—
TRAPcc	TRAPcc	—	9	—	—	—	—	—	—	—	—
TST	TST S	P		*	*	*	*	*	*	0	—
VSL	VSL S,i,L:ea	—	1 + U + A	—	—	—	—	—	—	—	—
WAIT	WAIT	—	10	—	—	—	—	—	—	—	—

Table D-5. Program Word and Timing Symbols

Column	Description and Symbols	
P	Parallel Move	
	P	Parallel Move
	N	No Parallel Move
	—	Not Applicable
T	Instruction Clock Cycle Counts (Add one cycle for each symbol in column)	
	U	Pre-Update
	A	Long Absolute
	I	Long Immediate

Table D-6. Condition Code Register (CCR) Symbols

Symbol	Description
S	Scaling bit indicating data growth is detected
L	Limit bit indicating arithmetic overflow and/or data limiting
E	Extension bit indicating if the integer portion is in use
U	Unnormalized bit indicating if the result is unnormalized
N	Negative bit indicating if Bit 35 (or 31) of the result is set
Z	Zero bit indicating if the result equals 0
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 D-7. Condition Code Register Notation

Notation	Description
*	Bit is set or cleared according to the standard definition by the result of the operation
—	Bit is not affected by the operation
0	Bit is always cleared by the operation
1	Bit is always set by the operation
U	Undefined
?	Bit is set or cleared according to the special computation definition by the result of the operation

D.3 MCU Internal I/O Memory Map

Table D-8 lists the MCU I/O registers in address numerical order. Unlisted addresses are reserved.

Table D-8. MCU Internal I/O Memory Map

Address	Register Name		Reset Value
Interrupts[1]			
$0020_0000	ISR	Interrupt Source Register	$0007
$0020_0004	NIER	Normal Interrupt Enable Register	$0000
$0020_0008	FIER	Fast Interrupt Enable Register	$0000
$0020_000C	NIPR	Normal Interrupt Pending Register	$0000
$0020_0010	FIPR	Fast Interrupt Pending Register	$0000
$0020_0014	ICR	Interrupt Control Register	$0000
External Interface Module (EIM)[1]			
$0020_1000	CS0	Chip Select 0 Register	$F861
$0020_1004	CS1	Chip Select 1 Register	$uuuu
$0020_1008	CS2	Chip Select 2 Register	$uuuu
$0020_100C	CS3	Chip Select 3 Register	$uuuu
$0020_1010	CS4	Chip Select 4 Register	$uuuu
$0020_1014	CS5	Chip Select 5 Register	$uuuu
$0020_1018	EIMCR	EIM Configuration Register	$0038
MCU-DSP Interface (MDI)			
$0020_2FF2	MCVR	MCU-Side Command Vector Register	$0060
$0020_2FF4	MCR	MCU-Side Control Register	$0000
$0020_2FF6	MSR	MCU-Side Status Register	$3080
$0020_2FF8	MTR1	MCU Transmit Register 1	$0000
$0020_2FFA	MTR0	MCU Transmit Register 0	$0000
$0020_2FFC	MRR1	MCU Receive Register 1	$0000
$0020_2FFE	MRR0	MCU Receive Register 0	$0000

Table D-8. MCU Internal I/O Memory Map (Continued)

Address	Register Name		Reset Value
Protocol Timer (PT)			
$0020_3800	PTCR	PT Control Register	$0000
$0020_3802	PTIER	PT Interrupt Enable Register	$0000
$0020_3804	PTSR	PT Status Register	$0000
$0020_3806	PTEVR	PT Event Register	$0000
$0020_3808	TIMR	Time Interval Modulus Register	$0000
$0020_380A	CTIC	Channel Time Interval Counter	$0000
$0020_380C	CTIMR	Channel Time Interval Modulus Register	$0000
$0020_380E	CFC	Channel Frame Counter	$0000
$0020_3810	CFMR	Channel Frame Modulus Register	$0000
$0020_3812	RSC	Reference Slot Counter	$0000
$0020_3814	RSMR	Reference Slot Modulus Register	$0000
$0020_3816	PTPCR	PT Port Control Register	$0000
$0020_3818	PTDDR	PT Data Direction Register	$0000
$0020_381A	PTPDR	PT Port Data Register	$uuuu
$0020_381C	FTPTR	Frame Table Pointer	$uuuu
$0020_381E	MTPTR	Macro Table Pointer	$uuuu
$0020_3820	FTBAR	Frame Tables Base Address Register	$uuuu
$0020_3822	MTBAR	Macro Tables Base Address Register	$uuuu
$0020_3824	DTPTR	Delay Table Pointer	$uuuu
UART			
$0020_4000 to $0020_403C	URX	UART Receiver Register[2]	$00uu
$0020_4040 to $0020_407C	UTX	UART Transmitter Register[3]	$00uu
$0020_4080	UCR1	UART Control Register 1	$0000
$0020_4082	UCR2	UART Control Register 2	$0000
$0020_4084	UBRGR	UART Bit Rate Generator Register	$0000
$0020_4086	USR	UART Status Register	$A000

Table D-8. MCU Internal I/O Memory Map (Continued)

Address	Register Name		Reset Value
$0020_4088	UTS	UART Test Register	$0000
$0020_408A	UPCR	UART Port Control Register	$0000
$0020_408C	UDDR	UART Data Direction Register	$0000
$0020_408E	UPDR	UART Port Data Register	$000u
Queued Serial Peripheral Interface (QSPI)			
$0020_5000 to $0020_507F		QSPI Control RAM	uuuu
$0020_5400 to $0020_547F		QSPI Data RAM	uuuu
$0020_5F00	QPCR	QSPI Port Control Register	$0000
$0020_5F02	QDDR	QSPI Data Direction Register	$0000
$0020_5F04	QPDR	QSPI Port Data Register	$0000
$0020_5F06	SPCR	Serial Port Control Register	$0000
$0020_5F08	QCR0	Queue Control Register 0	$0000
$0020_5F0A	QCR1	Queue Control Register 1	$0000
$0020_5F0C	QCR2	Queue Control Register 2	$0000
$0020_5F0E	QCR3	Queue Control Register 3	$0000
$0020_5F10	SPSR	Serial Port Status Register	$0000
$0020_5F12	SCCR0	Serial Channel Control Register 0	$0000
$0020_5F14	SCCR1	Serial Channel Control Register 1	$0000
$0020_5F16	SCCR2	Serial Channel Control Register 2	$0000
$0020_5F18	SCCR3	Serial Channel Control Register 3	$0000
$0020_5F1A	SCCR4	Serial Channel Control Register 4	$0000
$0020_5FF8		MCU Trigger for Queue 0	
$0020_5FFA		MCU Trigger for Queue 1	
$0020_5FFC		MCU Trigger for Queue 2	
$0020_5FFE		MCU Trigger for Queue 3	
General-Purpose Timer and Pulse Width Modulator (PWM)			
$0020_6000	TPWCR	Timers and PWM Control Register	$0000
$0020_6002	TPWMR	Timers and PWM Mode Register	$0000

Table D-8. MCU Internal I/O Memory Map (Continued)

Address	Register Name		Reset Value
$0020_6004	TPWSR	Timers and PWM Status Register	$0000
$0020_6006	TPWIR	Timers and PWM Interrupts Enable Register	$0000
$0020_6008	TOCR1	Timer 1 Output Compare Register	$0000
$0020_600A	TOCR3	Timer 3 Output Compare Register	$0000
$0020_600C	TOCR4	Timer 4 Output Compare Register	$0000
$0020_600E	TICR1	Timer 1 Input Capture Register	$0000
$0020_6010	TICR2	Timer 2 Input Capture Register	$0000
$0020_6012	PWOR	PWM Output Compare Register	$0000
$0020_6014	TCNT	Timer Counter	$0000
$0020_6016	PWMR	PWM Modulus Register	$0000
$0020_6018	PWCNT	PWM Counter	$0000
Periodic Interrupt Timer (PIT)			
$0020_7000	PITCSR	PIT Control and Status Register	$0000
$0020_7002	PITMR	PIT Modulus Register	$FFFF
$0020_7004	PITCNT	PIT Counter	$uuuu
Watchdog Timer			
$0020_8000	WCR	Watchdog Control Register	$0000
$0020_8002	WSR	Watchdog Service Register	$0000
Edge Port (EP)			
$0020_9000	EPPAR	Edge Port Pin Assignment Register	$0000
$0020_9002	EPDDR	Edge Port Data Direction Register	$0000
$0020_9004	EPDDR	Edge Port Data Register	$00uu
$0020_9006	EPFR	Edge Port Flag Register	$0000
Keypad Port (KP)			
$0020_A000	KPCR	Keypad Control Register	$0000
$0020_A002	KPSR	Keypad Status Register	$0000
$0020_A004	KDDR	Keypad Data Direction Register	$0000
$0020_A006	KPDR	Keypad Data Register	$uuuu

Table D-8. MCU Internal I/O Memory Map (Continued)

Address	Register Name		Reset Value
Smart Card Port (SCP)			
$0020_B000	SCPCR	SCP Control Register	$0000
$0020_B002	SCACR	Smart Card Activation Control Register	$0000
$0020_B004	SCPIER	SCP Interrupt Enable Register	$0000
$0020_B006	SCPSR	SCP Status Register	$00Cu
$0020_B008	SCPDR	SCP Data Register	$0000
$0020_B00A	SCPPCR	SCP Port Control Register	$000u
MCU Core			
$0020_C000	CKCTL	Clock Control Register	$0000
$0020_C400	RSR	Reset Source Register	
Emulation Port			
$0020_C800	EMDDR	Emulation Port Control Register	$0000
$0020_C802	EMDR	Emulation Port Data Register	$00uu
I/O Multiplexing			
$0020_CC00	GPCR	General Port Control Register	$0000

1. These registers are 32 bits wide.
2. These 16-bit registers are mapped on 32-bit boundaries to support the LDM instruction.
3. These 16-bit registers are mapped on 32-bit boundaries to support the STM instruction.

D.4 DSP Internal I/O Memory Map

Table D-9 lists the DSP I/O registers in address numerical order.

Table D-9. DSP Internal I/O Memory Map

Address	Register Name		Reset Value
MCU-DSP Interface (MDI)			
X:$FF8A	DCR	DSP-Side Control Register	$0
X:$FF8B	DSR	DSP-Side Status Register	$C000
X:$FF8C	DTR1	DSP Transmit Register 1	$0
X:$FF8D	DTR0	DSP Transmit Register 0	$0
X:$FF8E	DRR1	DSP Receive Register1	$0
X:$FF8F	DRR0	DSP Receive Register 0	$0
Baseband Port (BBP)			
X:$FFA4	BBPRMR	BBP Receive Counter Modulus Register	$0
X:$FFA5	BBPTMR	BBP Transmit Counter Modulus Register	$0
X:$FFA6	BBPCRA	BBP Control Register A	$0
X:$FFA7	BBPCRB	BBP Control Register B	$0
X:$FFA8	BBPCRC	BBP Control Register C	$0
X:$FFA9	BBPSR	BBP Status Register	$40
X:$FFAA	BBPRX	BBP Receive Data Register	$FFFF
X:$FFAB	BBPTSR	BBP Time Slot Register	$0
X:$FFAC	BBPTX	BBP Transmit Data Register	$0
X:$FFAD	BBPPDR	BBP Port Data Register	$0
X:$FFAE	BBPDDR	BBP GPIO Direction Register	$0
X:$FFAF	BBPPCR	BBP Port Control Register	$0
Serial Audio Port (SAP)			
X:$FFB4	SAPCNT	SAP Timer Counter	$0
X:$FFB5	SAPMR	SAP Timer Modulus Register	$0
X:$FFB6	SAPCRC	SAP Control Register A	$0
X:$FFB7	SAPCRB	SAP Control Register B	$0

Table D-9. DSP Internal I/O Memory Map (Continued)

Address	Register Name		Reset Value
X:$FFB8	SAPCRA	SAP Control Register C	$0
X:$FFB9	SAPSR	SAP Status Register	$40
X:$FFBA	SAPRX	SAP Receive Data Register	$FFFF
X:$FFBB	SAPTSR	SAP Time Slot Register	$0
X:$FFBC	SAPTX	SAP Transmit Data Register	$0
X:$FFBD	SAPPDR	SAP Port Data Register	$0
X:$FFBE	SAPDDR	SAP GPIO Data Direction Register	$0
X:$FFBF	SAPPCR	SAP Port Control Register	$0

Table D-9. DSP Internal I/O Memory Map (Continued)

Address	Register Name		Reset Value
DSP Core			
X:$FFF5	PAR3	Patch 3 Register	$uuuu
X:$FFF6	PAR2	Patch 2 Register	$uuuu
X:$FFF7	PAR1	Patch 1 Register	$uuuu
X:$FFF8	PAR0	Patch 0 Register	$uuuu
X:$FFF9	IDR	ID Register	$0652
X:$FFFB	OGDB	OnCE GDB Register	$0000
X:$FFFC	PCTL1	PLL Control Register 1	$0010
X:$FFFD	PCTL0	PLL Control Register 0	$0000
X:$FFFE	IPRP	Interrupt Priority Register—Peripheral	$0000
X:$FFFF	IPRC	Interrupt Priority Register—Core	$0000

D.5 Register Index

Table D-10 lists all DSP56652 registers in alphabetical order by acronym, and includes the name, peripheral, address and description page number for each register.

Table D-10. Register Index

Register Name		Peripheral	Address	Page
BBPCRA	BBP Control Register A	BBP	X:$FFA6	14-18
BBPCRB	BBP Control Register B	BBP	X:$FFA7	14-19
BBPCRC	BBP Control Register C	BBP	X:$FFA8	14-21
BBPDDR	BBP GPIO Direction Register	BBP	X:$FFAE	14-24
BBPPCR	BBP Port Control Register	BBP	X:$FFAF	14-25
BBPPDR	BBP Port Data Register	BBP	X:$FFAD	14-24
BBPRMR	BBP Receive Counter Modulus Register	BBP	X:$FFA4	14-17
BBPRX	BBP Receive Data Register	BBP	X:$FFAA	14-23
BBPSR	BBP Status Register	BBP	X:$FFA9	14-22
BBPTMR	BBP Transmit Counter Modulus Register	BBP	X:$FFA5	14-17
BBPTSR	BBP Time Slot Register	BBP	X:$FFAB	14-23
BBPTX	BBP Transmit Data Register	BBP	X:$FFAC	14-23
CFC	Channel Frame Counter	PT	$0020_380E	10-22
CFMR	Channel Frame Modulus Register	PT	$0020_3810	10-23
CKCTL	Clock Control Register	MCU Core	$0020_C000	4-5
CS0	Chip Select 0 Register	EIM	$0020_1000	6-9
CS1	Chip Select 1 Register	EIM	$0020_1004	
CS2	Chip Select 2 Register	EIM	$0020_1008	
CS3	Chip Select 3 Register	EIM	$0020_100C	
CS4	Chip Select 4 Register	EIM	$0020_1010	
CS5	Chip Select 5 Register	EIM	$0020_1014	
CTIC	Channel Time Interval Counter	PT	$0020_380A	10-22
CTIMR	Channel Time Interval Modulus Register	PT	$0020_380C	10-22
DCR	DSP-Side Control Register	MDI	X:$FF8A	5-25
DRR0	DSP Receive Register 0	MDI	X:$FF8F	5-28
DRR1	DSP Receive Register1	MDI	X:$FF8E	5-28
DSR	DSP-Side Status Register	MDI	X:$FF8B	5-26
DTPTR	Delay Table Pointer	PT	$0020_3824	10-25
DTR0	DSP Transmit Register 0	MDI	X:$FF8D	5-28
DTR1	DSP Transmit Register 1	MDI	X:$FF8C	5-28
EIMCR	EIM Configuration Register	EIM	$0020_1018	6-12
EMDDR	Emulation Port Control Register	Emulation	$0020_C800	6-13
EMDR	Emulation Port Data Register	Emulation	$0020_C802	6-13
EPDDR	Edge Port Data Direction Register	EP	$0020_9002	7-17
EPDR	Edge Port Data Register	EP	$0020_9004	7-18

Table D-10. Register Index (Continued)

Register Name		Peripheral	Address	Page
EPFR	Edge Port Flag Register	EP	$0020_9006	7-18
EPPAR	Edge Port Pin Assignment Register	EP	$0020_9000	7-17
FIER	Fast Interrupt Enable Register	Interrupts	$0020_0008	7-7
FIPR	Fast Interrupt Pending Register	Interrupts	$0020_0010	7-9
FTBAR	Frame Tables Base Address Register	PT	$0020_3820	10-24
FTPTR	Frame Table Pointer	PT	$0020_381C	10-24
GPCR	General Port Control Register	I/O Mux	$0020_CC0	4-18
ICR	Interrupt Control Register	Interrupts	$0020_0014	7-10
IDR	ID Register	JTAG	X:$FFF9	4-15
IPRC	Interrupt Priority Register—Core	Interrupts	X:$FFFF	7-15
IPRP	Interrupt Priority Register—Peripheral	Interrupts	X:$FFFE	7-14
ISR	Interrupt Source Register	Interrupts	$0020_0000	7-6
PITCNT	PIT Counter	Timers	$0020_7004	9-4
PITMR	PIT Modulus Register	Timers	$0020_7002	9-4
PITCSR	PIT Control and Status Register	Timers	$0020_7000	9-3
KDDR	Keypad Data Direction Register	KP	$0020_A004	13-6
KPCR	Keypad Control Register	KP	$0020_A000	13-5
KPDR	Keypad Data Register	KP	$0020_A006	13-6
KPSR	Keypad Status Register	KP	$0020_A002	13-5
MCR	MCU-Side Control Register	MDI	$0020_2FF4	5-19
MCVR	MCU-Side Command Vector Register	MDI	$0020_2FF2	5-18
MRR0	MCU Receive Register 0	MDI	$0020_2FFE	5-24
MRR1	MCU Receive Register 1	MDI	$0020_2FFC	5-24
MSR	MCU-Side Status Register	MDI	$0020_2FF6	5-21
MTBAR	Macro Tables Base Address Register	PT	$0020_3822	10-25
MTPTR	Macro Table Pointer	PT	$0020_381E	10-24
MTR0	MCU Transmit Register 0	MDI	$0020_2FFA	5-24
MTR1	MCU Transmit Register 1	MDI	$0020_2FF8	5-24
NIER	Normal Interrupt Enable Register	Interrupts	$0020_0004	7-7
NIPR	Normal Interrupt Pending Register	Interrupts	$0020_000C	7-9
OMR	Operating Mode Register	DSP Core		4-13
PCTL0	PLL Control Register 0	DSP Core	X:$FFFD	4-6
PCTL1	PLL Control Register 1	DSP Core	X:$FFFC	4-7
PTPDR	PT Port Data Register	PT	$0020_381A	10-26
PTDDR	PT Data Direction Register	PT	$0020_3818	10-26
PTPCR	PT Port Control Register	PT	$0020_3816	10-26
PTCR	PT Control Register	PT	$0020_3800	10-17
PTIER	PT Interrupt Enable Register	PT	$0020_3802	10-18
PWCNT	PWM Counter Register	Timers	$0020_6018	9-17
PWMR	PWM Modulus Register	Timers	$0020_6016	9-17

Table D-10. Register Index (Continued)

Register Name		Peripheral	Address	Page
PWOR	PWM Output Compare Register	Timers	$0020_6012	9-17
QCR0	Queue Control Register 0	QSPI	$0020_5F08	8-15
QCR1	Queue Control Register 1	QSPI	$0020_5F0A	
QCR2	Queue Control Register 2	QSPI	$0020_5F0C	
QCR3	Queue Control Register 3	QSPI	$0020_5F0E	
QDDR	QSPI Data Direction Register	QSPI	$0020_5F02	8-25
QPCR	QSPI Port Control Register	QSPI	$0020_5F00	8-24
QPDR	QSPI Port Data Register	QSPI	$0020_5F04	8-25
RSC	Reference Slot Counter	PT	$0020_3812	10-23
RSMR	Reference Slot Modulus Register	PT	$0020_3814	10-23
RSR	Reset Source Register	MCU Core	$0020_C400	4-11
SAPCNT	SAP Timer Counter	SAP	X:$FFB4	14-17
SAPCRA	SAP Control Register C	SAP	X:$FFB8	14-18
SAPCRB	SAP Control Register B	SAP	X:$FFB7	14-19
SAPCRC	SAP Control Register A	SAP	X:$FFB6	14-21
SAPDDR	SAP GPIO Data Direction Register	SAP	X:$FFBE	14-24
SAPMR	SAP Timer Modulus Register	SAP	X:$FFB5	14-17
SAPPCR	SAP Port Control Register	SAP	X:$FFBF	14-25
SAPPDR	SAP Port Data Register	SAP	X:$FFBD	14-24
SAPRX	SAP Receive Data Register	SAP	X:$FFBA	14-23
SAPSR	SAP Status Register	SAP	X:$FFB9	14-22
SAPTSR	SAP Time Slot Register	SAP	X:$FFBB	14-23
SAPTX	SAP Transmit Data Register	SAP	X:$FFBC	14-23
SCACR	Smart Card Activation Control Register	SCP	$0020_B002	12-12
SCCR0	Serial Channel Control Register 0	QSPI	$0020_5F12	8-19
SCCR1	Serial Channel Control Register 1	QSPI	$0020_5F14	
SCCR2	Serial Channel Control Register 2	QSPI	$0020_5F16	
SCCR3	Serial Channel Control Register 3	QSPI	$0020_5F18	
SCCR4	Serial Channel Control Register 4	QSPI	$0020_5F1A	
SCPCR	SCP Control Register	SCP	$0020_B000	12-11
SCPDR	SCP Data Register	SCP	$0020_B008	12-15
SCPIER	SCP Interrupt Enable Register	SCP	$0020_B004	12-13
SCPPCR	SCP Port Control Register	SCP	$0020_B00A	12-16
SCPSR	SCP Status Register	SCP	$0020_B006	12-14
SPCR	Serial Port Control Register	QSPI	$0020_5F06	8-13
SPSR	Serial Port Status Register	QSPI	$0020_5F10	8-17
TCNT	Timer Counter	Timers	$0020_6014	9-17
PTEVR	PT Event Register	PT	$0020_3806	10-21
TICR1	Timer 1 Input Capture Register	Timers	$0020_600E	9-16
TICR2	Timer 2 Input Capture Register	Timers	$0020_6010	

Table D-10. Register Index (Continued)

Register Name		Peripheral	Address	Page
TIMR	Time Interval Modulus Register	PT	$0020_3808	10-21
TOCR1	Timer 1 Output Compare Register	Timers	$0020_6008	9-16
TOCR3	Timer 3 Output Compare Register	Timers	$0020_600A	
TOCR4	Timer 4 Output Compare Register	Timers	$0020_600C	
TPWCR	Timers and PWM Control Register	Timers	$0020_6000	9-13
TPWIR	Timers and PWM Interrupts Enable Register	Timers	$0020_6006	9-16
TPWMR	Timers and PWM Mode Register	Timers	$0020_6002	9-14
TPWSR	Timers and PWM Status Register	Timers	$0020_6004	9-15
PTSR	PT Status Register	PT	$0020_3804	10-20
UBRGR	UART But Rate Generator Register	UART	$0020_4084	11-14
UCR1	UART Control Register 1	UART	$0020_4080	11-11
UCR2	UART Control Register 2	UART	$0020_4082	11-13
UDDR	UART Data Direction Register	UART	$0020_408C	11-16
UPCR	UART Port Control Register	UART	$0020_408A	11-16
UPDR	UART Port Data Register	UART	$0020_408E	11-16
URX	UART Receive Registers	UART	$0020_4000 to $0020_403C	11-9
USR	UART Status Register	UART	$0020_4086	11-14
UTS	UART Test Register	UART	$0020_4088	11-15
UTX	UART Transmit Registers	UART	$0020_4040 to $0020_407C	11-10
WCR	Watchdog Control Register	Timers	$0020_8000	9-6
WSR	Watchdog Service Register	Timers	$0020_8002	9-6

D.6 Acronym Changes

Some register and bit acronyms in the DSP56652 are different than those in previous DSP56000 and M•CORE family devices. Table D-11 presents a summary of the changes. Addresses containing X: are DSP X-memory addresses. All other addresses are the LSP of MCU addresses; the MSP is $0020.

Table D-11. DSP56652 Acronym Changes

Function	Address	Register		Bit #	Bit Name	
		Original	New		Original	New
Interrupts	$0000	ISR	–	30	SMPDINT	SMPD
	$0000	ISR	–	28–25	"L1" replaced with "PT" in all bit names	
	$0004	NIER				
	$0008	FIER				
	$000C	NIPR				
	$0010	FIPR				
	X:$FFFE	IPRP	–	7–6	TIMPL[1:0]	PTPL[1:0]
Edge Port	$9000	EPPAR	–	7–0	EPPAR[7:0]	EPPA[7:0]
QSPI	$5F00	QPCR	–	7–0	PC[7:0]	QPC[7:0]
	$5F02	QDDR	–	7–0	PD[7:0]	QDD[7:0]
	$5F04	QPDR	–	7–0	D[7:0]	QPD[7:0]
PIT	$7000	ITCSR	PITCSR			
	$7002	ITDR	PITMR			
	$7004	ITADR	PITCNT			
PWM	$6014	TCR	TCNT			
	$6016	PWCR	PWMR			
	$6018	PWCNR	PWCNT			
PT	$3800	TCTR	PTCR			
	$3802	TIER	PTIER			
	$3804	TSTR	PTSR			
	$3806	TEVR	PTEVR			
	$3808	TIPR	TIMR	8–0	TIPV[8:0]	TIMV[8:0]
	$380C	CTIPR	CTIMR	13–0	CTIPV[13:0]	CTIMV[13:0]
	$3810	CFPR	CFMR	8–0	CFPV[8:0]	CFMV[8:0]
	$3814	RSPR	RSMR	7–0	RSPV[7:0]	RSMV[7:0]
	$3816	PDPAR	PTPCR	7–0	PDGPC[7:0]	PTPC[7:0]
	$3818	PDDR	PTDDR	7–0	PDDR[7:0]	PTDD[7:0]
	$381A	PDDAT	PTPDR	7–0	PDDAT[7:0]	PTPD[7:0]
	$381E	RTPTR	MTPTR			
	$3822	RTBAR	MTBAR			

Table D-11. DSP56652 Acronym Changes (Continued)

Function	Address	Register		Bit #	Bit Name	
		Original	New		Original	New
UART	$4080	UCR1	–	13	TRDYEN	TRDYIE
				9	RRDYEN	RRDYIE
				6	TXMPTYEN	TXEIE
				5	RTSDEN	RTSDIE
				0	UARTEN	UEN
	$4082	UCR2	–	12	CTS	CTSD
				5	WS	CHSZ
	$4086	USR	–	15	TXMPTY	TXE
	$408A	UPCR	–	3–0	PC[3:0]	UPC[3:0]
	$408C	UDDR	–	3–0	PDC[3:0]	UDD[3:0]
	$408E	UPDR	–	3–0	D[3:0]	UPD[3:0]

Table D-11. DSP56652 Acronym Changes (Continued)

Function	Address	Register		Bit #	Bit Name	
		Original	New		Original	New
SCP	$B000	SIMCR	SCPCR –	9	VOLTSEL	CKSEL
				8	OVRSINK	NKOVR
				5	SISR	SCSSR
				4	SIPT	SCPT
				3	SIIC	SCIC
				2	SINK	NKPE
				1	SITE	SCTE
				0	SIRE	SCRE
	$B002	SIACR	SCACR	4	SICK	SCCLK
				3	SIRS	SCRS
				2	SIOE	SCDPE
				1	SIVE	SCPE
				0	SIAP	APDE
	$B004	SIICR	SCPIER	4	SITCI	SCTCIE
				3	SIFNI	SCFNIE
				2	SIFFI	SCFFIE
				1	SIRRI	SCRRIE
				0	SIPDI	SCSCIE
	$B006	SIMSR	SCPSR	9	SIFF	SCFF
				8	SIFN	SCFN
				7	SITY	SCTY
				6	SITC	SCTC
				5	SITK	TXNK
				4	SIPE	SCPE
				3	SIFE	SCFE
				2	SIOV	SCOE
				1	SIIP	SCSC
				0	SIPD	SCSP
	$B008	SIMDR	SCPDR	7–0	SIMD[7:0]	SCPD[7:0]
	$B00A	SIPCR	SCPPCR	9–5	PDIR[4:0]	SCPDD[4:0]
				4–0	PDAT[4:0]	SCPPD[4:0]

Table D-11. DSP56652 Acronym Changes (Continued)

Function	Address	Register		Bit #	Bit Name	
		Original	New		Original	New
SAP	X:$FFB4	TCRA	SAPCNT			
	X:$FFB5	TCLR	SAPMR			
	X:$FFB6	CRAA	SAPCRA			
	X:$FFB7	CRBA	SAPCRB			
	X:$FFB8	CRCA	SAPCRC			
	X:$FFB9	SSISRA	SAPSR			
	X:$FFBA	RXA	SAPRX			
	X:$FFBB	TSRA	SAPTSR			
	X:$FFBC	TXA	SAPTX			
	X:$FFBD	PDRA	SAPPDR	5–0	PD[5:0]	SAPPD[5:0]
	X:$FFBE	PRRA	SAPDDR	5–0	PDC[5:0]	SAPDD[5:0]
	X:$FFBF	PCRA	SAPPCR	5–0	PC[5:0]	SAPPC[5:0]
BBP	X:$FFA4	RCRB	BBPRMR			
	X:$FFA5	TCRB	BBPTMR			
	X:$FFA6	CRAB	BBPCRA			
	X:$FFA7	CRBB	BBPCRB			
	X:$FFA8	CRCB	BBPCRC			
	X:$FFA9	SSISRB	BBPSR			
	X:$FFAA	RXB	BBPRX			
	X:$FFAB	TSRB	BBPTSR			
	X:$FFCC	TXB	BBPTX			
	X:$FFAD	PDRB	BBPPDR	5–0	PD[5:0]	BBPPD[5:0]
	X:$FFAE	PRRB	BBPDDR	5–0	PDC[5:0]	BBPDD[5:0]
	X:$FFAF	PCRB	BBPPCR	5–0	PC[5:0]	BBPPC[5:0]

Appendix E
Programmer’s Data Sheets

These programmer’s sheets are intended to simplify programming the various registers in the DSP56652. They can be photocopied and used to write in the binary bit values and the hexadecimal value for each register. The programmer’s sheets are provided in the same order as the sections in this document. Sheets are also provided for certain registers that are described in other documents. Table E-1 lists each programmer’s sheet, the register described in the sheet, and the page in this appendix where the sheet is located.

Table E-1. List of Programmer’s Sheets

Functional Block	Register		Page
	Acronym	Name	
MCU Configuration	RSR	Reset Source Register	E-6
	CKCTL	Clock Control Register	E-6
	GPCR	General Port Control Register	E-7
DSP Configuration	PCTL0	PLL Control Register 0	E-8
	PCTL1	PLL Control Register 1	E-8
	OMR	Operating Mode Register	E-9
	PATCH	Patch Registers	E-10
MDI	MCR	MCU-Side Control Register	E-11
	MCVR	MCU-Side Command Vector Register	E-11
	MSR	MCU-Side Status Register	E-12
	MRR0	MCU Receive Register 0	E-13
	MRR1	MCU Receive Register 1	E-13
	MTR0	MCU Transmit Register 0	E-13
	MTR1	MCU Transmit Register 1	E-13
	DCR	DSP-Side Control Register	E-14
	DSR	DSP-Side Status Register	E-15
	DRR0	DSP Receive Register 0	E-16
	DRR1	DSP Receive Register1	E-16
	DTR0	DSP Transmit Register 0	E-16
	DTR1	DSP Transmit Register 1	E-16

Table E-1. List of Programmer's Sheets (Continued)

Functional Block	Register		Page
	Acronym	Name	
EIM	CS0	Chip Select 0 Register	E-17
	CS1	Chip Select 1 Register	E-18
	CS2	Chip Select 2 Register	E-19
	CS3	Chip Select 3 Register	E-20
	CS4	Chip Select 4 Register	E-21
	CS5	Chip Select 5 Register	E-22
	EIMCR	EIM Configuration Register	E-23
Emulation Port	EPDDR	Emulation Port Data Direction Register	E-24
	EPDR	Emulation Port Data Register	E-24
Interrupts	ISR	Interrupt Source Register	E-25
	NIER	Normal Interrupt Enable Register	E-27
	FIER	Fast Interrupt Enable Register	E-29
	NIPR	Normal Interrupt Pending Register	E-31
	FIPR	Fast Interrupt Pending Register	E-33
	ICR	Interrupt Control Register	E-35
	IPRP	Interrupt Priority Register, Peripherals	E-36
	IPRC	Interrupt Priority Register, Core	E-37
Edge Port	EPPAR	Edge Port Pin Assignment Register	E-38
	EPDDR	Edge Port Data Direction Register	E-38
	EPDDR	Edge Port Data Register	E-38
	EPFR	Edge Port Flag Register	E-38
QSPI	SPCR	Serial Port Control Register	E-39
	QCR0	Queue Control Register 0	E-40
	QCR1	Queue Control Register 1	E-40
	QCR2	Queue Control Register 2	E-41
	QCR3	Queue Control Register 3	E-41
	SPSR	Serial Port Status Register	E-42
	SCCR0	Serial Channel Control Register 0	E-43
	SCCR1	Serial Channel Control Register 1	E-44
	SCCR2	Serial Channel Control Register 2	E-45
	SCCR3	Serial Channel Control Register 3	E-46
	SCCR4	Serial Channel Control Register 4	E-47
		QSPI Control RAM	E-48
	QPCR	QSPI Port Control Register	E-49
	QDDR	QSPI Data Direction Register	E-49
	QPDR	QSPI Port Data Register	E-49
Periodic Interrupt Timer	PITCSR	PIT Control and Status Register	E-50
	PITMR	PIT Modulus Register	E-50
	PITCNT	PIT Counter	E-50

Table E-1. List of Programmer's Sheets (Continued)

Functional Block	Register		Page
	Acronym	Name	
Watchdog Timer	WCR	Watchdog Control Register	E-51
	WSR	Watchdog Service Register	E-51
G-P Timer and PWM	TPWCR	Timers and PWM Control Register	E-52
	TPWMR	Timers and PWM Mode Register	E-53
	TPWSR	Timers and PWM Status Register	E-54
	TPWIR	Timers and PWM Interrupts Enable Register	E-55
	TOCR1	Timer 1 Output Compare Register	E-56
	TOCR3	Timer 3 Output Compare Register	E-56
	TOCR4	Timer 4 Output Compare Register	E-56
	TICR1	Timer 1 Input Capture Register	E-56
	TICR2	Timer 2 Input Capture Register	E-56
	PWOR	PWM Output Compare Register	E-57
	TCNT	Timer Count Register	E-57
	PWMR	PWM Modulus Register	E-57
	PWCNT	PWM Counter	E-57
Protocol Timer	PTCR	PT Control Register	E-58
	PTIER	PT Interrupt Enable Register	E-59
	PTSR	PT Status Register	E-60
	PTEVR	PT Event Register	E-61
	TIMR	Time Interval Modulus Register	E-61
	CTIC	Channel Time Interval Counter	E-61
	CTIMR	Channel Time Interval Modulus Register	E-62
	CFC	Channel Frame Counter	E-62
	CFMR	Channel Frame Modulus Register	E-62
	RSC	Reference Slot Counter	E-63
	RSMR	Reference Slot Modulus Register	E-63
	FTPTR	Frame Table Pointer	E-64
	MTPTR	Macro Table Pointer	E-64
	FTBAR	Frame Tables Base Address Register	E-65
	MTBAR	Macro Tables Base Address Register	E-65
	DTPTR	Delay Table Pointer	E-65
	PTPCR	PT Port Control Register	E-66
	PTDDR	PT Data Direction Register	E-66
	PTPDR	PT Port Data Register	E-66

Table E-1. List of Programmer's Sheets (Continued)

Functional Block	Register		Page
	Acronym	Name	
UART	URX	UART Receiver Register	E-67
	UTX	UART Transmitter Register	E-67
	UCR1	UART Control Register 1	E-68
	UCR2	UART Control Register 2	E-69
	UBRGR	UART Bit Rate Generator Register	E-69
	USR	UART Status Register	E-70
	UTS	UART Test Register	E-70
	UPCR	UART Port Control Register	E-71
	UDDR	UART Data Direction Register	E-71
	UPDR	UART Port Data Register	E-71
SCP	SCPCR	SCP Control Register	E-72
	SCACR	Smart Card Activation Control Register	E-73
	SCPIER	SCP Interrupt Enable Register	E-73
	SCPSR	SCP Status Register	E-74
	SCPDR	SCP Data Register	E-75
	SCPPCR	SCP Port Control Register	E-75
Keypad Port	KPCR	Keypad Port Control Register	E-76
	KPSR	Keypad Status Register	E-76
	KPDDR	Keypad Data Direction Register	E-77
	KPDR	Keypad Data Register	E-77
Serial Audio Port	SAPCNT	SAP Timer Counter	E-78
	SAPMR	SAP Timer Modulus Register	E-78
	SAPCRC	SAP Control Register A	E-78
	SAPCRB	SAP Control Register B	E-79
	SAPCRA	SAP Control Register C	E-80
	SAPSR	SAP Status Register	E-81
	SAPRX	SAP Receive Data Register	E-82
	SAPTSR	SAP Time Slot Register	E-82
	SAPTX	SAP Transmit Data Register	E-82
	SAPPCR	SAP Port Control Register	E-83
	SAPDDR	SAP GPIO Data Direction Register	E-83
	SAPPDR	SAP Port Data Register	E-83

Table E-1. List of Programmer's Sheets (Continued)

Functional Block	Register		Page
	Acronym	Name	
Baseband Port	BBPRMR	BBP Receive Counter Modulus Register	E-84
	BBPTMR	BBP Transmit Counter Modulus Register	E-84
	BBPCRA	BBP Control Register A	E-84
	BBPCRB	BBP Control Register B	E-85
	BBPCRC	BBP Control Register C	E-87
	BBPSR	BBP Status Register	E-88
	BBPRX	BBP Receive Data Register	E-89
	BBPTSR	BBP Time Slot Register	E-89
	BBPTX	BBP Transmit Data Register	E-89
	BBPPCR	BBP Port Control Register	E-90
	BBPDDR	BBP GPIO Direction Register	E-90
	BBPPDR	BBP Port Data Register	E-90

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

MCU Core

RSR

Reset Source Register
Address = $0020_C400
Reset value depends on cause of reset
Read Only

EXR	Description
0	The last reset was not caused by an external reset (assertion of RESET_IN pin)
1	The last reset was caused by RESET_IN assertion

WDR	Description
0	The last reset was not caused by Watchdog timer expiration
1	The last reset was caused by Watchdog timer expiration

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	*	*	*	*	*	*	WDR	EXR
0	0	0	0	0	0	0	0	0	0	0	0	0	0		
$0				$0				$0							

CKCTL

Clock Control Register
Address = $0020_C000
Reset = $0000
Read/Write

MCD[0:2]	Description
000	MCU clock division factor = 1
001	MCU clock division factor = 2
010	MCU clock division factor = 4
011	MCU clock division factor = 8
100	MCU clock division factor = 16
101-111	(Reserved)

CKOS	Description
0	MCU clock driven on CKO pin
1	DSP clock driven on CKO pin

MCS	Description
0	CKIL selected at multiplexer output
1	CKIH selected at multiplexer output

CKOD	Description
0	CKO pin enabled
1	CKO pin disabled

CKIHD	Description
0	CKIH input buffer enabled
1	CKIH input buffer disabled when MCS bit cleared

CKOHD	Description
0	CKOH output buffer enabled
1	CKOH output buffer disabled

DCS	Description
0	CKIH provided to DSP core
1	CKIL provided to DSP core

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	DCS	CKOHD	CKOD	CKOS	MCD2	MCD1	MCD0	MCS	CKIHD
0	0	0	0	0	0	0									
$0															

* = Reserved,

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

MCU Core

GPCR

General Port Control Register
Address = $0020_CC00
Reset = $0000
Read/Write

GPC5	Description
0	Pin G13 functions as ROW6
1	Pin G13 functions as SC2A

$\overline{\text{DTR}}$ accessed by configuring as output in KDDR

GPC6	Description
0	Pin G11 functions as ROW7
1	Pin G11 functions as SCKA

GPC7	Description
0	Pin E11 functions as $\overline{\text{RTS}}$
1	Pin E11 functions as IC2

STO bit value is reflected

GPC4	Description
0	Pin H14 functions as ROW5
1	Pin H14 functions as IC2

GPC3	Description
0	Pin M13 functions as COL7
1	Pin M13 functions as PWM

GPC2	Description
0	Pin N13 functions as COL6
1	Pin N13 functions as OC1

GPC1	Description
0	Pin J12 functions as INT7
1	Pin J12 functions as SRDA

$\overline{\text{DTR}}$ accessed by configuring as output in EPDDR

GPC0	Description
0	Pin K11 functions as INT6
1	Pin K11 functions as STDA

$\overline{\text{DSR}}$ accessed by configuring as output in EPDDR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STO	*	*	*	*	*	*	*	GPC7	GPC6	GPC5	GPC4	GPC3	GPC2	GPC1	GPC0
	0	0	0	0	0	0	0								
				$0											

* = Reserved,

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

DSP Core

PCTL0

PLL Control Register 0
Address = X:$FFFD
Reset = $0000

Predivider Factor

Multiplication Factor

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PD3	PD2	PD1	PD0	MF11	MF10	MF9	MF8	MF7	MF6	MF5	MF4	MF3	MF2	MF1	MF0

PCTL1

PLL Control Register 1
Address = X:$FFFC
Reset = $0000

PEN	Description
0	PLL disabled
1	PLL enabled

PSTP	Description
0	PLL disabled during STOP mode
1	PLL operates during STOP mode

Predivider Factor

Division Factor

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
* 0	* 0	* 0	* 0	PD6	PD5	PD4	* 0	* 0	PEN	PSTP	* 0	* 0	DF2	DF1	DF0
$0															

∗ = Reserved,

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

DSP Core

OMR

Operating Mode Register
Reset determined by hardware
Read/Write

XYS	Description
0	Stack Extension mapped to X memory
1	Stack Extension mapped to Y memory

Extended Stack
Underflow Flag

SD	Description
0	128 K clock cycle delay
1	16 clock cycle delay

SEN	Description
0	Stack Extension disabled
1	Stack Extension enabled

PCD	Description
0	PC relative instructions enabled
1	PC relative instructions disabled

ATE	Description
0	Address Trace disabled
1	Address Trace enabled

MB	Description
	Reflects state of DSP_IRQ at negation of $\overline{\text{RESET\_IN}}$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ATE	*	*	SEN	WRP	EOV	EUN	XYS	*	SD	PCD	*	*	*	MB	*
	0	0						0			0	0	0		1
EOM Extended Operating Mode Register								**COM Chip Operating Mode Register**							

* =

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

DSP Core

PAR0

Patch Register 0
Address = X:$FFF8
Reset = $uuuu

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PAR15	PAR14	PAR13	PAR12	PAR11	PAR10	PAR9	PAR8	PAR7	PAR6	PAR5	PAR4	PAR3	PAR2	PAR1	PAR0

PAR1

Patch Register 1
Address = X:$FFF7
Reset = $uuuu

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PAR15	PAR14	PAR13	PAR12	PAR11	PAR10	PAR9	PAR8	PAR7	PAR6	PAR5	PAR4	PAR3	PAR2	PAR1	PAR0

PAR2

Patch Register 2
Address = X:$FFF6
Reset = $uuuu

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PAR15	PAR14	PAR13	PAR12	PAR11	PAR10	PAR9	PAR8	PAR7	PAR6	PAR5	PAR4	PAR3	PAR2	PAR1	PAR0

PAR3

Patch Register 3
Address = X:$FFF5
Reset = $uuuu

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PAR15	PAR14	PAR13	PAR12	PAR11	PAR10	PAR9	PAR8	PAR7	PAR6	PAR5	PAR4	PAR3	PAR2	PAR1	PAR0

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

MCU MDI

MCR

MCU-Side Control Register
Address = $0020_2FF4
Reset = $0000
Read/Write

MTIE0	**Description**
0	Interrupt disabled
1	MCU Transmit Interrupt 0 enabled

MRIE1	**Description**
0	Interrupt disabled
1	MCU Receive Interrupt 1 enabled

MRIE0	**Description**
0	Interrupt disabled
1	MCU Receive Interrupt 0 enabled

MTIE1	**Description**
0	Interrupt disabled
1	MCU Transmit Interrupt 1 enabled

MGIE1	**Description**
0	Interrupt disabled
1	MCU General Interrupt 1 enabled

MGIE0	**Description**
0	Interrupt disabled
1	MCU General Interrupt 0 enabled

DHR	**Description**
0	No reset issued
1	Resets DSP

MDIR	**Description**
0	No reset issued
1	Resets MDI on MCU and DSP

MCU-to-DSP Flags

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MRIE0	MRIE1	MTIE0	MTIE1	MGIE0	MGIE1	*	*	DHR	MDIR	*	*	*	MDF2	MDF1	MDF0
						0	0			0	0	0			

MCVR

MCU-Side Command Vector Register
Address = $0020_2FF2
Reset = $0060
Read/Write

MC	**Description**
0	No interrupt issued
1	Sets MCP bit in $\overline{\text{DSR}}$

Command Vector Address

MNMI	**Description**
0	Command Interrupt is maskable
1	Command Interrupt is non-maskable

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	MC	MCV6	MCV5	MCV4	MCV3	MCV4	MCV1	MCV0	MNMI
0	0	0	0	0	0	0									
$0															

* = Reserved,

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

MCU MDI

MSR

MCU-Side Status Register
Address = $0020_2FF6
Reset = $3080
Read/Write

MGIP0	Description
0	No interrupt pending
1	MCU General Interrupt 0 pending

MTE1	Description
0	MTR1 has data
1	MTR1 is empty

MTE0	Description
0	MTR0 has data
1	MTR0 is empty

MRF1	Description
0	MRR1 is empty
1	MRR1 has data

MRF0	Description
0	MRR0 is empty
1	MRR0 has data

MGIP1	Description
0	No interrupt pending
1	MCU General Interrupt 1 pending

MTIR	Description
0	No interrupt pending
1	Protocol Timer DSP Interrupt pending

DWS	Description
0	No interrupt pending
1	$\overline{\text{IRQC}}$ asserted to awaken DSP from STOP mode

DRS	Description
0	DSP is not in RESET state
1	DSP currently in RESET state

MSMP	Description
0	No memory access pending
1	Shared memory access pending

DPM	Description
0	DSP is in Normal mode
1	DSP is in STOP mode

MEP	Description
0	No event pending
1	MCU-Side event pending

MCU-Side Flags

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MRF0	MRF1	MTE0	MTE1	MGIP0	MGIP1	MTIR	DWS	DRS	MSMP	DPM	MEP	*	MF2	MF1	MF0
												0			

* = Reserved,

Application: ____________________ Date: ____________________

____________________ Programmer: ____________________

MCU MDI

MRR0

MCU Receive Register 0
Address = $0020_2FFE
Reset = $uuuu
Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
data	data	data	data	data	data	data	data	data	data	data	data	data	data	data	data

MRR1

MCU Receive Register 1
Address = $0020_2FFC
Reset = $uuuu
Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
data	data	data	data	data	data	data	data	data	data	data	data	data	data	data	data

MTR0

MCU Transmit Register 0
Address = $0020_2FFA
Reset = $uuuu
Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
data	data	data	data	data	data	data	data	data	data	data	data	data	data	data	data

MTR1

MCU Transmit Register 1
Address = $0020_2FF8
Reset = $uuuu
Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
data	data	data	data	data	data	data	data	data	data	data	data	data	data	data	data

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

DSP MDI

DCR

DSP-Side Control Register
Address = X:$FF8A
Reset = $0000
Read/Write

DRIE0	Description
0	Interrupt disabled
1	DSP Receive Interrupt 0 enabled

DRIE1	Description
0	Interrupt disabled
1	DSP Receive Interrupt 1 enabled

DTIE1	Description
0	Interrupt disabled
1	DSP Transmit Interrupt 1 enabled

MCIE	Description
0	Interrupt disabled
1	MCU Command Interrupt enabled

DTIE0	Description
0	Interrupt disabled
1	DSP Transmit Interrupt 0 enabled

DSP-to-MCU Flags

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DTIE0	DTIE1	DRIE0	DRIE1	*	*	*	MCIE	*	*	*	*	*	DMF2	DMF1	DMF0
				0	0	0		0	0	0	0	0			
								$0							

* = Reserved,

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

DSP MDI

DSR

DSP-Side Status Register
Address = X:$FF8B
Reset = $C060
Read/Write

DGIR0	**Description**
0	No interrupt pending
1	MCU General Interrupt 0 pending (MGIP0 is set)

DRF1	**Description**
0	DRR1 is empty
1	DRR1 has data

DRF0	**Description**
0	DRR0 is empty
1	DRR0 has data

DTE1	**Description**
0	DTR1 has data
1	DTR1 is empty

DTE0	**Description**
0	DTR0 has data
1	DTR0 is empty

DGIR1	**Description**
0	No interrupt pending
1	MCU General Interrupt 1 pending (MGIP1 is set)

DTIC	**Description**
1	Signal to MCU to clear MTIR bit in MSR (bit 9) (write-only)

MCP	**Description**
0	No interrupt pending
1	MCU-Side Command interrupt pending

DWSC	**Description**
1	Signal to MCU to clear DWS bit in MSR (bit 8) (write-only)

MPM[0:1]	**Description**
00	MCU in STOP mode
01	MCU in WAIT mode
10	MCU in DOZE mode
11	MCU in Normal mode

DEP	**Description**
0	No event pending
1	DSP-Side event pending

DSP-Side Flags

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DTE0	DTE1	DRF0	DRF1	DGIR0	DGIR1	DTIC	MCP	DWSC	MPM1	MPM0	DEP	* 0	DF2	DF1	DF0

* = Reserved,

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

DSP MDI

DRR0

DSP Receive Register 0
Address = X:$FF8F
Reset = $uuuu
Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
data	data	data	data	data	data	data	data	data	data	data	data	data	data	data	data

DRR1

DSP Receive Register 1
Address = X:$FF8E
Reset = $uuuu
Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
data	data	data	data	data	data	data	data	data	data	data	data	data	data	data	data

DTR0

DSP Transmit Register 0
Address = X:$FF8D
Reset = $uuuu
Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
data	data	data	data	data	data	data	data	data	data	data	data	data	data	data	data

DTR1

DSP Transmit Register 1
Address = X:$FF8C
Reset = $uuuu
Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
data	data	data	data	data	data	data	data	data	data	data	data	data	data	data	data

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

EIM

CSCR0

Chip Select Register 0
Address = $0020_1000
Reset = $F861
Read/Write

OEA	Description
0	The $\overline{\text{OE}}$ signal is negated normally
1	The $\overline{\text{OE}}$ signal is asserted half a clock cycle later on read accesses

CSA	Description
0	The CS signal is asserted normally
1	The CS signal is asserted one cycle later on read and write accesses, and an extra cycle inserted between back-to-back cycles

EDC	Description
0	No delay occurs after a read cycle
1	One clock cycle is inserted after a read cycle

WWS	Description
0	Read and write WAIT states same
1	Write WAIT states = Read WAIT states + 1

WSC[0:3] Description
Binary value of number of external memory wait states

WEN	Description
0	The $\overline{\text{EB0–1}}$ signals are negated normally
1	The $\overline{\text{EB0–1}}$ signals are negated half a clock cycle earlier on write accesses

EBC	Description
0	Read and write accesses both assert $\overline{\text{EB0–1}}$
1	Only write accesses can assert $\overline{\text{EB0–1}}$

DSZ1	DSZ0	Description
0	0	8-bit port on D[8:15] pins
0	1	8-bit port on D[0:7] pins
1	0	16-bit port on D[0:15] pins
1	1	(Reserved)

SP	Description
0	User mode accesses allowed
1	User mode accesses prohibited

WP	Description
0	Writes are allowed
1	Writes are prohibited

CSEN	Description
0	Chip Select function is disabled, and $\overline{\text{CS0}}$ pin is an output
1	Chip Select function is enabled

31–16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	WSC3	WSC2	WSC1	WSC0	WWS	EDC	CSA	OEA	WEN	EBC	DSZ1	DSZ0	SP	WP	*	CSEN
0															0	
$0																

* = Reserved,

Application: ______________________________ Date: ____________________

______________________________ Programmer: ____________________

EIM

CSCR1

Chip Select Register 1
Address = $0020_1004
Reset = $uuuu
Read/Write

OEA	Description
0	The $\overline{OE}$ signal is negated normally
1	The $\overline{OE}$ signal is asserted half a clock cycle later on read accesses

CSA	Description
0	The CS signal is asserted normally
1	The CS signal is asserted one cycle later on read and write accesses, and an extra cycle inserted between back-to-back cycles

EDC	Description
0	No delay occurs after a read cycle
1	One clock cycle is inserted after a read cycle

WWS	Description
0	Read and write WAIT states same
1	Write WAIT states = Read WAIT states + 1

WSC[0:3] Description
Binary value of number of external memory wait states

WEN	Description
0	The $\overline{EB0\text{–}1}$ signals are negated normally
1	The $\overline{EB0\text{–}1}$ signals are negated half a clock cycle earlier on write accesses

EBC	Description
0	Read and write accesses both assert $\overline{EB0\text{–}1}$
1	Only write accesses can assert $\overline{EB0\text{–}1}$

DSZ1	DSZ0	Description
0	0	8-bit port on D[8:15] pins
0	1	8-bit port on D[0:7] pins
1	0	16-bit port on D[0:15] pins
1	1	(Reserved)

SP	Description
0	User mode accesses allowed
1	User mode accesses prohibited

WP	Description
0	Writes are allowed
1	Writes are prohibited

CSEN	Description
0	Chip Select function is disabled, and $\overline{CS0}$ pin is an output
1	Chip Select function is enabled

31–16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	WSC3	WSC2	WSC1	WSC0	WWS	EDC	CSA	OEA	WEN	EBC	DSZ1	DSZ0	SP	WP	*	CSEN
0															0	
$0																

* = Reserved,

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

EIM
CSCR2

Chip Select Register 2
Address = $0020_1008
Reset = $uuuu
Read/Write

WEN	Description
0	The EB0–1 signals are negated normally
1	The EB0–1 signals are negated half a clock cycle earlier on write accesses

OEA	Description
0	The OE signal is negated normally
1	The OE signal is asserted half a clock cycle later on read accesses

CSA	Description
0	The CS signal is asserted normally
1	The CS signal is asserted one cycle later on read and write accesses, and an extra cycle inserted between back-to-back cycles

EDC	Description
0	No delay occurs after a read cycle
1	One clock cycle is inserted after a read cycle

WWS	Description
0	Read and write WAIT states same
1	Write WAIT states = Read WAIT states + 1

WSC[0:3] Description
Binary value of number of external memory wait states

EBC	Description
0	Read and write accesses both assert EB0–1
1	Only write accesses can assert EB0–1

DSZ1	DSZ0	Description
0	0	8-bit port on D[8:15] pins
0	1	8-bit port on D[0:7] pins
1	0	16-bit port on D[0:15] pins
1	1	(Reserved)

SP	Description
0	User mode accesses allowed
1	User mode accesses prohibited

WP	Description
0	Writes are allowed
1	Writes are prohibited

PA	Description
0	CS pin at logic high
1	CS pin at logic low

CSEN	Description
0	Chip Select function is disabled, and CS0 pin is an output
1	Chip Select function is enabled

31–16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	WSC3	WSC2	WSC1	WSC0	WWS	EDC	CSA	OEA	WEN	EBC	DSZ1	DSZ0	SP	WP	PA	CSEN
0																
$0																

* = Reserved,

Application: ______________________________ Date: ________________

______________________________ Programmer: ________________

EIM
CSCR3

Chip Select Register 3
Address = $0020_100C
Reset = $uuuu
Read/Write

WEN	**Description**
0	The $\overline{\text{EB0–1}}$ signals are negated normally
1	The $\overline{\text{EB0–1}}$ signals are negated half a clock cycle earlier on write accesses

OEA	**Description**
0	The $\overline{\text{OE}}$ signal is negated normally
1	The $\overline{\text{OE}}$ signal is asserted half a clock cycle later on read accesses

CSA	**Description**
0	The CS signal is asserted normally
1	The CS signal is asserted one cycle later on read and write accesses, and an extra cycle inserted between back-to-back cycles

EDC	**Description**
0	No delay occurs after a read cycle
1	One clock cycle is inserted after a read cycle

WWS	**Description**
0	Read and write WAIT states same
1	Write WAIT states = Read WAIT states + 1

WSC[0:3] Description
Binary value of number of external memory wait states

EBC	**Description**
0	Read and write accesses both assert $\overline{\text{EB0–1}}$
1	Only write accesses can assert $\overline{\text{EB0–1}}$

DSZ1	**DSZ0**	**Description**
0	0	8-bit port on D[8:15] pins
0	1	8-bit port on D[0:7] pins
1	0	16-bit port on D[0:15] pins
1	1	(Reserved)

SP	**Description**
0	User mode accesses allowed
1	User mode accesses prohibited

WP	**Description**
0	Writes are allowed
1	Writes are prohibited

PA	**Description**
0	CS pin at logic high
1	CS pin at logic low

CSEN	**Description**
0	Chip Select function is disabled, and $\overline{\text{CS0}}$ pin is an output
1	Chip Select function is enabled

31–16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	WSC3	WSC2	WSC1	WSC0	WWS	EDC	CSA	OEA	WEN	EBC	DSZ1	DSZ0	SP	WP	PA	CSEN
0																
$0																

* = Reserved,

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

EIM
CSCR4

Chip Select Register 4
Address = $0020_1010
Reset = $uuuu
Read/Write

WEN	Description
0	The $\overline{\text{EB0–1}}$ signals are negated normally
1	The $\overline{\text{EB0–1}}$ signals are negated half a clock cycle earlier on write accesses

OEA	Description
0	The $\overline{\text{OE}}$ signal is negated normally
1	The $\overline{\text{OE}}$ signal is asserted half a clock cycle later on read accesses

CSA	Description
0	The CS signal is asserted normally
1	The CS signal is asserted one cycle later on read and write accesses, and an extra cycle inserted between back-to-back cycles

EDC	Description
0	No delay occurs after a read cycle
1	One clock cycle is inserted after a read cycle

WWS	Description
0	Read and write WAIT states same
1	Write WAIT states = Read WAIT states + 1

WSC[0:3] Description
Binary value of number of external memory wait states

EBC	Description
0	Read and write accesses both assert $\overline{\text{EB0–1}}$
1	Only write accesses can assert $\overline{\text{EB0–1}}$

DSZ1	DSZ0	Description
0	0	8-bit port on D[8:15] pins
0	1	8-bit port on D[0:7] pins
1	0	16-bit port on D[0:15] pins
1	1	(Reserved)

SP	Description
0	User mode accesses allowed
1	User mode accesses prohibited

WP	Description
0	Writes are allowed
1	Writes are prohibited

PA	Description
0	CS pin at logic high
1	CS pin at logic low

CSEN	Description
0	Chip Select function is disabled, and $\overline{\text{CS0}}$ pin is an output
1	Chip Select function is enabled

31–16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	WSC3	WSC2	WSC1	WSC0	WWS	EDC	CSA	OEA	WEN	EBC	DSZ1	DSZ0	SP	WP	PA	CSEN
0																
$0																

∗ = Reserved,

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

EIM
CSCR5

Chip Select Register 5
Address = $0020_1014
Reset = $uuuu
Read/Write

WEN	**Description**
0	The $\overline{\text{EB0–1}}$ signals are negated normally
1	The $\overline{\text{EB0–1}}$ signals are negated half a clock cycle earlier on write accesses

OEA	**Description**
0	The $\overline{\text{OE}}$ signal is negated normally
1	The $\overline{\text{OE}}$ signal is asserted half a clock cycle later on read accesses

CSA	**Description**
0	The CS signal is asserted normally
1	The CS signal is asserted one cycle later on read and write accesses, and an extra cycle inserted between back-to-back cycles

EDC	**Description**
0	No delay occurs after a read cycle
1	One clock cycle is inserted after a read cycle

WWS	**Description**
0	Read and write WAIT states same
1	Write WAIT states = Read WAIT states + 1

WSC[0:3] Description
Binary value of number of external memory wait states

EBC	**Description**
0	Read and write accesses both assert $\overline{\text{EB0–1}}$
1	Only write accesses can assert $\overline{\text{EB0–1}}$

DSZ1	**DSZ0**	**Description**
0	0	8-bit port on D[8:15] pins
0	1	8-bit port on D[0:7] pins
1	0	16-bit port on D[0:15] pins
1	1	(Reserved)

SP	**Description**
0	User mode accesses allowed
1	User mode accesses prohibited

WP	**Description**
0	Writes are allowed
1	Writes are prohibited

PA	**Description**
0	CS pin at logic high
1	CS pin at logic low

CSEN	**Description**
0	Chip Select function is disabled, and $\overline{\text{CS0}}$ pin is an output
1	Chip Select function is enabled

31–16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
* 0	WSC3	WSC2	WSC1	WSC0	WWS	EDC	CSA	OEA	WEN	EBC	DSZ1	DSZ0	SP	WP	PA	CSEN
$0																

* = Reserved,

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

EIM

EIMCR

EIM Configuration Register
Address = $0020_1018
Reset = $0038
Read/Write

SPROM	Description
0	User mode access to internal ROM is allowed
1	User mode access to internal ROM is prohibited. Only Supervisor access is allowed

SPRAM	Description
0	User mode access to internal RAM is allowed
1	User mode access to internal RAM is prohibited. Only Supervisor access is allowed

HDB	Description
0	Lower data bus D[0:15] driven externally
1	Upper data bus D[16:31] driven externally

SHEN1	SHEN0	Description
0	0	Show cycles disabled
0	1	Show cycles enabled, transfers during EDC/CSA idle cycles not visible externally
1	0	Show cycles enabled, all transfers visible (causes performance loss)
1	1	(Reserved)

SPIPER	Description
0	User mode access to peripherals is allowed
1	User mode access to internal peripherals is prohibited. Only Supervisor access is allowed

EPEN	Description
0	Emulation port pins configured as GPIO
1	Emulation port pins configured as SIZ[0:1] and PSTAT[0:3]

31–16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	*	*	EPEN	SPIPER	SPRAM	SPROM	HDB	SHEN1	SHEN0
0	0		0	0	0	0	0	0	0							
$0	$0				$0											

* = Reserved,

Application: ______________________ Date: ______________

______________________ Programmer: ______________

EIM

EMDDR

Emulation Port Data Direction Register
Address = $0020_C800
Reset = $0000
Read/Write

EMDDn	Description
0	Pin is GPIO input
1	Pin is GPIO output

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	*	*	EMDD5	EMDD4	EMDD3	EMDD2	EMDD1	EMDD0
0	0	0	0	0	0	0	0	0	0						
$0				$0											

EMDR

Emulation Port Data Register
Address = $0020_C802
Reset = $0000
Read/Write

Port Data Bits

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	*	*	EMD5	EMD4	EMD3	EMD2	EMD1	EMD0
0	0	0	0	0	0	0	0	0	0						
$0				$0											

* = Reserved,

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

MCU Interrupts

ISR

Upper Halfword

Interrupt Souce Register
Upper Halfword
Address = $0020_0000
Reset = $0000
Read/Write

PT1	Description
0	No interrupt request
1	Protocol Timer MCU1 interrupt request pending

PT2	Description
0	No interrupt request
1	Protocol Timer MCU2 interrupt request pending

UTX	Description
0	No interrupt request
1	UART Transmitter Ready interrupt request pending

SMPD	Description
0	No interrupt request
1	SIM Auto Power Down interrupt request pending

URX	Description
0	No interrupt request
1	UART Receiver Ready interrupt request pending

PT0	Description
0	No interrupt request
1	Protocol Timer MCU0 interrupt request pending

PTM	Description
0	No interrupt request
1	Protocol Timer interrupt request pending

QSPI	Description
0	No interrupt request
1	QSPI interrupt request pending

MDI	Description
0	No interrupt request
1	MDI interrupt request pending

SCP	Description
0	No interrupt request
1	SIM Card Tx, Rx, or Error interrupt request pending

TPW	Description
0	No interrupt request
1	General Purpose Timer/PWM interrupt request pending

PIT	Description
0	No interrupt request
1	Periodic Interrupt Timer interrupt request pending

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
URX	SMPD	UTX	PT2	PT1	PT0	PTM	QSPI	MDI	SCP	*	*	*	*	TPW	PIT
										0	0	0	0		

* = Reserved,

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

MCU Interrupts

ISR

Lower Halfword

Interrupt Souce Register
Lower Halfword
Address = $0020_0002
Reset = $0007
Read/Write

INT5	Description
0	No interrupt request
1	INT5 interrupt request pending

INT4	Description
0	No interrupt request
1	INT4 interrupt request pending

INT3	Description
0	No interrupt request
1	INT3 interrupt request pending

INT2	Description
0	No interrupt request
1	INT2 interrupt request pending

INT1	Description
0	No interrupt request
1	INT1 interrupt request pending

INT0	Description
0	No interrupt request
1	INT0 interrupt request pending

INT6	Description
0	No interrupt request
1	INT6 interrupt request pending

INT7	Description
0	No interrupt request
1	INT7 interrupt request pending

URTS	Description
0	No interrupt request
1	UART RTS Delta interrupt request pending

KPD	Description
0	No interrupt request
1	Keypad Interface interrupt request pending

Software Interrupt

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	KPD	URTS	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0	*	*	S2	S1	S0
0											0	0			

* = Reserved,

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

MCU Interrupts
NIER
Upper Halfword

Normal Interrupt Enable Register
Upper Halfword
Address = $0020_0004
Reset = $0000
Read/Write

EPT1	Description
0	Interrupt source is masked
1	Protocol Timer MCU1 interrupt source enabled

EPT2	Description
0	Interrupt source is masked
1	Protocol Timer MCU2 interrupt source enabled

EUTX	Description
0	Interrupt source is masked
1	UART Transmitter Ready interrupt source enabled

ESMPD	Description
0	Interrupt source is masked
1	SIM Auto Power Down interrupt source enabled

EURX	Description
0	Interrupt source is masked
1	UART Receiver Ready interrupt source enabled

EPT0	Description
0	Interrupt source is masked
1	Protocol Timer MCU0 interrupt source enabled

EPTM	Description
0	Interrupt source is masked
1	Protocol Timer interrupt source enabled

EQSPI	Description
0	Interrupt source is masked
1	QSPI interrupt source enabled

EMDI	Description
0	Interrupt source is masked
1	MDI interrupt source enabled

ESCP	Description
0	Interrupt source is masked
1	SIM Card Tx, Rx, or Error interrupt source enabled

ETPW	Description
0	Interrupt source is masked
1	General Purpose Timer/PWM interrupt source enabled

EPIT	Description
0	Interrupt source is masked
1	Periodic Interrupt Timer interrupt source enabled

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EURX	ESMPD	EUTX	EPT2	EPT1	EPT0	EPTM	EQSPI	EMDI	ESCP	* 0	* 0	* 0	* 0	ETPW	EPIT

NOTE:NIER can only be written as a 32-bit * = Reserved,

Application: ____________________ Date: ____________

____________________ Programmer: ____________

MCU Interrupts

NIER

Lower Halfword

Normal Interrupt Enable Register
Lower Halfword
Reset = $0000
Read/Write

EINT5	Description
0	Interrupt source is masked
1	INT5 interrupt source enabled

EINT6	Description
0	Interrupt source is masked
1	INT6 interrupt source enabled

EINT7	Description
0	Interrupt source is masked
1	INT7 interrupt source enabled

EURTS	Description
0	Interrupt source is masked
1	UART RTS Delta interrupt source enabled

EKPD	Description
0	Interrupt source is masked
1	Keypad Interface interrupt source enabled

EINT4	Description
0	Interrupt source is masked
1	INT4 interrupt source enabled

EINT3	Description
0	Interrupt source is masked
1	INT3 interrupt source enabled

EINT2	Description
0	Interrupt source is masked
1	INT2 interrupt source enabled

EINT1	Description
0	Interrupt source is masked
1	INT1 interrupt source enabled

EINT0	Description
0	Interrupt source is masked
1	INT0 interrupt source enabled

ES2	Description
0	Interrupt source is masked
1	Software Interrupt 2 source enabled

ES1	Description
0	Interrupt source is masked
1	Software Interrupt 1 source enabled

ES0	Description
0	Interrupt source is masked
1	Software Interrupt 0 source enabled

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
* 0	EKPD	EURTS	EINT7	EINT6	EINT5	EINT4	EINT3	EINT2	EINT1	EINT0	* 0	* 0	ES2	ES1	ES0

NOTE:NIER can only be written as a 32-bit * = Reserved,

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

MCU Interrupts
FIER
Upper Halfword

Fast Interrupt Enable Register
Upper Halfword
Address = $0020_0008
Reset = $0000
Read/Write

EFPT1	Description
0	Interrupt source is masked
1	Protocol Timer MCU1 interrupt source enabled

EFPT2	Description
0	Interrupt source is masked
1	Protocol Timer MCU2 interrupt source enabled

EFUTX	Description
0	Interrupt source is masked
1	UART Transmitter Ready interrupt source enabled

EFSMPD	Description
0	Interrupt source is masked
1	SIM Auto Power Down interrupt source enabled

EFURX	Description
0	Interrupt source is masked
1	UART Receiver Ready interrupt source enabled

EFPT0	Description
0	Interrupt source is masked
1	Protocol Timer MCU0 interrupt source enabled

EFPTM	Description
0	Interrupt source is masked
1	Protocol Timer interrupt source enabled

EFQSPI	Description
0	Interrupt source is masked
1	QSPI interrupt source enabled

EFMDI	Description
0	Interrupt source is masked
1	MDI interrupt source enabled

EFSCP	Description
0	Interrupt source is masked
1	SIM Card Tx, Rx, or Error interrupt source enabled

EFTPW	Description
0	Interrupt source is masked
1	General Purpose Timer/PWM interrupt source enabled

EFPIT	Description
0	Interrupt source is masked
1	Periodic Interrupt Timer interrupt source enabled

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EFURX	EFSMPD	EFUTX	EFPT2	EFPT1	EFPT0	EFPTM	EFQSPI	EFMDI	EFSCP	* 0	* 0	* 0	* 0	EFTPW	EFPIT

NOTE:FIER can only be written as a 32-bit * = Reserved,

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

MCU Interrupts
FIER
Lower Halfword

Fast Interrupt Enable Register
Lower Halfword
Reset = $0000
Read/Write

EFINT5	**Description**
0	Interrupt source is masked
1	INT5 interrupt source enabled

EFINT6	**Description**
0	Interrupt source is masked
1	INT6 interrupt source enabled

EFINT7	**Description**
0	Interrupt source is masked
1	INT7 interrupt source enabled

EFURTS	**Description**
0	Interrupt source is masked
1	UART RTS Delta interrupt source enabled

EFKPD	**Description**
0	Interrupt source is masked
1	Keypad Interface interrupt source enabled

EFINT4	**Description**
0	Interrupt source is masked
1	INT4 interrupt source enabled

EFINT3	**Description**
0	Interrupt source is masked
1	INT3 interrupt source enabled

EFINT2	**Description**
0	Interrupt source is masked
1	INT2 interrupt source enabled

EFINT1	**Description**
0	Interrupt source is masked
1	INT1 interrupt source enabled

EFINT0	**Description**
0	Interrupt source is masked
1	INT0 interrupt source enabled

EFS2	**Description**
0	Interrupt source is masked
1	Software Interrupt 2 source enabled

EFS1	**Description**
0	Interrupt source is masked
1	Software Interrupt 1 source enabled

EFS0	**Description**
0	Interrupt source is masked
1	Software Interrupt 0 source enabled

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
* 0	EFKPD	EFURTS	EFINT7	EFINT6	EFINT5	EFINT4	EFINT3	EFINT2	EFINT1	EFINT0	* 0	* 0	EFS2	EFS1	EFS0

NOTE:FIER can only be written as a 32-bit * = Reserved,

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

MCU Interrupts
NIPR
Upper Halfword

Normal Interrupt Pending Register
Upper Halfword
Address = $0020_000C
Reset = $0000
Read/Write

NPT1	Description
0	No interrupt pending
1	Protocol Timer MCU1 interrupt request pending

NPT2	Description
0	No interrupt pending
1	Protocol Timer MCU2 interrupt request pending

NUTX	Description
0	No interrupt pending
1	UART Transmitter Ready interrupt request pending

NSMPD	Description
0	No interrupt pending
1	SIM Auto Power Down interrupt request pending

NURX	Description
0	No interrupt pending
1	UART Receiver Ready interrupt request pending

NPT0	Description
0	No interrupt pending
1	Protocol Timer MCU0 interrupt request pending

NPTM	Description
0	No interrupt pending
1	Protocol Timer interrupt request pending

NQSPI	Description
0	No interrupt pending
1	QSPI interrupt request pending

NMDI	Description
0	No interrupt pending
1	MDI interrupt request pending

NSCP	Description
0	No interrupt pending
1	SIM Card Tx, Rx, or Error interrupt request pending

NTPW	Description
0	No interrupt pending
1	General Purpose Timer/PWM interrupt request pending

NPIT	Description
0	No interrupt pending
1	Periodic Interrupt Timer interrupt request pending

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
NURX	NSMPD	NUTX	NPT2	NPT1	NPT0	NPTM	NQSPI	NMDI	NSCP	*	*	*	*	NTPW	NPIT
										0	0	0	0		

NOTE:NIPR can only be written as a 32-bit * = Reserved,

Application: ______________________________ Date: ________________

______________________________ Programmer: ________________

MCU Interrupts
NIPR
Lower Halfword

Normal Interrupt Pending Register
Lower Halfword
Address = $0020_000E
Reset = $0000
Read/Write

NINT5	Description
0	No interrupt pending
1	INT5 interrupt request pending

NINT6	Description
0	No interrupt pending
1	INT6 interrupt request pending

NINT7	Description
0	No interrupt pending
1	INT7 interrupt request pending

NURTS	Description
0	No interrupt pending
1	UART RTS Delta interrupt request pending

NKPD	Description
0	No interrupt pending
1	Keypad Interface interrupt request pending

NINT4	Description
0	No interrupt pending
1	INT4 interrupt request pending

NINT3	Description
0	No interrupt pending
1	INT3 interrupt request pending

NINT2	Description
0	No interrupt pending
1	INT2 interrupt request pending

NINT1	Description
0	No interrupt pending
1	INT1 interrupt request pending

NINT0	Description
0	No interrupt pending
1	INT0 interrupt request pending

NS2	Description
0	No interrupt pending
1	Software Interrupt 2 request pending

NS1	Description
0	No interrupt pending
1	Software Interrupt 1 request pending

NS0	Description
0	No interrupt pending
1	Software Interrupt 0 request pending

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
* 0	NKPD	NURTS	NINT7	NINT6	NINT5	NINT4	NINT3	NINT2	NINT1	NINT0	* 0	* 0	NS2	NS1	NS0

NOTE:NIPR can only be written as a 32-bit * = Reserved,

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

MCU Interrupts
FIPR
Upper Halfword

Fast Interrupt Pending Register
Upper Halfword
Address = $0020_0010
Reset = $0000
Read/Write

FPT1	Description
0	No interrupt pending
1	Protocol Timer MCU1 interrupt request pending

FPT2	Description
0	No interrupt pending
1	Protocol Timer MCU2 interrupt request pending

FUTX	Description
0	No interrupt pending
1	UART Transmitter Ready interrupt request pending

FSMPD	Description
0	No interrupt pending
1	SIM Auto Power Down interrupt request pending

FURX	Description
0	No interrupt pending
1	UART Receiver Ready interrupt request pending

FPT0	Description
0	No interrupt pending
1	Protocol Timer MCU0 interrupt request pending

FPTM	Description
0	No interrupt pending
1	Protocol Timer interrupt request pending

FQSPI	Description
0	No interrupt pending
1	QSPI interrupt request pending

FMDI	Description
0	No interrupt pending
1	MDI interrupt request pending

FSCP	Description
0	No interrupt pending
1	SIM Card Tx, Rx, or Error interrupt request pending

FTPW	Description
0	No interrupt pending
1	General Purpose Timer/PWM interrupt request pending

FPIT	Description
0	No interrupt pending
1	Periodic Interrupt Timer interrupt request pending

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
FURX	FSMPD	FUTX	FPT2	FPT1	FPT0	FPTM	FQSPI	FMDI	FSCP	*	*	*	*	FTPW	FPIT
										0	0	0	0		

NOTE:FIPR can only be written as a 32-bit * = Reserved,

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

MCU Interrupts
FIPR
Lower Halfword

Fast Interrupt Pending Register
Lower Halfword
Address = $0020_0012
Reset = $0000
Read/Write

FINT5	Description
0	No interrupt pending
1	INT5 interrupt request pending

FINT6	Description
0	No interrupt pending
1	INT6 interrupt request pending

FINT7	Description
0	No interrupt pending
1	INT7 interrupt request pending

FURTS	Description
0	No interrupt pending
1	UART RTS Delta interrupt request pending

FKPD	Description
0	No interrupt pending
1	Keypad Interface interrupt request pending

FINT4	Description
0	No interrupt pending
1	INT4 interrupt request pending

FINT3	Description
0	No interrupt pending
1	INT3 interrupt request pending

FINT2	Description
0	No interrupt pending
1	INT2 interrupt request pending

FINT1	Description
0	No interrupt pending
1	INT1 interrupt request pending

FINT0	Description
0	No interrupt pending
1	INT0 interrupt request pending

FS2	Description
0	No interrupt pending
1	Software Interrupt 2 request pending

FS1	Description
0	No interrupt pending
1	Software Interrupt 1 request pending

FS0	Description
0	No interrupt pending
1	Software Interrupt 0 request pending

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
* 0	FKPD	FURTS	FINT7	FINT6	FINT5	FINT4	FINT3	FINT2	FINT1	FINT0	* 0	* 0	FS2	FS1	FS0

NOTE:FIPR can only be written as a 32-bit * = Reserved,

Application: ______________________________ Date: ________________

______________________________ Programmer: ________________

MCU Interrupts

ICR

Upper Halfword

Interrupt Control Register
Upper Halfword
Address = $0020_0014
Reset = $0000
Read/Write
Accessible Only in Supervisor Mode

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$0				$0				$0				$0			

ICR

Lower Halfword

Interrupt Control Register
Lower Halfword
Reset = $0000
Read/Write
Accessible Only in Supervisor Mode

Source Number

Vector Number

EN	Description
0	Priority hardware disabled
1	Priority hardware disabled

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EN	*	*	*	SRC4	SRC3	SRC2	SRC1	SRC0	VEC6	VEC5	VEC4	VEC3	VEC2	VEC1	VEC0
	0	0	0												

NOTE:ICR can only be written as a 32-bit

* = Reserved,

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

DSP Interrupts

IPRP

Interrupt Priority Register, Peripheral
Address = X:$FFFE
Reset = $0000
Read/Write

SAP IPL		
PL1	**PL0**	**Mode**
0	0	Interrupts disabled
0	1	Interrupts enabled, IPL = 0
1	0	Interrupts enabled, IPL = 1
1	1	Interrupts enabled, IPL = 2

BBP IPL		
PL1	**PL0**	**Mode**
0	0	Interrupts disabled
0	1	Interrupts enabled, IPL = 0
1	0	Interrupts enabled, IPL = 1
1	1	Interrupts enabled, IPL = 2

Protocol Timer IPL		
PL1	**PL0**	**Mode**
0	0	Interrupts disabled
0	1	Interrupts enabled, IPL = 0
1	0	Interrupts enabled, IPL = 1
1	1	Interrupts enabled, IPL = 2

MCU Default Command IPL		
PL1	**PL0**	**Mode**
0	0	Interrupts disabled
0	1	Interrupts enabled, IPL = 0
1	0	Interrupts enabled, IPL = 1
1	1	Interrupts enabled, IPL = 2

MDI IPL		
PL1	**PL0**	**Mode**
0	0	Interrupts disabled
0	1	Interrupts enabled, IPL = 0
1	0	Interrupts enabled, IPL = 1
1	1	Interrupts enabled, IPL = 2

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
* 0	* 0	* 0	* 0	* 0	* 0	MDIPL1	MDIPL0	PTPL1	PTPL0	SAPPL1	SAPPL0	BBPPL1	BBPPL0	MDCPL1	MDCPL0
$0															

* = Reserved,

Application: ______________________________ Date: ____________________

______________________________ Programmer: ____________________

DSP Interrupts

IPRC

Interrupt Priority Register, Core
Address = X:$FFFF
Reset = $0000
Read/Write

IATM	IAPL1	IAPL0	IRQ A Mode	Trigger Mode
0	0	0	IRQ A disabled, no IPL	Level-sensitive
0	0	1	IRQ A enabled, IPL = 0	Level-sensitive
0	1	0	IRQ A enabled, IPL = 1	Level-sensitive
0	1	1	IRQ A enabled, IPL = 2	Level-sensitive
1	0	0	IRQ A disabled, no IPL	Edge-sensitive
1	0	1	IRQ A enabled, IPL = 0	Edge-sensitive
1	1	0	IRQ A enabled, IPL = 1	Edge-sensitive
1	1	1	IRQ A enabled, IPL = 2	Edge-sensitive

IBTM	IBPL1	IBPL0	IRQ B Mode	Trigger Mode
0	0	0	IRQ B disabled, no IPL	Level-sensitive
0	0	1	IRQ B enabled, IPL = 0	Level-sensitive
0	1	0	IRQ B enabled, IPL = 1	Level-sensitive
0	1	1	IRQ B enabled, IPL = 2	Level-sensitive
1	0	0	IRQ B disabled, no IPL	Edge-sensitive
1	0	1	IRQ B enabled, IPL = 0	Edge-sensitive
1	1	0	IRQ B enabled, IPL = 1	Edge-sensitive
1	1	1	IRQ B enabled, IPL = 2	Edge-sensitive

ICTM	ICPL1	ICPL0	IRQ C Mode	Trigger Mode
0	0	0	IRQ C disabled, no IPL	Level-sensitive
0	0	1	IRQ C enabled, IPL = 0	Level-sensitive
0	1	0	IRQ C enabled, IPL = 1	Level-sensitive
0	1	1	IRQ C enabled, IPL = 2	Level-sensitive
1	0	0	IRQ C disabled, no IPL	Edge-sensitive
1	0	1	IRQ C enabled, IPL = 0	Edge-sensitive
1	1	0	IRQ C enabled, IPL = 1	Edge-sensitive
1	1	1	IRQ C enabled, IPL = 2	Edge-sensitive

IDTM	IDPL1	IDPL0	IRQ D Mode	Trigger Mode
0	0	0	IRQ D disabled, no IPL	Level-sensitive
0	0	1	IRQ D enabled, IPL = 0	Level-sensitive
0	1	0	IRQ D enabled, IPL = 1	Level-sensitive
0	1	1	IRQ D enabled, IPL = 2	Level-sensitive
1	0	0	IRQ D disabled, no IPL	Edge-sensitive
1	0	1	IRQ D enabled, IPL = 0	Edge-sensitive
1	1	0	IRQ D enabled, IPL = 1	Edge-sensitive
1	1	1	IRQ D enabled, IPL = 2	Edge-sensitive

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
* 0	* 0	* 0	* 0	IDTM	IDPL1	IDPL0	ICTM	ICPL1	ICPL0	IBTM	IBPL1	IBPL0	IATM	IAPL1	IAPL0
$0															

∗ = Reserved,

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

Edge Port

EPPAR

Edge Port Pin Assignment Register
Address = $0020_9000
Reset = $0000
Read/Write

EPPAn	Description
00	Pin INTn is level-sensitive
01	Pin INTn defined as rising-edge detect
10	Pin INTn defined as falling-edge detect
11	Pin INTn defined as both rising- and falling-edge detect

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EPPA7		EPPA6		EPPA5		EPPA4		EPPA3		EPPA2		EPPA1		EPPA0	

EPDDR

Edge Port Data Direction Register
Address = $0020_9002
Reset = $0000
Read/Write

EPDDn	Description
0	Pin is input
1	Pin is output

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	EPDD7	EPDD6	EPDD5	EPDD4	EPDD3	EPDD2	EPDD1	EPDD0
0	0	0	0	0	0	0	0								
$0				$0											

EPDR

Edge Port Data Register
Address = $0020_9004
Reset = $00uu
Read/Write

Port Data Bits

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	EPD7	EPD6	EPD5	EPD4	EPD3	EPD2	EPD1	EPD0
0	0	0	0	0	0	0	0								
$0				$0											

EPFR

Edge Port Flag Register
Address = $0020_9006
Reset = $0000
Read/Write

Edge Port Flags

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	EPF7	EPF6	EPF5	EPF4	EPF3	EPF2	EPF1	EPF0
0	0	0	0	0	0	0	0								
$0				$0											

* = Reserved,

Application: ______________________________ Date: ____________________

______________________________ Programmer: ____________________

QSPI

SPCR

Serial Port Control Register
Address = $0020_5F06
Reset = $0000
Read/Write

WIE	Description
0	Queue wraparounds do not cause hardware interrupts from QSPI to MCU
1	Queue wraparounds (QPWF flag set) cause hardware interrupts from QSPI to MCU

TACE	Description
0	Trigger accumulation for Queue 1 is disabled
1	Trigger accumulation for Queue 1 is enabled. Queues 0, 2, and 3 are unaffected

TRCIE	Description
0	Trigger collisions do not cause hardware interrupts from QSPI to MCU
1	Trigger collisions cause hardware interrupts from QSPI to MCU

HALT	Description
0	QSPI HALT is disabled
1	QSPI HALT is requested

HLTIE	Description
0	Hardware interrupts from QSPI to MCU caused by HALTA flag are disabled
1	Hardware interrupts from QSPI to MCU caused by HALTA flag are enabled

DOZE	Description
0	QSPI ignores DOZE mode
1	DOZE mode causes QSPI to halt at end of executing queue

QEn	Description
0	Queue n triggering is inactive
1	Queue n triggering is active on MCU or Protocol Timer triggers

QSPE	Description
0	QSPI disabled
1	QSPI enabled

CSPOLn	Description
0	SPICSn is active low
1	SPICSn is active high

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSPOL4	CSPOL3	CSPOL2	CSPOL1	CSPOL0	QE3	QE2	QE1	QE0	HLTIE	TRCIE	WIE	TACE	HALT	DOZE	QSPE

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

QSPI

QCR0

Queue Control Register 0
Address = $0020_5F08
Reset = $0000
Read/Write

HMD0	**Description**
0	Queue 0 halts only at end of queue
1	Queue 0 halts on any sub-queue boundary

Queue 0 Pointer

LE0	**Description**
0	Queue 0 reloading disabled
1	Queue 0 reloading enabled

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LE0	HMD0	*	*	*	*	*	*	*	*	QP05	QP04	QP03	QP02	QP01	QP00
		0	0	0	0	0	0	0	0						
				$0											

QCR1

Queue Control Register 1
Address = $0020_5F0A
Reset = $0000
Read/Write

HMD1	**Description**
0	Queue 1 halts only at end of queue
1	Queue 1 halts on any sub-queue boundary

Trigger Counter

Queue 1 Pointer

LE1	**Description**
0	Queue 1 reloading disabled
1	Queue 1 reloading enabled

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LE1	HMD1	*	*	*	*	TRCNT3	TRCNT2	TRCNT1	TRCNT0	QP15	QP14	QP13	QP12	QP11	QP10
		0	0	0	0										

* = Reserved,

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

QSPI

QCR2

Queue Control Register 2
Address = $0020_5F0C
Reset = $0000
Read/Write

HMD2	Description
0	Queue 2 halts only at end of queue
1	Queue 2 halts on any sub-queue boundary

Queue 2 Pointer

LE2	Description
0	Queue 2 reloading disabled
1	Queue 2 reloading enabled

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LE2	HMD2	*	*	*	*	*	*	*	*	QP25	QP24	QP23	QP22	QP21	QP20
		0	0	0	0	0	0	0	0						
				$0											

QCR3

Queue Control Register 3
Address = $0020_5F0E
Reset = $0000
Read/Write

HMD3	Description
0	Queue 3 halts only at end of queue
1	Queue 3 halts on any sub-queue boundary

Queue 3 Pointer

LE3	Description
0	Queue 3 reloading disabled
1	Queue 3 reloading enabled

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LE3	HMD3	*	*	*	*	*	*	*	*	QP35	QP34	QP33	QP32	QP31	QP30
		0	0	0	0	0	0	0	0						
				$0											

* = Reserved,

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

QSPI

SPSR

Serial Port Status Register
Address = $0020_5F10
Reset = $0000
Read/Write

QA2	Description
0	Queue 2 is not active
1	Queue 2 is active

QA3	Description
0	Queue 3 is not active
1	Queue 3 is active

QX0	Description
0	Queue 0 is not executing
1	Queue 0 is executing

QX1	Description
0	Queue 1 is not executing
1	Queue 1 is executing

QX2	Description
0	Queue 2 is not executing
1	Queue 2 is executing

QX3	Description
0	Queue 3 is not executing
1	Queue 3 is executing

QA1	Description
0	Queue 1 is not active
1	Queue 1 is active

QA0	Description
0	Queue 0 is not active
1	Queue 0 is active

HALTA	Description
0	QSPI has not halted
1	QSPI has halted

TRC	Description
0	Trigger collision has not occurred
1	Trigger collision has occurred

QPWF	Description
0	Queue pointer has not wrapped around
1	Queue pointer has wrapped around

EOT3	Description
0	Queue 3 transfer not complete
1	Queue 3 transfer complete

EOT2	Description
0	Queue 2 transfer not complete
1	Queue 2 transfer complete

EOT1	Description
0	Queue 1 transfer not complete
1	Queue 1 transfer complete

EOT0	Description
0	Queue 0 transfer not complete
1	Queue 0 transfer complete

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
QX3	QX2	QX1	QX0	QA3	QA2	QA1	QA0	* 0	HALTA	TRC	QPWF	EOT3	EOT2	EOT1	EOT0

* = Reserved,

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

QSPI

SCCR0

Serial Channel Control Register 0
Address = $0020_5F12
Reset = $0000
Read/Write

DATR0[0:2]	Delay After Transfer
000	1 SCK cycle delay
001	2 SCK cycles delay
010	4 SCK cycles delay
011	8 SCK cycles delay
100	16 SCK cycles delay
101	32 SCK cycles delay
110	64 SCK cycles delay
111	128 SCK cycles delay

LSBF0	Description
0	Data transferred MSB first
1	Data transferred LSB first

CKPOL0	Description
0	SCK inactive at logic 1
1	SCK inactive at logic 0

CPHA0	Description
0	Data changes on first SCK transition
1	Data latches on first SCK transition

CSCKDF0[0:2]	Assertion to Activation Delay
000	1 SCK cycle delay
001	2 SCK cycles delay
010	4 SCK cycles delay
011	8 SCK cycles delay
100	16 SCK cycles delay
101	32 SCK cycles delay
110	64 SCK cycles delay
111	128 SCK cycles delay

$$\mathbf{SCK} = \frac{\mathbf{MCU\_CLK}}{\mathbf{2\cdot\{3(SCKFD0[6]+1)\cdot(SCKDF0[0:5]+1)\}}}$$

All values for SCKDF0[0:6] are valid.

Sample values are shown.

SCKDF0[0:6]	Description
000_0000	SCK = MCU_CLK ÷ 2
000_0001	SCK = MCU_CLK ÷ 4
000_0111	SCK = MCU_CLK ÷ 16
100_0000	SCK = MCU_CLK ÷ 8
000_0100	SCK = MCU_CLK ÷ 10
100_1011	SCK = MCU_CLK ÷ 96
111_1110	SCK = MCU_CLK ÷ 504
111_1111	SCK = MCU_CLK ÷ 1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CAPHA0	CKPOL0	LSBF0	DATR02	DATR01	DATR00	CSCKD02	CSCKD01	CSCKD00	SCKDF06	SCKDF05	SCKDF04	SCKDF03	SCKDF02	SCKDF01	SCKDF00

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

QSPI

SCCR1

Serial Channel Control Register 1
Address = $0020_5F14
Reset = $0000
Read/Write

DATR1[0:2]	Delay After Transfer
000	1 SCK cycle delay
001	2 SCK cycles delay
010	4 SCK cycles delay
011	8 SCK cycles delay
100	16 SCK cycles delay
101	32 SCK cycles delay
110	64 SCK cycles delay
111	128 SCK cycles delay

CSCKDF1[0:2]	Assertion to Activation Delay
000	1 SCK cycle delay
001	2 SCK cycles delay
010	4 SCK cycles delay
011	8 SCK cycles delay
100	16 SCK cycles delay
101	32 SCK cycles delay
110	64 SCK cycles delay
111	128 SCK cycles delay

$$\mathbf{SCK} = \frac{\mathbf{MCU\_CLK}}{\mathbf{2\cdot\{3(SCKFD1[6]+1)\cdot(SCKDF1[0:5]+1)\}}}$$

All values for SCKDF1[0:6] are valid.

Sample values are shown.

LSBF1	Description
0	Data transferred MSB first
1	Data transferred LSB first

CKPOL1	Description
0	SCK inactive at logic 1
1	SCK inactive at logic 0

CPHA1	Description
0	Data changes on first SCK transition
1	Data latches on first SCK transition

SCKDF1[0:6]	Description
000_0000	SCK = MCU_CLK ÷ 2
000_0001	SCK = MCU_CLK ÷ 4
000_0111	SCK = MCU_CLK ÷ 16
100_0000	SCK = MCU_CLK ÷ 8
000_0100	SCK = MCU_CLK ÷ 10
100_1011	SCK = MCU_CLK ÷ 96
111_1110	SCK = MCU_CLK ÷ 504
111_1111	SCK = MCU_CLK ÷ 1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CAPHA1	CKPOL1	LSBF1	DATR12	DATR11	DATR10	CSCKD12	CSCKD11	CSCKD10	SCKDF16	SCKDF15	SCKDF14	SCKDF13	SCKDF12	SCKDF11	SCKDF10

Application: ____________________ Date: ____________

____________________ Programmer: ____________

QSPI

SCCR2

Serial Channel Control Register 2
Address = $0020_5F16
Reset = $0000
Read/Write

DATR2[0:2]	Delay After Transfer
000	1 SCK cycle delay
001	2 SCK cycles delay
010	4 SCK cycles delay
011	8 SCK cycles delay
100	16 SCK cycles delay
101	32 SCK cycles delay
110	64 SCK cycles delay
111	128 SCK cycles delay

LSBF2	Description
0	Data transferred MSB first
1	Data transferred LSB first

CKPOL2	Description
0	SCK inactive at logic 1
1	SCK inactive at logic 0

CPHA2	Description
0	Data changes on first SCK transition
1	Data latches on first SCK transition

CSCKDF2[0:2]	Assertion to Activation Delay
000	1 SCK cycle delay
001	2 SCK cycles delay
010	4 SCK cycles delay
011	8 SCK cycles delay
100	16 SCK cycles delay
101	32 SCK cycles delay
110	64 SCK cycles delay
111	128 SCK cycles delay

$$\text{SCK} = \frac{\text{MCU\_CLK}}{2\cdot\{3(\text{SCKFD2}[6]+1)\cdot(\text{SCKDF2}[0:5]+1)\}}$$

All values for SCKDF2[0:6] are valid.

Sample values are shown.

SCKDF2[0:6]	Description
000_0000	SCK = MCU_CLK ÷ 2
000_0001	SCK = MCU_CLK ÷ 4
000_0111	SCK = MCU_CLK ÷ 16
100_0000	SCK = MCU_CLK ÷ 8
000_0100	SCK = MCU_CLK ÷ 10
100_1011	SCK = MCU_CLK ÷ 96
111_1110	SCK = MCU_CLK ÷ 504
111_1111	SCK = MCU_CLK ÷ 1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CAPHA2	CKPOL2	LSBF2	DATR22	DATR21	DATR20	CSCKD22	CSCKD21	CSCKD20	SCKDF26	SCKDF25	SCKDF24	SCKDF23	SCKDF22	SCKDF21	SCKDF20

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

QSPI

SCCR3

Serial Channel Control Register 3
Address = $0020_5F18
Reset = $0000
Read/Write

DATR3[0:2]	Delay After Transfer
000	1 SCK cycle delay
001	2 SCK cycles delay
010	4 SCK cycles delay
011	8 SCK cycles delay
100	16 SCK cycles delay
101	32 SCK cycles delay
110	64 SCK cycles delay
111	128 SCK cycles delay

LSBF3	Description
0	Data transferred MSB first
1	Data transferred LSB first

CKPOL3	Description
0	SCK inactive at logic 1
1	SCK inactive at logic 0

CPHA3	Description
0	Data changes on first SCK transition
1	Data latches on first SCK transition

CSCKDF3[0:2]	Assertion to Activation Delay
000	1 SCK cycle delay
001	2 SCK cycles delay
010	4 SCK cycles delay
011	8 SCK cycles delay
100	16 SCK cycles delay
101	32 SCK cycles delay
110	64 SCK cycles delay
111	128 SCK cycles delay

$$\mathbf{SCK} = \frac{\mathbf{MCU\_CLK}}{\mathbf{2\cdot\{3(SCKFD3[6]+1)\cdot(SCKDF3[0:5]+1)\}}}$$

All values for SCKDF3[0:6] are valid.

Sample values are shown.

SCKDF3[0:6]	Description
000_0000	SCK = MCU_CLK ÷ 2
000_0001	SCK = MCU_CLK ÷ 4
000_0111	SCK = MCU_CLK ÷ 16
100_0000	SCK = MCU_CLK ÷ 8
000_0100	SCK = MCU_CLK ÷ 10
100_1011	SCK = MCU_CLK ÷ 96
111_1110	SCK = MCU_CLK ÷ 504
111_1111	SCK = MCU_CLK ÷ 1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CAPHA3	CKPOL3	LSBF3	DATR32	DATR31	DATR30	CSCKD32	CSCKD31	CSCKD30	SCKDF36	SCKDF35	SCKDF34	SCKDF33	SCKDF32	SCKDF31	SCKDF30

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

QSPI

SCCR4

Serial Channel Control Register 4
Address = $0020_5F1A
Reset = $0000
Read/Write

CSCKDF4[0:2]	Assertion to Activation Delay
000	1 SCK cycle delay
001	2 SCK cycles delay
010	4 SCK cycles delay
011	8 SCK cycles delay
100	16 SCK cycles delay
101	32 SCK cycles delay
110	64 SCK cycles delay
111	128 SCK cycles delay

DATR4[0:2]	Delay After Transfer
000	1 SCK cycle delay
001	2 SCK cycles delay
010	4 SCK cycles delay
011	8 SCK cycles delay
100	16 SCK cycles delay
101	32 SCK cycles delay
110	64 SCK cycles delay
111	128 SCK cycles delay

$$\mathbf{SCK} = \frac{\mathbf{MCU\_CLK}}{\mathbf{2\cdot\{3(SCKFD4[6]+1)\cdot(SCKDF4[0:5]+1)\}}}$$

All values for SCKDF4[0:6] are valid.

Sample values are shown.

LSBF4	Description
0	Data transferred MSB first
1	Data transferred LSB first

SCKDF4[0:6]	Description
000_0000	SCK = MCU_CLK ÷ 2
000_0001	SCK = MCU_CLK ÷ 4
000_0111	SCK = MCU_CLK ÷ 16
100_0000	SCK = MCU_CLK ÷ 8
000_0100	SCK = MCU_CLK ÷ 10
100_1011	SCK = MCU_CLK ÷ 96
111_1110	SCK = MCU_CLK ÷ 504
111_1111	SCK = MCU_CLK ÷ 1

CKPOL4	Description
0	SCK inactive at logic 1
1	SCK inactive at logic 0

CPHA4	Description
0	Data changes on first SCK transition
1	Data latches on first SCK transition

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CAPHA4	CKPOL4	LSBF4	DATR42	DATR41	DATR40	CSCKD42	CSCKD41	CSCKD40	SCKDF46	SCKDF45	SCKDF44	SCKDF43	SCKDF42	SCKDF41	SCKDF40

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

QSPI

Control RAM

Control RAM
Address = $0020_5000 to 507F
Reset = $0000
Read/Write

CONT	Description
0	Deactivate chip select
1	Keep chip select active

PAUSE	Description
0	Not a queue boundary
1	Queue boundary

PCS[0:2]	Delay After Transfer
000	SPIC0 activated
001	SPIC1 activated
010	SPIC2 activated
011	SPIC3 activated
100	SPIC4 activated
101	NOP–No SPIC line activated
110	EOTIE–End-of-transfer interrupt enabled
111	EOQ–End of queue

RE	Description
0	Receive disabled
1	Receive enabled

BYTE	Description
0	16-bit data
1	8-bit data

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	*	BYTE	RE	PAUSE	CONT	PCS2	PCS1	PCS0
0	0	0	0	0	0	0	0	0							
$0				$0											

* = Reserved,

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

QSPI

QPCR

QSPI Port Control Register
Address = $0020_5F00
Reset = $0000
Read/Write

QPCn	Description
0	Pin is GPIO pin
1	Pin is QSPI pin

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	QPC7 (SCK)	QPC6 (MOSI)	QPC5 (MISO)	QPC4 (CS4)	QPC3 (CS3)	QPC2 (CS2)	QPC1 (CS1)	QPC0 (CS0)
0	0	0	0	0	0	0	0								
$0				$0											

QDDR

QSPI Data Direction Register
Address = $0020_5F02
Reset = $0000
Read/Write

QDDn	Description
0	Pin is input
1	Pin is outut

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	QDD7	QDD6	QDD5	QDD4	QDD3	QDD2	QDD1	QDD0
0	0	0	0	0	0	0	0								
$0				$0											

QPDR

QSPI Port Data Register
Address = $0020_5F04
Reset = $00uu
Read/Write

Port Data Bits

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	QPD7	QPD6	QPD5	QPD4	QPD3	QPD2	QPD1	QPD0
0	0	0	0	0	0	0	0								
$0				$0											

* = Reserved,

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

PIT

PITCSR

PIT Control and Status Register
Address = $0020_7000
Reset = $0000
Read/Write

ITIE	Description
0	PIT interrupt disabled
1	PIT interrupt enabled

ITIF	Description
0	PITCNT has not reached zero
1	PITCNT has rolled over

RLD	Description
0	Counter rolls over to $FFFF
1	Counter rolls over to PITMR value

OVW	Description
0	Write to modulus latch does not overwrite PITCNT
1	Write to modulus latch immediately overwrites PITCNT

DBG	Description
0	PIT not affected by Debug mode
1	PIT halted by Debug mode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	*	*	DBG	OVW	ITIE	ITIF	RLD	*
0	0	0	0	0	0	0	0	0	0						0
$0				$0											

PITMR

PIT Modulus Register
Address = $0020_7002
Reset = $FFFF
Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
data	data	data	data	data	data	data	data	data	data	data	data	data	data	data	data

PITCNT

PIT Counter
Address = $0020_7004
Reset = $uuuu
Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
data	data	data	data	data	data	data	data	data	data	data	data	data	data	data	data

* = Reserved,

Application: ______________________________ Date: ______________________

______________________________ Programmer: ______________________

Watchdog Timer

WCR

Watchdog Control Register
Address = $0020_8000
Reset = $0000
Read/Write

WDBG	**Description**
0	Watchdog Timer not affected by Debug mode
1	Watchdog Timer disabled in Debug mode

WDE	**Description**
0	Watchdog Timer is disabled
1	Watchdog Timer is enabled

WDZE	**Description**
0	Watchdog Timer not affected by DOZE mode
1	Watchdog Timer disabled in DOZE mode

Watchdog Time-Out

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WT5	WT4	WT3	WT2	WT1	WT0	*	*	*	*	*	*	*	WDE	WDBG	WDZE
						0	0	0	0	0	0	0			
								$0							

WSR

Watchdog Service Register
Address = $0020_8002
Reset = $0000
Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WS15	WS14	WS13	WS12	WS11	WS10	WS9	WS8	WS7	WS6	WS5	WS4	WS3	WS2	WS1	WS0

* = Reserved,

Application: ______________________________ Date: ____________

______________________________ Programmer: ____________

PWM

TPWCR

Timers and PWM Control Register
Address = $0020_6000
Reset = $0000
Read/Write

TD	**Description**
0	Timers are enabled in DOZE mode
1	Timers are disabled in DOZE mode

TE	**Description**
0	Timers are disabled
1	Timers are enabled

PWE	**Description**
0	PWM counter is disabled
1	PWM counter is enabled

PSPW[0:2]	**Description**
000	PWM prescaler factor = 1
001	PWM prescaler factor = 2
010	PWM prescaler factor = 4
011	PWM prescaler factor = 8
100	PWM prescaler factor = 16
101	PWM prescaler factor = 32
110	PWM prescaler factor = 64
111	PWM prescaler factor = 128

PWD	**Description**
0	PWM counter is enabled in DOZE mode
1	PWM counter is disabled in DOZE mode

TDBG	**Description**
0	Timer stops in Debug mode
1	Timer runs in Debug mode

PST[0:2]	**Description**
000	Timer prescaler factor = 1
001	Timer prescaler factor = 2
010	Timer prescaler factor = 4
011	Timer prescaler factor = 8
100	Timer prescaler factor = 16
101	Timer prescaler factor = 32
110	Timer prescaler factor = 64
111	Timer prescaler factor = 128

PWDBG	**Description**
0	PWM counter stops during Debug mode
1	PWM counter runs during Debug mode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
* 0	* 0	* 0	* 0	PWDBG	TDBG	PWD	PWE	TD	TE	PSPW2	PSPW1	PSPW0	PST2	PST1	PST0
$0															

* = Reserved,

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

PWM

TPWMR

Timers and PWM Mode Register
Address = $0020_6002
Reset = $0000
Read/Write

FO1	**Description**
Writing a 1 to this bit forces the Output Compare 1 function	

IM2[0:1]	**Description**
00	Capture disabled
01	Capture on rising edge only
10	Capture on falling edge only
11	Capture on any edge

FO3	**Description**
(Not pinned out)	

FO4	**Description**
(Not pinned out)	

IM1[0:1]	**Description**
00	Capture disabled
01	Capture on rising edge only
10	Capture on falling edge only
11	Capture on any edge

PWP	**Description**
0	PWM pin active high
1	PWM pin active low

OM1[0:1]	**Description**
00	Timer disconnected from pin
01	Toggle output pin
10	Clear output pin
11	Set output pin

PWC	**Description**
0	PWM disconnected from PWM pin
1	PWM connected to PWM pin

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	PWC	PWP	FO4	FO3	FO1	IM21	IM20	IM11	IM10	*	*	*	*	OM11	OM10
0										0	0	0	0		

* = Reserved,

Application: ______________________________ Date: ____________

______________________________ Programmer: ____________

PWM

TPWSR

Timers and PWM Status Register
Address = $0020_6004
Reset = $0000
Read/Write

IF2	**Description**
0	Timer 2 Input Capture has not occurred
1	Timer 2 Input Capture has occurred

PWF	**Description**
0	PWM compare has not occurred
1	PWM compare has occurred

TOV	**Description**
0	TCNT overflow has not occurred
1	TCNT overflow has occurred

PWO	**Description**
0	PWCNT rollover has not occurred
1	PWCNT rollover has occurred

IF1	**Description**
0	Timer 1 Input Capture has not occurred
1	Timer 1 Input Capture has occurred

OF4	**Description**
0	Timer 4 Output Compare has not occurred
1	Timer 4 Output Compare has occurred

OF3	**Description**
0	Timer 3 Output Compare has not occurred
1	Timer 3 Output Compare has occurred

OF1	**Description**
0	Timer 1 Output Compare has not occurred
1	Timer 1 Output Compare has occurred

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	PWO	TOV	PWF	IF2	IF1	OF4	OF3	OF1
0	0	0	0	0	0	0	0								
$0				$0											

* = Reserved,

Application: ______________________ Date: ______________

______________________ Programmer: ______________

PWM

TPWIR

Timers and PWM Interrupt Register
Address = $0020_6006
Reset = $0000
Read/Write

IF1IE	Description
0	Interrupt disabled
1	Timer 1 Input Capture interrupt enabled

OF4IE	Description
0	Interrupt disabled
1	Timer 4 Output Compare interrupt enabled

IF2IE	Description
0	Interrupt disabled
1	Timer 2 Input Capture interrupt enabled

OF3IE	Description
0	Interrupt disabled
1	Timer 3 Output Compare interrupt enabled

PWFIE	Description
0	Interrupt disabled
1	PWM Output Compare interrupt enabled

OF1IE	Description
0	Interrupt disabled
1	Timer 1 Output Compare interrupt enabled

TOVIE	Description
0	Interrupt disabled
1	TCNT overflow interrupt enabled

PWOIE	Description
0	Interrupt disabled
1	PWCNT rollover interrupt enabled

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	PWOIE	TOVIE	PWFIE	IF2IE	IF1IE	OF4IE	OF3IE	OF1IE
0	0	0	0	0	0	0	0								
$0				$0											

∗ = Reserved,

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

PWM

TOCR1

Timer 1 Output Compare Register
Address = $0020_6008
Reset = $0000
Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
data	data	data	data	data	data	data	data	data	data	data	data	data	data	data	data

TOCR3

Timer 3 Output Compare Register
Address = $0020_600A
Reset = $0000
Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
data	data	data	data	data	data	data	data	data	data	data	data	data	data	data	data

TOCR4

Timer 4 Output Compare Register
Address = $0020_600C
Reset = $0000
Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
data	data	data	data	data	data	data	data	data	data	data	data	data	data	data	data

TICR1

Timer 1 Input Capture Register
Address = $0020_600E
Reset = $0000
Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
data	data	data	data	data	data	data	data	data	data	data	data	data	data	data	data

TICR2

Timer 2 Input Capture Register
Address = $0020_6010
Reset = $0000
Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
data	data	data	data	data	data	data	data	data	data	data	data	data	data	data	data

Application: ______________________________

Date: ____________________

Programmer: ____________________

PWM

PWOR

PWM Output Compare Register
Address = $0020_6012
Reset = $0000
Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
data	data	data	data	data	data	data	data	data	data	data	data	data	data	data	data

TCNT

Timer Count Register
Address = $0020_6014
Reset = $0000
Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
data	data	data	data	data	data	data	data	data	data	data	data	data	data	data	data

PWMR

PWM Modulus Register
Address = $0020_6016
Reset = $0000
Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
data	data	data	data	data	data	data	data	data	data	data	data	data	data	data	data

PWCNT

PWM Count Register
Address = $0020_6018
Reset = $0000
Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
data	data	data	data	data	data	data	data	data	data	data	data	data	data	data	data

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

Protocol Timer

PTCR

PT Control Register
Address = $0020_3800
Reset = $0000
Read/Write

TDZD	Description
0	Protocol Timer ignores DOZE mode
1	Protocol Timer stops during DOZE mode

MTER	Description
0	Active macro execution continues to end of macro
1	Active macro execution halts immediately when HLTR bit is set or ‘End_of_frame_halt’ received

SPBP	Description
0	Reference Slot Prescaler Counter (RSPC) drives RSC
1	RSPC bypassed, TICK drives RSC

TIME	Description
0	Protocol Timer event disabled until CFE occurs
1	Protocol Timer event executes immediately after TE assertion or HALT state is exited

HLTR	Description
0	Timer HALT not requested
1	Timer HALT requested

CFCE	Description
0	Channel Frame Counter disabled
1	Channel Frame Counter enabled

TE	Description
0	Protocol Timer disabled
1	Protocol Timer enabled

RSCE	Description
0	Reference Slot Counter disabled
1	Reference Slot Counter enabled

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	RSCE	CFCE	*	*	HLTR	SPBP	TDZD	MTER	TIME	TE
0	0	0	0	0	0			0	0						
$0															

* = Reserved,

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

Protocol Timer

PTIER

PT Interrupt Enable Register
Address = $0020_3802
Reset = $0000
Read/Write

DSIE	Description
0	Interrupt disabled
1	DSP Interrupt enabled

DVIE	Description
0	Interrupt disabled
1	DSP Vector Interrupt enabled

THIE	Description
0	Interrupt disabled
1	Timer HALT Interrupt enabled

TERIE	Description
0	Interrupt disabled
1	Timer Error Interrupt enabled

MCIE2	Description
0	Interrupt disabled
1	MCU Interrupt 2 enabled

MCIE1	Description
0	Interrupt disabled
1	MCU Interrupt 1 enabled

MCIE0	Description
0	Interrupt disabled
1	MCU Interrupt 0 enabled

RSNIE	Description
0	Interrupt disabled
1	Reference Slot Interrupt enabled

CFNIE	Description
0	Interrupt disabled
1	Channel Frame Number Interrupt enabled

CFIE	Description
0	Interrupt disabled
1	Channel Frame Interrupt enabled

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	TERIE	THIE	DVIE	DSIE	*	*	MCIE2	MCIE1	MCIE0	*	RSNIE	CFNIE	CFIE
0	0	0					0	0				0			

* = Reserved,

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

Protocol Timer

PTSR

PT Status Register
Address = $0020_3804
Reset = $0000
Read/Write

DVI	Description
0	Interrupt has not occurred
1	DSP Vector Interrupt event has occurred

THS	Description
0	Timer is not in HALT state
1	Timer is in HALT state

EOFE	Description
0	No error
1	End of Frame Error has occurred

MBUE	Description
0	No error
1	Macro Being Used Error has occurred

PCE	Description
0	No error
1	Pin Contention Error has occurred

DSPI	Description
0	Interrupt has not occurred
1	DSP Interrupt event has occurred

MCUI2	Description
0	Interrupt has not occurred
1	MCU Interrupt 2 event has occurred

MCUI1	Description
0	Interrupt has not occurred
1	MCU Interrupt 1 event has occurred

MCUI0	Description
0	Interrupt has not occurred
1	MCU Interrupt 0 event has occurred

RSNI	Description
0	Interrupt has not occurred
1	Reference Slot Number Interrupt has occurred

CFNI	Description
0	Interrupt has not occurred
1	Channel Frame Number Interrupt event has occurred

CFI	Description
0	Interrupt has not occurred
1	Channel Frame Interrupt event has occurred

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
* 0	PCE	MBUE	EOFE	THS	DVI	DSPI	* 0	* 0	MCUI2	MCUI1	MCUI0	* 0	RSNI	CFNI	CFI

* = Reserved,

Application: ______________________________ Date: ____________________

______________________________ Programmer: ____________________

Protocol Timer

PTEVR

PT Event Register
Address = $0020_3806
Reset = $0000
Read/Write

RXMA	**Description**
0	Macro not active
1	Receive macro is active

ACT	**Description**
0	Frame Table 0 active
1	Frame Table 1 active

TXMA	**Description**
0	Macro not active
1	Transmit macro is active

THIP	**Description**
0	Timer not halted
1	Timer HALT in progress

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
* 0	* 0	* 0	* 0	* 0	* 0	* 0	* 0	* 0	* 0	* 0	* 0	THIP	TXMA	RXMA	ACT
$0				$0				$0							

TIMR

Time Interval Modulus Register
Address = $0020_3808
Reset = $0000
Read/Write

Timer Interval Modulus Value

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
* 0	* 0	* 0	* 0	* 0	* 0	* 0	TIMV8	TIMV7	TIMV6	TIMV5	TIMV4	TIMV3	TIMV2	TIMV1	TIMV0
$0															

CTIC

Channel Time Interval Counter
Address = $0020_380A
Reset = $0000
Read/Write

Channel Time Interval Value

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
* 0	* 0	CTIV13	CTIV12	CTIV11	CTIV10	CTIV9	CTIV8	CTIV7	CTIV6	CTIV5	CTIV4	CTIV3	CTIV2	CTIV1	CTIV0

* = Reserved,

Application: ______________________ Date: ______________

______________________ Programmer: ______________

Protocol Timer

CTIMR

Channel Time Interval Modulus Register
Address = $0020_380C
Reset = $0000
Read/Write

Channel Time Interval Modulus Value

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
* 0	* 0	CTIMV13	CTIMV12	CTIMV11	CTIMV10	CTIMV9	CTIMV8	CTIMV7	CTIMV6	CTIMV5	CTIMV4	CTIMV3	CTIMV2	CTIMV1	CTIMV0

CFC

Channel Frame Counter
Address = $0020_380E
Reset = $0000
Read/Write

Channel Frame Count Value

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
* 0	* 0	* 0	* 0	* 0	* 0	* 0	CFCV8	CFCV7	CFCV6	CFCV5	CFCV4	CFCV3	CFCV2	CFCV1	CFCV0
$0															

CFMR

Channel Frame Modulus Register
Address = $0020_3810
Reset = $0000
Read/Write

Channel Frame Modulus Value

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
* 0	* 0	* 0	* 0	* 0	* 0	* 0	CFMV8	CFMV7	CFMV6	CFMV5	CFMV4	CFMV3	CFMV2	CFMV1	CFMV0
$0															

* = Reserved,

Application: ______________________ Date: ______________

______________________ Programmer: ______________

Protocol Timer

RSC

Reference Slot Counter
Address = $0020_3812
Reset = $0000
Read/Write

Reference Slot Count Value

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	RSCV7	RSCV6	RSCV5	RSCV4	RSCV3	RSCV2	RSCV1	RSCV0
0	0	0	0	0	0	0	0								
$0				$0											

RSMR

Reference Slot Modulus Register
Address = $0020_3814
Reset = $0000
Read/Write

Reference Slot Modulus Value

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	RSMV7	RSMV6	RSMV5	RSMV4	RSMV3	RSMV2	RSMV1	RSMV0
0	0	0	0	0	0	0	0								
$0				$0											

* = Reserved,

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

Protocol Timer

FTPTR

Frame Table Pointer
Address = $0020_381C
Reset = $00uu
Read/Write

Frame Table Pointer

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	*	FTPTR6	FTPTR5	FTPTR4	FTPTR3	FTPTR2	FTPTR1	FTPTR0
0	0	0	0	0	0	0	0	0							
$0				$0											

MTPTR

Macro Table Pointer
Address = $0020_381E
Reset = $uuuu
Read/Write

Transmit Macro Table Pointer

Receive Macro Table Pointer

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	TxPTR6	TxPTR5	TxPTR4	TxPTR3	TxPTR2	TxPTR1	TxPTR0	*	RxPTR6	RxPTR5	RxPTR4	RxPTR3	RxPTR2	RxPTR1	RxPTR0
0								0							

$*$ = Reserved,

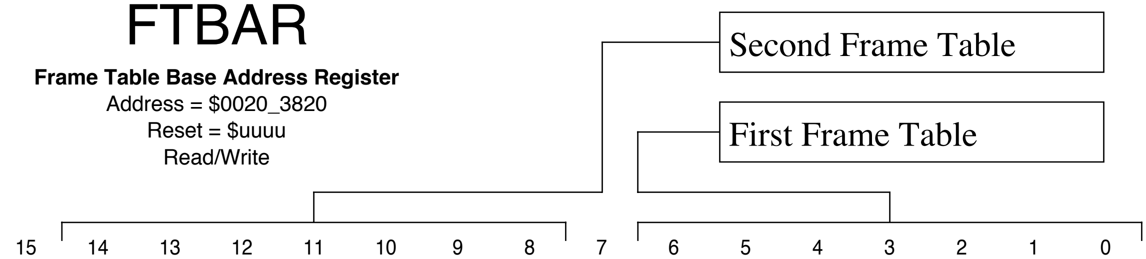
Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

Protocol Timer

FTBAR

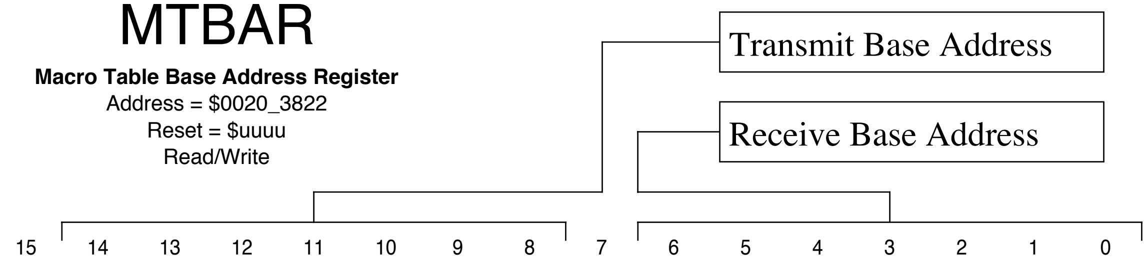
Frame Table Base Address Register
Address = $0020_3820
Reset = $uuuu
Read/Write



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	FTBA16	FTBA15	FTBA14	FTBA13	FTBA12	FTBA11	FTBA10	*	FTBA6	FTBA5	FTBA4	FTBA3	FTBA2	FTBA1	FTBA0
0								0							

MTBAR

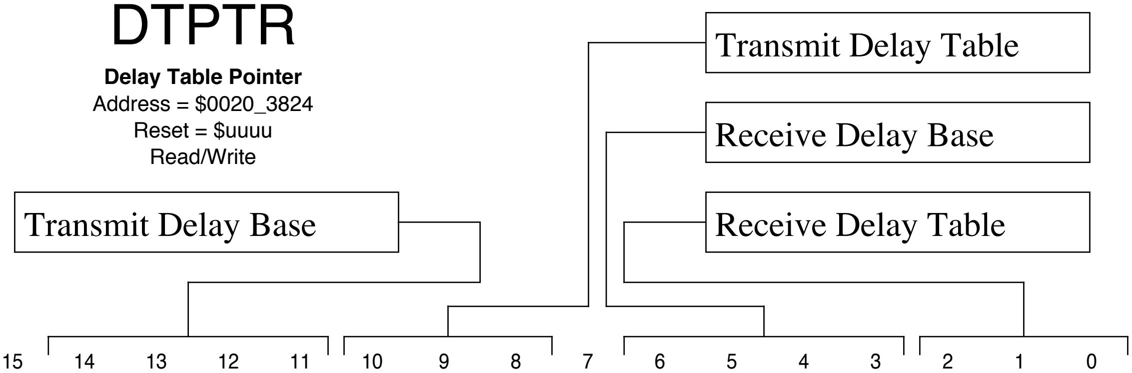
Macro Table Base Address Register
Address = $0020_3822
Reset = $uuuu
Read/Write



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	TxBA6	TxBA5	TxBA4	TxBA3	TxBA2	TxBA1	TxBA0	*	RxBA6	RxBA5	RxBA4	RxBA3	RxBA2	RxBA1	RxBA0
0								0							

DTPTR

Delay Table Pointer
Address = $0020_3824
Reset = $uuuu
Read/Write



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	TDBA3	TDBA2	TDBA1	TDBA0	TDPTR2	TDPTR1	TDPTR0	*	RDBA3	RDBA2	RDBA1	RDBA0	RDPTR2	RDPTR1	RDPTR0
0								0							

* = Reserved,

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

Protocol Timer

PTPCR

PT Port Control Register
Address = $0020_3816
Reset = $0000
Read/Write

PTPCn	Description
0	Pin is GPIO output
1	Pin is Protocol Timer output

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	PTPC7	PTPC6	PTPC5	PTPC4	PTPC3	PTPC2	PTPC1	PTPC0
0	0	0	0	0	0	0	0								
$0				$0											

PTDDR

PT Data Direction Register
Address = $0020_3818
Reset = $0000
Read/Write

PTDDn	Description
0	Pin is input (when GPIO)
1	Pin is output (when GPIO)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	PTDD7	PTDD6	PTDD5	PTDD4	PTDD3	PTDD2	PTDD1	PTDD0
0	0	0	0	0	0	0	0								
$0				$0											

PTPDR

PT Port Data Register
Address = $0020_381A
Reset = $00uu
Read/Write

Port Data Bits

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	PTDAT7	PTDAT6	PTDAT5	PTDAT4	PTDAT3	PTDAT2	PTDAT1	PTDAT0
0	0	0	0	0	0	0	0								
$0				$0											

* = Reserved,

Application: ____________________ Date: ____________

____________________ Programmer: ____________

UART

URX

UART Receive Register
Address = $0020_4000 to 403C
Reset = $00uu
Read/Write

OVRRUN	Description
0	No FIFO overrun
1	URX FIFO overrun detected

FRMERR	Description
0	No framing error detected
1	Character has a framing error

BRK	Description
0	Character is not a BREAK
1	Character is a BREAK

PRERR	Description
0	No parity error detected
1	Parity error detected

ERR	Description
0	No error detected in bits 13–10
1	Error detected

CHARRDY	Description
0	Character not ready
1	Character ready

Rx Data

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CHARRDY	ERR	OVRRUN	FRMERR	BRK	PRERR	*	*	data	data	data	data	data	data	data	data
						0	0								

UTX

UART Transmit Register
Address = $0020_4040 to 407C
Reset = $uuuu
Read/Write

Tx Data

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,

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

UART
UCR1

UART Control Register 1
Address = $0020_4080
Reset = $0000
Read/Write

RXFL[0:1]	Description
00	Interrupt if RX FIFO contains 1 or more characters
01	Interrupt if RX FIFO contains 4 or more characters
10	Interrupt if RX FIFO contains 8 or more characters
11	Interrupt if RX FIFO contains 14 or more characters

TXEN	Description
0	Transmitter disabled
1	Transmitter enabled

TRDYIE	Description
0	Interrupt disabled
1	Transmitter Ready Interrupt enabled

TXFL[0:1]	Description
00	Interrupt if TX FIFO has slot for 1 or more characters
01	Interrupt if TX FIFO has slot for 4 or more characters
10	Interrupt if TX FIFO has slot for 8 or more characters
11	Interrupt if TX FIFO has slot for 14 or more characters

RRDYIE	Description
0	Interrupt disabled
1	Receiver Ready Interrupt enabled

RXEN	Description
0	Receiver disabled
1	Receiver enabled

IREN	Description
0	Infrared interface (IrDA) disabled
1	Infrared interface (IrDA) enabled

TXEIE	Description
0	Interrupt disabled
1	Transmitter Empty Interrupt enabled

RTSDIE	Description
0	$\overline{\text{RTS}}$ interrupt disabled
1	$\overline{\text{RTS}}$ interrupt enabled

SNDBRK	Description
0	(Bit is cleared)
1	Send continuous BREAK

DOZE	Description
0	UART enabled during DOZE mode
1	UART disabled during DOZE mode

UEN	Description
0	UART disabled
1	UART enabled

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TXFL1	TXFL0	TRDYIE	TXEN	RXFL1	RXFL0	RRDYIE	RXEN	IREN	TXEIE	RTSDIE	SNDBRK	* 0	* 0	DOZE	UEN

* = Reserved,

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

UART

UCR2

UART Control Register 2
Address = $0020_4082
Reset = $0000
Read/Write

CTSD	Description
0	$\overline{\text{CTS}}$ pin is inactive (high)
1	$\overline{\text{CTS}}$ pin is active (low)

CTSC	Description
0	$\overline{\text{CTS}}$ pin controlled by CTSD
1	$\overline{\text{CTS}}$ pin controlled by receiver

IRTS	Description
0	Transmit only when $\overline{\text{RTS}}$ pin is asserted
1	$\overline{\text{RTS}}$ pin is ignored

PREN	Description
0	Parity disabled
1	Parity enabled

PROE	Description
0	Even parity
1	Odd parity

STPB	Description
0	1 Stop bit
1	2 Stop bits

CHSZ	Description
0	7-bit characters
1	8-bit characters

CLKSRC	Description
0	Bit clock generated from 16x bit clock generator
1	Bit clock derived from IRQ7/$\overline{\text{DTR}}$ pin (input)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	IRTS	CTSC	CTSD	*	*	*	PREN	PROE	STPB	CHSZ	CLKSRC	*	*	*	*
0				0	0	0						0	0	0	0
												$0			

UBRGR

UART Bit Rate Generator Register
Address = $0020_4084
Reset = $0000
Read/Write

Clock Divider Bits

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	CD11	CD10	CD9	CD8	CD7	CD6	CD5	CD4	CD3	CD2	CD1	CD0
0	0	0	0												
$0															

* = Reserved,

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

UART

USR

UART Status Register
Address = $0020_4086
Reset = $A000
Read Only

TXE	**Description**
0	Unsent transmit data
1	All transmit data has been sent

RTSS	**Description**
0	$\overline{\text{RTS}}$ pin is high (inactive)
1	$\overline{\text{RTS}}$ pin is low (active)

TRDY	**Description**
0	Unsent characters above TXFL[0:1]
1	Unsent characters below TXFL[0:1]

RRDY	**Description**
0	Unread characters below RXFL[0:1]
1	Unread characters above RXFL[0:1]

RTSD	**Description**
0	$\overline{\text{RTS}}$ pin has not changed state
1	$\overline{\text{RTS}}$ pin has changed state

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TXE	RTSS	TRDY	*	*	*	RRDY	*	*	*	RTSD	*	*	*	*	*
			0	0	0		0	0	0		0	0	0	0	0
												$0			

UTS

UART Test Register
Address = $0020_4088
Reset = $0000
Read/Write

NOTE: This register is included for test

FRCPERR	**Description**
0	No intentional parity errors generated
1	Intentional parity error generated

LOOP	**Description**
0	Normal operation
1	Receiver connected to transmitter

LOOPIR	**Description**
0	Normal IR operation
1	IR Receiver connected to IR transmitter

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	FRCPERR	LOOP	*	LOOPIR	*	*	*	*	*	*	*	*	*	*
0	0			0		0	0	0	0	0	0	0	0	0	0
								$0				$0			

* = Reserved,

Application: ______________________________ Date: ______________

Programmer: ______________

UART

UPCR

UART Port Control Register
Address = $0020_408A
Reset = $0000
Read/Write

UPCn	Description
0	Pin is GPIO pin
1	Pin is UART pin

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	*	*	*	*	UPC3	UPC2	UPC1	UPC0
0	0	0	0	0	0	0	0	0	0	0	0				
$0				$0				$0							

UDDR

UART Data Direction Register
Address = $0020_408C
Reset = $0000
Read/Write

UDDn	Description
0	Pin is input (when GPIO)
1	Pin is output (when GPIO)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	*	*	*	*	UDD3	UDD2	UDD1	UDD0
0	0	0	0	0	0	0	0	0	0	0	0				
$0				$0				$0							

UPDR

UART Port Data Register
Address = $0020_408E
Reset = $000u
Read/Write

Port Data Bits

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	*	*	*	*	UPD3	UPD2	UPD1	UPD0
0	0	0	0	0	0	0	0	0	0	0	0				
$0				$0				$0							

* = Reserved,

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

SCP

SCPCR

SCP Control Register
Address = $0020_B000
Reset = $0000
Read/Write

SCSR	Description
0	(bit is cleared)
1	Reset SIM

SCPT	Description
0	Even parity
1	Odd parity

SCIC	Description
0	Parity determined by SIPT bit
1	Parity determined by smart card

NKPE	Description
0	No NACK on parity error
1	NACK generated on parity error

SCTE	Description
0	Transmitter disabled
1	Transmitter enabled

SCRE	Description
0	Receiver disabled
1	Receiver enabled

SIBR	Description
0	Baud rate = SIM_CLK ÷ 372
1	Baud rate = SIM_CLK ÷ 64

DOZE	Description
0	SCP ignores DOZE mode
1	SCP halts in DOZE mode

NKOVR	Description
0	No NACK on overrun error
1	NACK generated on overrun error

CLKSEL	Description
0	SIM_CLK = CKIH ÷ 5
1	SIM_CLK = CKIH ÷ 4

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	CLKSEL	NKOVR	DOZE	SIBR	SCSR	SCPT	SCIC	NKPE	SCTE	SCRE
0	0	0	0	0	0										
$0															

* = Reserved,

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

SCP

SCACR

Smart Card Activation Control Register
Address = $0020_B002
Reset = $0000
Read/Write

SCRS	Description
0	$\overline{\text{SIMRESET}}$ pin asserted
1	$\overline{\text{SIMRESET}}$ pin deasserted

SCCLK	Description
0	SIMCLK pin disabled
1	SIMCLK pin enabled

SCDPE	Description
0	SIMDATA pin disabled
1	SIMDATA pin enabled

SCPE	Description
0	PWR_EN pin disabled
1	PWR_EN pin enabled

APDE	Description
0	Auto power-down disabled
1	Auto power-down enabled

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	*	*	*	SCCLK	SCRS	SCDPE	SCPE	APDE
0	0	0	0	0	0	0	0	0	0	0					
$0				$0											

SCPIER

SCP Interrupt Enable Register
Address = $0020_B004
Reset = $0000
Read/Write

SCFNIE	Description
0	Interrupt disabled
1	Enable interrupt for data reception

SCTCIE	Description
0	Interrupt disabled
1	Enable interrupt for transmission complete

SCFFIE	Description
0	Interrupt disabled
1	Enable interrupt for receive FIFO full

SCRRIE	Description
0	Interrupt disabled
1	Enable interrupt for receive error

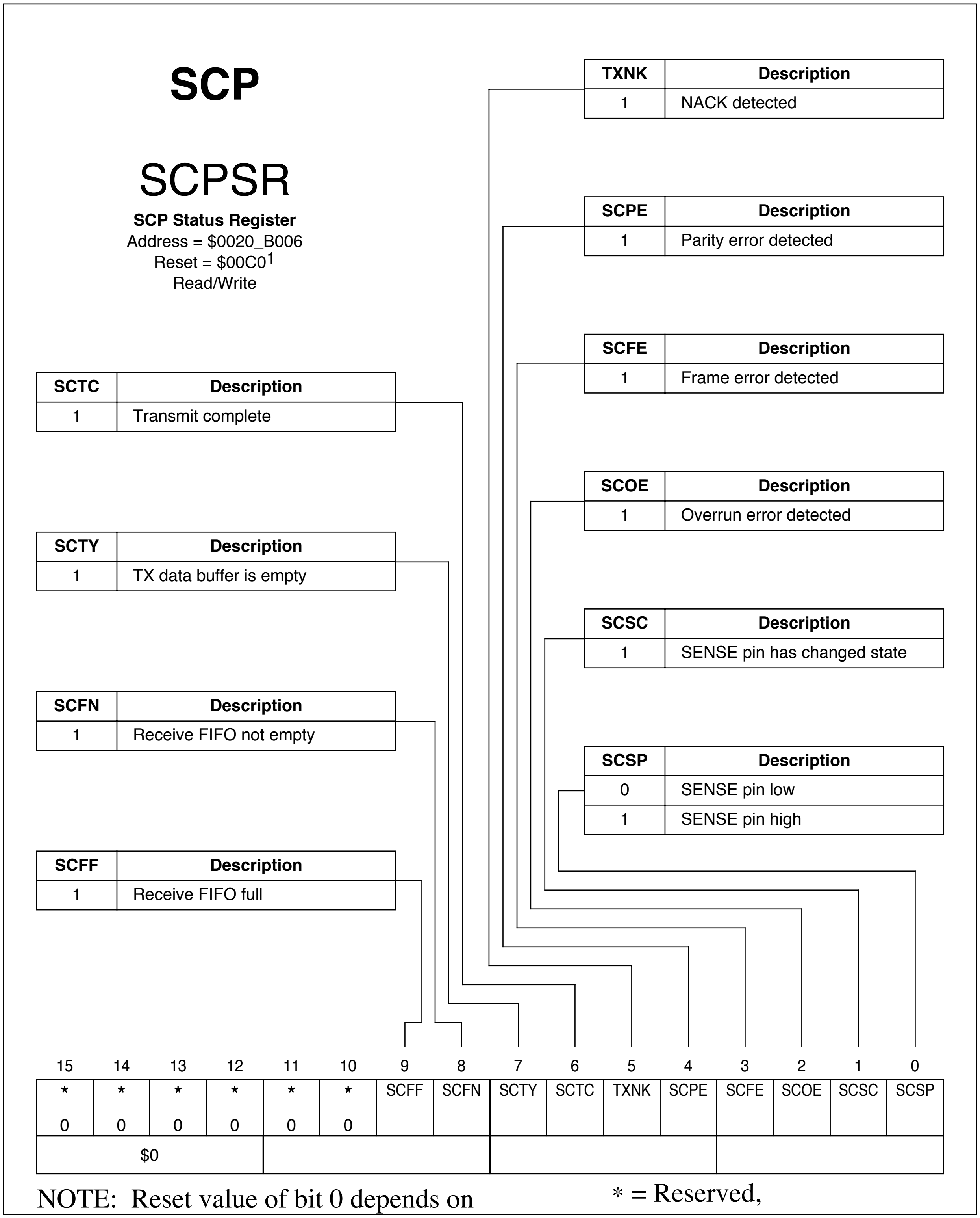
SCSCIE	Description
0	Interrupt disabled
1	Enable interrupt for card sense change

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	*	*	*	SCTCIE	SCFNIE	SCFFIE	SCRRIE	SCSCIE
0	0	0	0	0	0	0	0	0	0	0					
$0				$0											

* = Reserved,

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________



SCP

SCPSR

SCP Status Register
Address = $0020_B006
Reset = $00C0[1]
Read/Write

TXNK	**Description**
1	NACK detected

SCPE	**Description**
1	Parity error detected

SCFE	**Description**
1	Frame error detected

SCOE	**Description**
1	Overrun error detected

SCSC	**Description**
1	SENSE pin has changed state

SCSP	**Description**
0	SENSE pin low
1	SENSE pin high

SCTC	**Description**
1	Transmit complete

SCTY	**Description**
1	TX data buffer is empty

SCFN	**Description**
1	Receive FIFO not empty

SCFF	**Description**
1	Receive FIFO full

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
* 0	* 0	* 0	* 0	* 0	* 0	SCFF	SCFN	SCTY	SCTC	TXNK	SCPE	SCFE	SCOE	SCSC	SCSP
$0															

NOTE: Reset value of bit 0 depends on

* = Reserved,

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

SCP

SCPDR

SCP Data Register
Address = $0020_B008
Reset = $00uu
Read/Write

SCP Data Bits

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	SCPD7	SCPD6	SCPD5	SCPD4	SCPD3	SCPD2	SCPD1	SCPD0
0	0	0	0	0	0	0	0								
$0				$0											

SCPPCR

SCP Port Control Register
Address = $0020_B00A
Reset = $00uu
Read/Write

SMEN	Description
0	Pins function as GPIO
1	Pins function as SCP

SCPDDn	Description
0	Pin is an input when configured as GPIO
1	Pin is an output when configured as GPIO

Port Data Bits

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMEN	*	*	*	*	*	SCPDD4	SCPDD3	SCPDD2	SCPDD1	SCPDD0	SCPPD4	SCPPD3	SCPPD2	SCPPD1	SCPPD0
	0	0	0	0	0										

* = Reserved,

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

KP

KPCR

Keypad Port Control Register
Address = $0020_A000
Reset = $0000
Read/Write

KCOn	Description
0	Column strobe output n is totem-pole drive
1	Column strobe output n is open-drain

KREn	Description
0	Row n is not included in key press detect
1	Row n is included in key press detect

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
KCO7	KCO6	KCO5	KCO4	KCO3	KCO2	KCO1	KCO0	KRE7	KRE6	KRE5	KRE4	KRE3	KRE2	KRE1	KRE0

KPSR

Keypad Status Register
Address = $0020_A002
Reset = $0000
Read/Write

KPKD	Description
0	No keypad press detected
1	Keypad press detected

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	KPKD
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
$0				$0				$0							

* = Reserved,

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

KP

KDDR

Keypad Data Direction Register
Address = $0020_A004
Reset = $0000
Read/Write

KCDDn	Description
0	Column strobe n pin is an input
1	Column strobe n pin is an output

KRDDn	Description
0	Row n pin is an input
1	Row n pin is an output

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
KCDD7	KCDD6	KCDD5	KCDD4	KCDD3	KCDD2	KCDD1	KCDD0	KRDD7	KRDD6	KRDD5	KRDD4	KRDD3	KRDD2	KRDD1	KRDD0

KPDR

Keypad Data Register
Address = $0020_A006
Reset = $0000
Read/Write

Column Data Bits

Row Data Bits

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
KCD7	KCD6	KCD5	KCD4	KCD3	KCD2	KCD1	KCD0	KRD7	KRD6	KRD5	KRD4	KRD3	KRD2	KRD1	KRD0

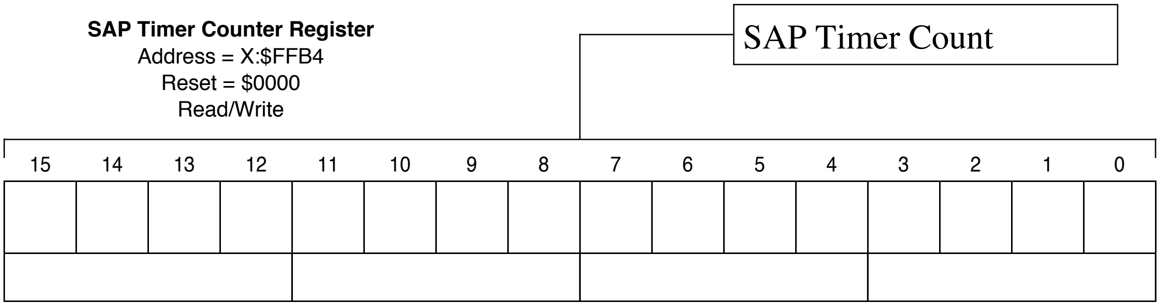
Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

SAP

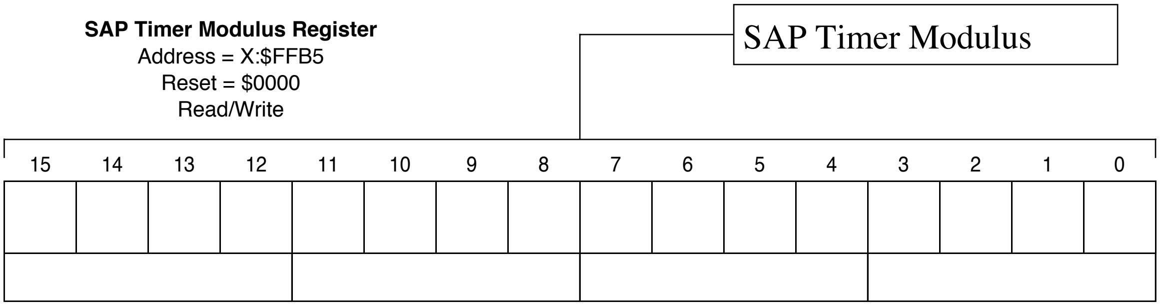
SAPCNT

SAP Timer Counter Register
Address = X:$FFB4
Reset = $0000
Read/Write



SAPMR

SAP Timer Modulus Register
Address = X:$FFB5
Reset = $0000
Read/Write

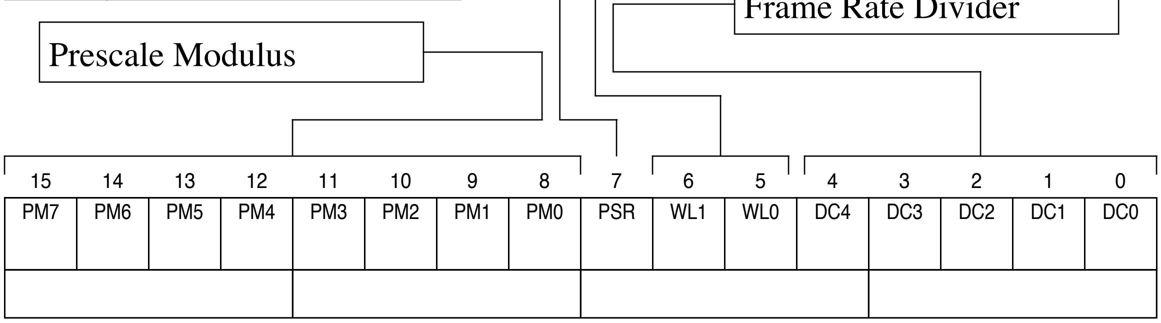


SAPCRA

SAP Control Register A
Address = X:$FFB6
Reset = $0000
Read/Write

WL1	WL0	Description
0	0	8 bits per word
0	1	12 bits per word
1	0	16 bits per word
1	1	(Reserved)

PSR	Description
0	No prescale
1	Prescale applied



Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

SAP

SAPCRB

SAP Control Register B
Address = X:$FFB7
Reset = $0000
Read/Write

RLIE	Description
0	Disable interrupt
1	Enable interrupt for last receive time slot

TEIE	Description
0	Disable interrupt
1	Enable interrupt for transmit error

REIE	Description
0	Disable interrupt
1	Enable interrupt for receive error

TLIE	Description
0	Disable interrupt
1	Enable interrupt for last transmit time slot

RIE	Description
0	Disable interrupt
1	Enable interrupt when a word is received

TIE	Description
0	Transmit Interrupt disabled
1	Transmit Interrupt enabled

RE	Description
0	Receiver disabled
1	Receiver enabled

TE	Description
0	Transmitter disabled
1	Transmitter enabled

TCE	Description
0	Timer disabled
1	Enable SAP timer

Serial Output Flags

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
REIE	TEIE	RLIE	TLIE	RIE	TIE	RE	TE	*	*	*	*	*	TCE	OF1	OF0
								0	0	0	0	0			
								$0							

* = Reserved,

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

SAP

SAPCRC

SAP Control Register C
Address = X:$FFB8
Reset = $0000
Read/Write

SHFD	Description
0	Data shifted out MSB first
1	Data shifted out LSB first

BRM	Description
0	SAP clock source is DSP_CLK
1	SAP clock source is BRM_CLK

FSL1	FSL0	Description
0	0	WL bit clock for both TX and RX
0	1	1-bit clock for TX WL bit clock for RX
1	0	1-bit clock for both TX and RX
1	1	WL bit clock for TX 1-bit clock for RX

FSR	Description
0	Frame sync occurs with first bit of current frame
1	Frame sync occurs with last bit of previous frame

FSP	Description
0	Positive frame sync
1	Negative frame sync

CKP	Description
0	Transmit – bit clock rising edge Receive – bit clock falling edge (default)
1	Transmit – bit clock falling edge Receive – bit clock rising edge

SCKD	Description
0	External clock source
1	Internal clock source

SCD2	Description
0	SCD2 pin is input
1	SCD2 pin is output

SCD1	Description
0	SCD1 pin is input
1	SCD1 pin is output

SCD0	Description
0	SCD0 pin is input
1	SCD0 pin is output

MOD	Description
0	Normal mode
1	Network mode

SYN	Description
0	Asynchronous mode
1	Synchronous mode

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FSP	FSR	FSL1	FSL0	*	*	*	BRM	SHFD	CKP	SCKD	SCD2	SCD1	SCD0	MOD	SYN
				0	0	0									

* = Reserved,

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

SAP

SAPSR

SAP Status Register
Address = X:$FFB9
Reset = $0000
Read/Write

TUE	**Description**
1	TX underrun occurred

TFS	**Description**
1	TX frame sync occurred

ROE	**Description**
1	RX overrun occurred

RFS	**Description**
1	RX frame sync occurred

TDE	**Description**
1	TX data register empty

Serial Input Flags

RDF	**Description**
1	RX data register has data

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	RDF	TDE	ROE	TUE	RFS	TFS	IF1	IF0
0	0	0	0	0	0	0	0								
$0				$0											

* = Reserved,

Application: ______________________________ Date: ____________________

______________________________ Programmer: ____________________

SAP

SAPRX

SAP Receive Data Register
Address = X:$FFBA
Reset = $0000
Read/Write

<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="8">High Byte</td><td colspan="8">Low Byte</td></tr>
<tr><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td></tr>
</table>

SAPTSR

SAP Time Slot Register
Address = X:$FFBB
Reset = $0000
Read/Write

<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="16">Dummy Register, Written During Inactive Time Slots</td></tr>
<tr><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td></tr>
</table>

SAPTX

SAP Transmit Data Register
Address = X:$FFBC
Reset = $0000
Read/Write

<table>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="8">High Byte</td><td colspan="8">Low Byte</td></tr>
<tr><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td><td colspan="4"></td></tr>
</table>

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

SAP

SAPPDR

SAP Port Data Register
Address = X:$FFBD
Reset = $0000
Read/Write

Port Data Bits

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	*	*	SAPPD5	SAPPD4	SAPPD3	SAPPD2	SAPPD1	SAPPD0
0	0	0	0	0	0	0	0	0	0						
$0				$0											

PEN	Description
0	Port pins are tri-stated
1	Port pins enabled

SAPPCR

SAP Port Control Register
Address = X:$FFBF
Reset = $0000
Read/Write

SAPPCn	Description
0	Pin configured as GPIO
1	Pin configured as SAP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	PEN	*	SAPPC5 (STDA)	SAPPC4 (SRDA)	SAPPC3 (SCKA)	SAPPC2 (SC2A)	SAPPC1 (SC1A)	SAPPC0 (SC0A)
0	0	0	0	0	0	0	0		0						
$0				$0											

SAPDDR

SAP Data Direction Register
Address = X:$FFBE
Reset = $0000
Read/Write

SAPDDn	Description
0	Pin is input
1	Pin is output

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	*	*	SAPDD5	SAPDD4	SAPDD3	SAPDD2	SAPDD1	SAPDD0
0	0	0	0	0	0	0	0	0	0						
$0				$0											

* = Reserved,

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

BBP

BBPRMR

BBP Receive Counter Modulus Register
Address = X:$FFA4
Reset = $0000
Read/Write

Receive Counter Modulus

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

BBPTMR

BBP Transmit Counter Modulus Register
Address = X:$FFA5
Reset = $0000
Read/Write

Transmit Counter

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

BBPCRA

BBP Control Register A
Address = X:$FFA6
Reset = $0000
Read/Write

WL1	WL0	Description
0	0	8 bits per word
0	1	12 bits per word
1	0	16 bits per word
1	1	(Reserved)

PSR	Description
0	No prescale
1	Prescale applied

Frame Rate Divider

Prescale Modulus

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PM7	PM6	PM5	PM4	PM3	PM2	PM1	PM0	PSR	WL1	WL0	DC4	DC3	DC2	DC1	DC0

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

BBP

BBPCRB

BBP Control Register B
Address = X:$FFA7
Reset = $0000
Read/Write

RIE	Description
0	Disable interrupt
1	Enable interrupt when a word is received

TLIE	Description
0	Disable interrupt
1	Enable interrupt for last transmit time slot

RLIE	Description
0	Disable interrupt
1	Enable interrupt for last receive time slot

TEIE	Description
0	Disable interrupt
1	Enable interrupt for transmit error

REIE	Description
0	Disable interrupt
1	Enable interrupt for receive error

TIE	Description
0	Transmit Interrupt disabled
1	Transmit Interrupt enabled

RE	Description
0	Receive disabled
1	Receive enabled

TE	Description
0	Transmit disabled
1	Transmit enabled

RCIE	Description
0	Disable interrupt
1	Enable receive interrupt

TCIE	Description
0	Disable interrupt
1	Enable transmit interrupt

RCE	Description
0	Counter disabled
1	Enable receive counter

TCE	Description
0	Counter disabled
1	Enable transmit counter

Serial Output Flags

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
REIE	TEIE	RLIE	TLIE	RIE	TIE	RE	TE	RCIE	TCIE	RCE	TCE	* 0	* 0	OF1	OF0

* = Reserved,

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

BBP

BBPCRC

BBP Control Register C
Address = X:$FFA8
Reset = $0000
Read/Write

CKP	Description
0	Transmit – bit clock rising edge Receive – bit clock falling edge (default)
1	Transmit – bit clock falling edge Receive – bit clock rising edge

SCKD	Description
0	External clock source
1	Internal clock source

SCD2	Description
0	SCD2 pin is input
1	SCD2 pin is output

SCD1	Description
0	SCD1 pin is input
1	SCD1 pin is output

SCD0	Description
0	SCD0 pin is input
1	SCD0 pin is output

MOD	Description
0	Normal mode
1	Network mode

SYN	Description
0	Asynchronous mode
1	Synchronous mode

SHFD	Description
0	Data shifted out MSB first
1	Data shifted out LSB first

FSL1	FSL0	Description
0	0	WL bit clock for both TX and RX
0	1	1-bit clock for TX WL bit clock for RX
1	0	1-bit clock for both TX and RX
1	1	WL bit clock for TX 1-bit clock for RX

FSR	Description
0	Frame sync occurs with first bit of current frame
1	Frame sync occurs with last bit of previous frame

FSP	Description
0	Positive frame sync
1	Negative frame sync

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FSP	FSR	FSL1	FSL0	*	*	*	*	SHFD	CKP	SCKD	SCD2	SCD1	SCD0	MOD	SYN
				0	0	0	0								
				$0											

∗ = Reserved,

Application: ______________________________ Date: ______________

______________________________ Programmer: ______________

BBP

BBPSR

BBP Status Register
Address = X:$FFA9
Reset = $0000
Read/Write

TUE	Description
1	TX underrun occurred

TFS	Description
1	TX frame sync occurred

ROE	Description
1	RX overrun occurred

RFS	Description
1	RX frame sync occurred

TDE	Description
1	TX data register empty

Serial Input Flags

RDF	Description
1	RX data register has data

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	RDF	TDE	ROE	TUE	RFS	TFS	IF1	IF0
0	0	0	0	0	0	0	0								
$0				$0											

* = Reserved,

Application: ______________________________ Date: ____________________

______________________________ Programmer: ____________________

BBP

BBPRX

BBP Receive Data Register
Address = X:$FFAA
Reset = $0000
Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
High Byte								Low Byte							

BBPTSR

BBP Time Slot Register
Address = X:$FFAB
Reset = $0000
Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Dummy Register, Written During Inactive Time Slots															

BBPTX

BBP Transmit Data Register
Address = X:$FFAC
Reset = $0000
Read/Write

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
High Byte								Low Byte							

Application: ____________________ Date: ____________________

____________________ Programmer: ____________________

BBP

BBPPDR

BBP Port Data Register
Address = X:$FFAD
Reset = $0000
Read/Write

Port Data Bits

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	*	*	BBPPD5	BBPPD4	BBPPD3	BBPPD2	BBPPD1	BBPPD0
0	0	0	0	0	0	0	0	0	0						
$0				$0											

PEN	Description
0	Port pins are tri-stated
1	Port pins enabled

BBPPCR

BBP Port Control Register
Address = X:$FFAF
Reset = $0000
Read/Write

BBPPCn	Description
0	Pin configured as GPIO
1	Pin configured as SAP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	PEN	*	BBPPC5 (STDA)	BBPPC4 (SRDA)	BBPPC3 (SCKA)	BBPPC2 (SC2A)	BBPPC1 (SC1A)	BBPPC0 (SC0A)
0	0	0	0	0	0	0	0		0						
$0				$0											

BBPDDR

BBP Data Direction Register
Address = X:$FFAE
Reset = $0000
Read/Write

BBPDDn	Description
0	Pin is input
1	Pin is output

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	*	*	BBPDD5	BBPDD4	BBPDD3	BBPDD2	BBPDD1	BBPDD0
0	0	0	0	0	0	0	0	0	0						
$0				$0											

* = Reserved,